


Supporting Digital Twins Systems Integrating the MERODE Approach

Ivan Compagnucci 

University of Camerino

Department of Computer Science
Camerino, Italy


ivan.compagnucci@unicam.it

Monique Snoeck 

KU Leuven

Faculty of Economics and Business
Leuven, Belgium

monique.snoeck@kuleuven.be

Estefanía Serral Asensio 

KU Leuven

Faculty of Economics and Business
Leuven, Belgium

estefania.serralasensio@kuleuven.be

Abstract—Digital Twins (DTs) have emerged as a promising concept, representing a new paradigm in smart systems engineering. However, the development of DTs pose significant challenges, particularly on the modeling and data management aspects. This paper presents a holistic approach to develop a DT implementation using a domain model derived from the standard SSN/SOSA ontologies. The proposed approach leverages the MERODE modeling method for rapid prototyping and validation, guaranteeing robust model quality checks and enabling the creation of flexible and reusable code adaptable to different architectural settings. Data management is enhanced through the use of the SOSA/SSN ontology, enabling semantic data models and inference strategies to achieve semantic interoperability. By combining MERODE and IoT ontologies, our approach aims to take a step forward in addressing the challenges linked to system complexity and data management in the development of DT. The proposed approach was validated by instantiating the proposed model with a real-life case.

Index Terms—Conceptual modeling, Model-Driven Engineering, Digital Twins, Ontology, Internet of Things, Model-Based Software Engineering

I. INTRODUCTION & MOTIVATION

Digital Twins (DTs) have emerged as a prominent concept, representing a new paradigm in the field of smart systems engineering. A DT refers to a virtual replica or simulation of a physical object, process, or system that enables real-time monitoring, analysis, predictive maintenance, and optimization [18]. It offers enhanced system management capabilities by leveraging real-time data obtained from the components of the system. Demonstrating their efficacy across many application domains, DTs have established themselves as an indispensable element within the Model-Based Systems Engineering (MBSE) field [4]. Moreover, adopting a Model-Driven Engineering (MDE) approach in the development of DT, is crucial to fully leverage their potential [19].

While the benefits of DT have been demonstrated in many areas, their development, operation, and evolution pose significant challenges. In fact, at present, no tool or platform can fully support the development of a DT [13]. The development of digital twin-based systems requires a holistic systems engineering approach where the modeling and data management aspects are critical [18]. Considering the complexity of IoT systems, maintaining a consistent system structure is a non-trivial task. Part of such an issue can be addressed from the

perspective of MDE [1] using ad-hoc domain models. These models act as the foundational building blocks for developing robust information systems, providing a solid foundation for capturing and organizing domain-specific knowledge [11].

In this paper, to facilitate the holistic development of a DT for an IoT system, we have derived a domain model based on a standard IoT ontology. The domain model has been designed to be integrated as a module within “regular” information systems, so as to facilitate enriching these systems with DT features. We adopt the MERODE modeling method, which enables rapid prototyping and validation of the model using real-life cases. MERODE further offers the advantages of thorough model quality checks, and a layered approach which allows for the required flexibility and reusability of the code in different architectural settings [15]. From the data management perspective, ontologies offer a relevant approach to achieving semantic interoperability, as they enable the definition of semantic data models combined with domain knowledge, as well as the formulation of inference strategies [8], [11]. By leveraging the advantages offered by combining MERODE and IoT ontologies, our intention is to move a step forward in addressing the system complexity and data management challenges involved in the development of IoT systems and the corresponding DT.

The paper is structured as follows. In Section II, we present a background on MDE and data management in DTs. The section concludes with an analysis of some standard ontologies for the IoT. Section III provides an overview of the research methodology. Section IV briefly introduces the MERODE approach. Section V presents the development of a DT by combining MERODE and the SSN/SOSA ontology. In Section VI we present the evaluation of the approach based on a real-world use case. Finally, in Section VII, we present the conclusions and future work.

II. BACKGROUND & RELATED WORK

This section presents the evolution of modeling in systems engineering from craft-based development to the new paradigm of DTs. Then, it provides an overview of relevant works that have explored the application of data management techniques for IoT. Finally, we will discuss two of the most important ontologies for IoT, namely the Semantic Sensor

Network (SSN) and the Sensor, Observation, Sample, and Actuator (SOSA) Ontology.

A. Model-Driven Engineering in Digital Twins

In the context of DT, MDE aims to create a system that effectively captures the behavior, structure and relationships of IoT system components [2]. By representing the digital version of the IoT system through a model, it is possible to simulate, analyze and optimize the system's performance without directly manipulating the physical system itself. A study from [18] explored the evolution and adoption of modeling in systems engineering from craft-based development to the adoption of more sophisticated models as DTs. In this subsection, we present an adapted perspective on this study by shifting the focus towards the development of software systems, rather than physical artifacts. Figure 1 shows the evolution of modeling and engineering applied to DT.

In the early days of craft-based software development, people relied on practical knowledge and trial-and-error to develop software solutions. As complexity increased and communication became important, model-based development emerged, using sketches and designs to guide production. The adoption of MBSE improved product quality and communication within development teams, facilitating advancements in engineering [18]. However, using sketch and blueprint models does not allow to create a strong and formal link between the model, a physical object and its digital representation.

Model-driven engineering uses executable models to automatically generate software and/or digital objects without human intervention. The models thus establish a formal link between the design and the generated software [18]. This approach allows for easy updates and adaptations by modifying the model rather than the software directly. In MDE, models are used as blueprints for software development and they rely on a *Metamodel* which defines the vocabulary used in the input models (i.e., abstract syntax, concrete syntax, well-formedness rules, and semantics). The MDE process involves the generation of multiple intermediate models using generators, which are then utilized to produce a final artifact (i.e., software and/or digital objects).

With model-driven engineering, the connection between the model and the physical artefact is typically lost once the physical artifact is modified. As a result, the software that represents the physical artifact does not remain synchronized with it. In recent years, the concept of the DT has emerged to address this limitation [18]. DT represents a next stage, where the physical object is continuously connected to its digital counterpart throughout the entire lifecycle. A key concern in DT is ensuring the continuous near-real-time synchronization of the state between the DT and the physical artifact. This enables a consistent and up-to-date representation between a tangible entity and its digital replica facilitating real-time monitoring, simulation, and testing of the entity's performance. As a result, this integration empowers stakeholders to make informed decisions and drive operational optimization by harnessing the insights derived from the DT.

B. Data Management in Digital Twins

According to [17], a DT adheres to a five-component architecture described as a five-element tuple:

$$M_{DT} = \langle PE, VE, Service, Data, Connection \rangle$$

Where PE represents a *Physical Entity* of the real world; VE the corresponding *Virtual Entity*; *Service* the operations provided for both entities; *Data* the information produced, processed, and exchanged between these elements, and *Connection* indicates how the parties are linked. Figure 2 depicts a visual representation of these five dimensions.

The *Data* component serves as a point of ingestion of the original data from these systems and of return at the right time to direct the interactive optimization process resulting from the interaction between them [5]. Often, in DT systems, real-time data is extracted from sensors or actuators which are associated with the Physical Entity component. The corresponding Virtual Entity is then defined as a model that virtually replicate the characteristics and behavior of the Physical Entity. However, achieving a consistent mechanism for managing data across DT systems is a non-trivial task.

In 2022, a systematic literature review on data management solutions in the DT context was published [5]. This review analyzes challenges and key points about data in the context of DTs identifying issues, trends, and opportunities. According to [5], there are five well-known challenges that a DT system should properly address in the context of data management. Providing *Data integration* is required to combine data from multiple sources to provide a high-level unified view that makes the data available for analysis and use. Support for *Data heterogeneity* is required because of the significant semantic, terminological, and syntactic differences across different application domains. *Data Interoperability* is required to make data accessible, reusable, and understandable by all entities involved in the system. Finally, data should be easily accessible (*Data Search*) and trustworthy (*Data Quality*) so that value can be derived from it.

The survey shows that dealing with data integration, heterogeneity, interoperability, search, and quality are common issues observed in developing MDE systems for DTs. Moreover, the study highlights that the maturity level of data management solutions remains at an early stage, necessitating substantial efforts for enhancement.

In light of this, in [5], authors propose several techniques to address, at least partially, the challenges associated with data management in the DT context. Adopting a reference architecture that organizes data/information into distinct abstraction layers can enhance the quality of raw data. By leveraging the concept of "separation of concerns", providing high-quality data can be facilitated. Therefore, it is logical to seek a solution based on a well-defined conceptual modeling methodology, such as MERODE. To address the challenge posed by significant semantic, terminological, and syntactic differences across various application domains (*Data Heterogeneity*), the

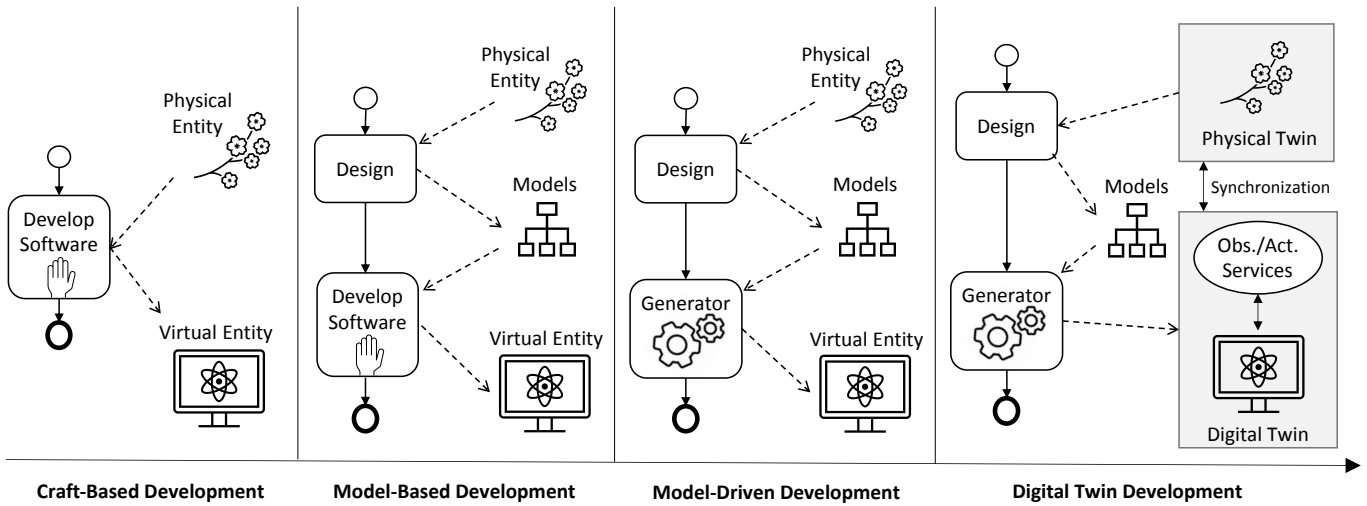


Fig. 1: The Evolution of Modeling from Craft-Based Development to Digital Twins Development.

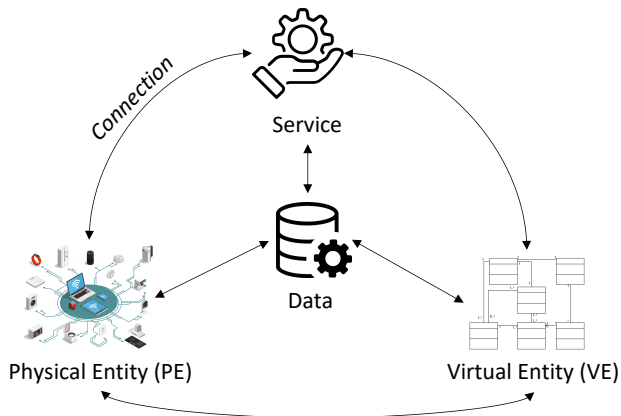


Fig. 2: The Five Dimensions of a Digital Twin.

adoption of a standardized ontology can provide an effective solution [5]. The utilization of a standardized ontology for representing entities in the context of DTs, such as sensors, power plants, and manufacturing, can effectively mitigate semantic heterogeneity, ensuring consistent comprehension of different or similar concepts regardless of modeling disparities [5]. Furthermore, ontologies enable a shared understanding and interconnectedness of concepts across various domains or disciplines, facilitating the *Data Interoperability* of DTs [12].

C. Ontologies for the Internet of Things

One of the most important aspects to develop DT is the capacity to mirror and interact with the physical environment. For that, IoT technology is essential, being the link between the physical and the digital world. Several ontologies have been developed to conceptualize and represent IoT. A well-known and generally accepted ontology for IoT, is the Semantic Sensor Network (SSN) Ontology designed by the W3C Incubator group. SSN provides a standardized framework for describing sensors, actuators, observations, features of

interest, observed properties, and more. By defining a common semantics and structure, SSN facilitates interoperability and enables seamless data exchange and collaboration among heterogeneous IoT systems. In addition, they offer researchers and practitioners a powerful tool for enhancing data management, knowledge representation, and decision-making processes.

SSN has served as the basis of many different ontologies, among others, the SOSA ontology, one of the most used ontologies for IoT systems. With its formal representation and standardized vocabulary, SOSA provides a comprehensive framework for modeling and interlinking devices' measurement, their associated metadata, and the physical entities involved in the IoT ecosystem [6]. As such, it supports a wide range of applications and use cases, including smart environments. The SOSA ontology encompasses over fifty distinct classes, facilitating a granular representation of various aspects of IoT data [6]. These classes are interconnected through more than a hundred object properties, forming a rich and expressive knowledge graph. The ontology has been designed to be efficient and compact, allowing for easy implementation and minimizing resource overhead on IoT devices. Another advantage of the SOSA ontology is its lightweight nature, making it highly suitable for resource-constrained IoT environments [10]. The widespread adoption of SOSA reflects its effectiveness in enabling seamless integration and interoperability among heterogeneous IoT systems.

For all of these reasons, we chose to reference both SNN and SOSA standard specifications for representing IoT entities in our approach. Their modular structure and class-based approach allow us to represent the different components of IoT systems in a detailed and interconnected manner, facilitating integration and interoperability among the components and services involved. However, the SSN and SOSA ontology models, *as-it-is*, are not “operational” or “ready for use” in the context of MDE systems. Additional steps are necessary to operationalize the SOSA ontology by transforming it into

a Platform-Independent Model (PIM) that can be effectively interpreted by an MDE system. To achieve this, the concepts from the ontology need to be mapped to appropriate classes, attributes and associations.

III. METHODOLOGY

In this section, we present an overview of the research steps we followed to operationalize SOSA to a PIM. The key steps and involved components are depicted in Figure 3 as an instantiation of the last column in Figure 1.

As a starting point, based on the SSN/SOSA ontology, we developed a generic data model representing the structural view of the domain model. Following the MERODE methodology, the data view is completed with default behavioural and object interaction views. In line with Design Science Research [7], we adopted an iterative approach, switching between build/design and evaluation through prototyping, aiming to continuously enhance the quality of the domain model. The evaluation step aimed at verifying 1) that the domain model was designed to be static and standardized, enabling its reusability in the development of various IoT systems and 2) that it can serve as executable model for the automated generation software using a code generator.

The final evaluation of the resulting domain model is done through the implementation of a real life case using the generated software. In the following sections, we present a brief introduction to MERODE (section IV), the model that was designed (section V) and the evaluation on the Leuven.cool case (section VI).

IV. THE MERODE APPROACH

In this section we present an overview of the MERODE approach, highlighting its main features, the domain model used to generate the Enterprise Layer (EL), System Service Layer (ISL), and related tools.

A. MERODE in a Nutshell

Over the years, various approaches have been developed to design and develop information systems, aiming to improve their effectiveness and alignment with organizational needs. One notable approach in this field is the MERODE approach [15]. The MERODE approach builds on UML, but uses only a subset of its constructs, complemented with proprietary notions, to enable the generation of fully functional code from conceptual models. In particular, the class diagram is refined to an “existence dependency” graph, and object interaction is formalized by means of the Communicating Sequential Processes (CSP) process algebra [14].

B. The MERODE Domain Model (EDG, OET, and FSMs)

A typical MERODE analysis or conceptualization consists of three views: a so-called Existence Dependency Graph (EDG) similar to a UML class diagram, a proprietary concept called Object Event Table (OET) defining object-event interactions, and a set of Finite State Machines (FSMs). We can refer to the set of these three models as the *Domain Model*.

Figure 4 depicts the three models describing the domain of people renting a car. The Existence Dependency Graph (EDG) is designed to define business object types (classes) and their associations. The EDG is represented by a UML class diagram, where all associations express existence dependency. Each class in the diagram is also associated with a State Chart (SC) or Finite State Machine (FSM).

Business event types, considered phenomena shared between the real world and the information system [9], are operationalized as call events. These events can trigger state changes in multiple business objects. The Object-Event table (OET) maps business event types to business object types, indicating the type of state change caused by each business event type: creation (C), modification (M), or ending (E). The OET helps identify the necessary operations for handling events in corresponding classes. In particular, the propagation rule specifies that a master object can always “see” the business events to which its dependents react, thereby participating in those events as well. Consequently, a single business event can trigger state changes in multiple business objects, allowing them to synchronize and interact through their joint participation in these events.

Finite State Machines (FSMs) define the behavior of objects by illustrating how events cause state transitions. Each object type follows a default lifecycle, involving creation triggered by any creation event (*C), multiple modifications triggered by modification events (*M), and transitions triggered by ending events (*E) that lead to the final state. More specific FSMs can be defined to add desired behaviors to the system.

Domain models can be created using the MERLIN tool¹, which also provides features for checking the consistency and readiness of the models for transformation.

C. The MERLIN Code Generator for MERODE

MERLIN Code Generator² is a standalone Java application that can be used to generate code from a MERODE domain model exported from MERLIN as *.mxp* file. Two different types of code can be generated: a Java prototype of the application, or a RESTful web application. Both solutions are based on the logic defined in the domain model. The Java prototype encompasses a *Domain Layer* comprising the database and business logic, and a series of default services that are accessible through a default Graphical User Interface (GUI). The GUI has one tab per object type defined in the domain model, displaying a list and details of the objects that exist in the system, and it offers buttons for triggering the business events, thus allowing to create, modify and end object instances. Furthermore, the prototypes are enhanced with feedback, both textual and graphical. In particular, the models are embedded in the application, thus enabling the tracing of the prototype’s behavior back to the models and validating their semantic quality.

The RESTful web application consists of a Maven project, containing the same *Domain Layer* and offering a range of

¹<https://www.merlin-academic.com/>

²<https://merode.econ.kuleuven.be/CodeGeneration.html>

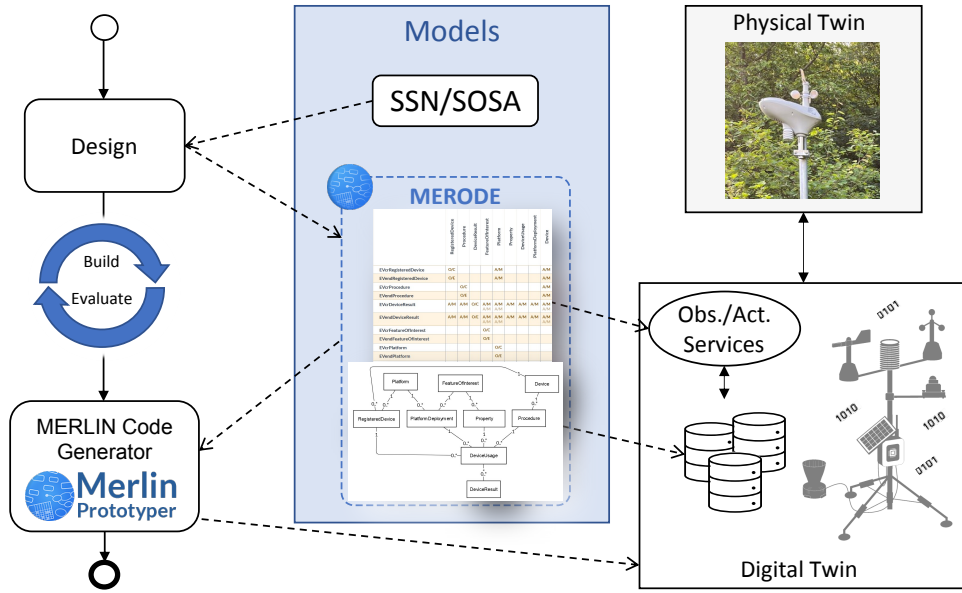


Fig. 3: Overview of the Proposed Approach.

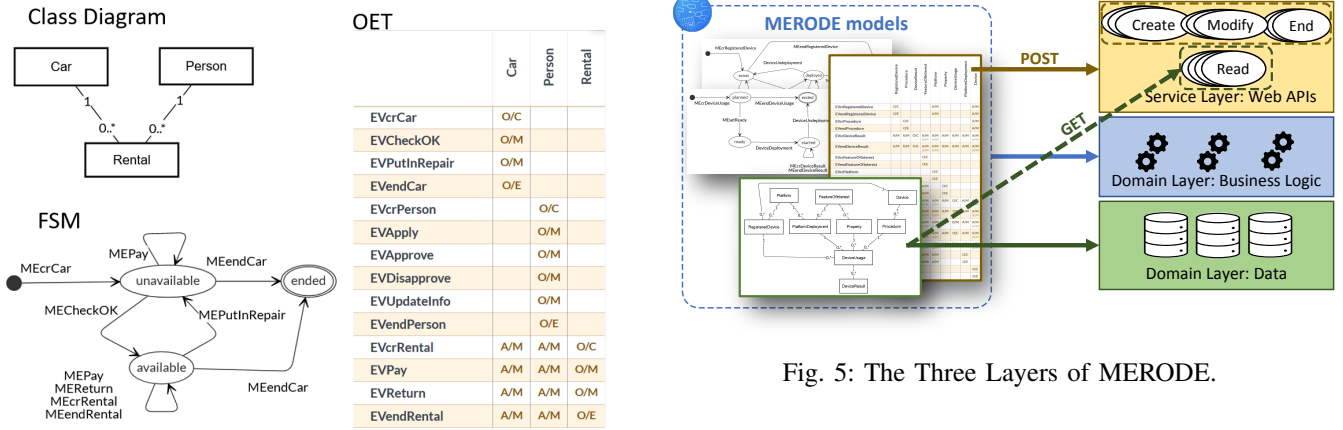


Fig. 4: The MERODE Domain Model.

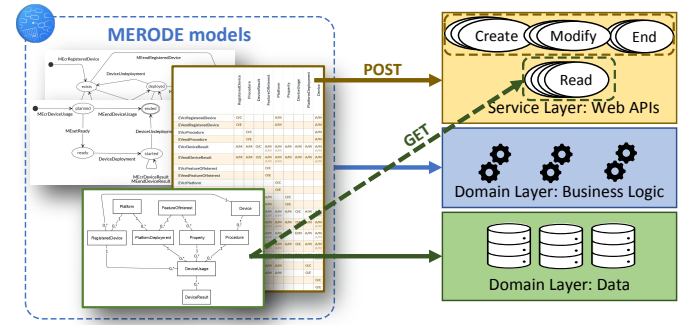


Fig. 5: The Three Layers of MERODE.

API services for each business event and for querying data from the system. Instead of triggering the business event types through buttons in the GUI, they can be invoked through POST API services. To retrieve information about the system's entities, GET requests can be made by directly referring to each instance of the objects. Figure 5 provides a visual representation helping to clarify the mapping of business events into API services.

The logic behind the creation of these APIs is orchestrated by the constraints, rules, and existing dependencies defined in the domain model. For example, if an object β has a dependency on the creation of another object α , the API to create the object β cannot be executed unless the object α has been created first. This ensures that the required dependencies are satisfied before proceeding with the creation of the object β . The approach proposed in this paper considers the RESTful

web application APIs for generating safe IoT system entities and controlled functionalities.

V. SUPPORTING DIGITAL TWINS SYSTEMS INTEGRATING THE MERODE APPROACH

This section delves into the mapping process through which we derived the domain model from the SSN/SOSA ontology. Then, we discuss constraints and changes applied to the domain model derived.

A. Mapping of SSN/SOSA Ontology to Domain Model

To create a PIM for IoT that can be effectively interpreted by a MDE system, we started with an in-depth examination of the SOSA ontology's standard specifications. To translate the necessary classes and association into a MERODE-model, we adopted an iterative approach. In each iteration, after completing a version of the model, we conducted tests to assess its effectiveness to represent various cases of IoT systems and to identify improvements. The first iteration started with the straightforward translation of the standard. The tests revealed

several deficiencies mostly related to the associations between the classes, and revealed ambiguities in the interpretation of SOSA concepts. We therefore made necessary modifications to the model in subsequent iterations, gradually enhancing its quality both in terms of model structure as definition of the classes. The final diagram is depicted in Figure 6.

Table I lists all the nine classes involved in our standard IoT EDG. In defining the model, we aimed to adhere closely to the standard, while at the same time generalizing certain aspects. In particular, we chose to consolidate and generalize certain classes from the standard into a single class. Out of these, five classes align precisely with the SSN/SOSA standard, namely: *Platform*, *Feature Of Interest*, *Procedure*, *Property* and *Result*. The classes *Device* and *Device Usage* serve as generalizations of one or more classes in the standard. Specifically, *Device* encompasses the *sosa:Sensor* and *sosa:Actuator* classes, while *Device Usage* encompasses the *sosa:Observation*, *sosa:Actuation*, and *sosa:Sampling* classes. Additionally, two classes introduced by us are *Registered Device* and *Platform Deployment*. The former is used for referring to a specific instance of a device in the platform, while the latter refers to a specific instance usage of the platform for a feature of interest.

Starting from the top, we have the class *Platform* which represents an entity on which IoT devices are hosted. The *Feature of Interest* class represents a real-world thing on which measurements take place. The *Device* class represents a component capable of sensing or actuating in an IoT system. The *Procedure* class represents a specific process or method followed by a device to generate a result of a measurement. It encapsulates the steps, actions, or algorithms involved in carrying out a particular task or operation. The *Property* class specifies a measurable characteristic associated with a feature of interest. The *Result* class represents the value of the output obtained from a measurement performed by a device. *Device Usage* class specifies the type of usage of a device to perform observations or actuations and when they are made.

B. Adding Multiple Path Constraints and Customized State

In some cases, objects are existence dependent on a master object. This means that an instance of such an object can be created if and only if an instance of the master class exists [15]. Sometimes an object type is indirectly dependent on the same master object type in two different ways (i.e. via multiple paths). For example, in Figure 6, both *Property* and *Platform Deployment* are existence dependent on the *Feature of Interest* class. Consequently, the *Device Usage* class is existence dependent on both *Platform Deployment* and *Property* and indirectly existence dependent on *Feature of Interest*. From a logical point of view, the purpose of the *Device Usage* class is to provide the means to specify the type of operation to be performed on a device, such as observation or actuation. However, for the safe execution of these operations, the *Device Usage* class relies on operational information obtained from the master classes it is connected to. In our model, we have established business rules to enforce

these requirements and ensure the secure execution of device operations. These rules dictate that an instance of the *Device Usage* class must adhere to three indirectly multiple path constraints depicted in Figure 7.

As an example, we consider the second rule. It states that the *Feature of Interest*, (i.e., a private garden), must be consistent across both the derived *Property* (i.e., relative humidity of that specific garden) and the corresponding *Platform Deployment*. In this context, consistency means that the instance of the *Property* class and the associated *Platform Deployment* must both reference the same instance of the *Feature of Interest* class. This constraint is explicitly expressed within the *Device Usage* object: `self.Property.FeatureOfInterest == self.PlatformDeployment.FeatureOfInterest`. By imposing this constraint, we ensure that each *Device Usage* object relates to only one Feature of Interest and that each event type of the *Device Usage* involves a single object of the Feature of Interest.

Additionally, we have refined the lifecycle of the *Registered Device* and *Device Usage* entities. The “Device Usage” class enables the deployment of a device. By creating an instance of the “Device Usage” object, the *planned* state is achieved. Next, a modifying method “setReady” is executed and the *ready* state is reached, indicating that the device is ready for use. Finally, by executing the “DeviceDeployment” event, the device starts to perform the measurement. In the case of the “Registered Device” FSM, after creating an instance of the object, the “DeviceDeployment” method can be executed to achieve the *deployed* state. If you want to deploy the same device for other measurements, it is required to return to the initial state by executing the “DeviceUndeployment” method first, thereby ending the previous device usage.

VI. EVALUATION

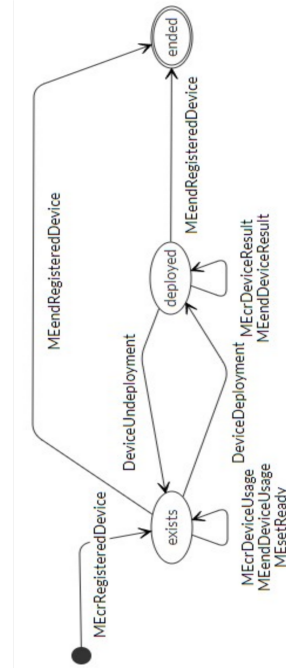
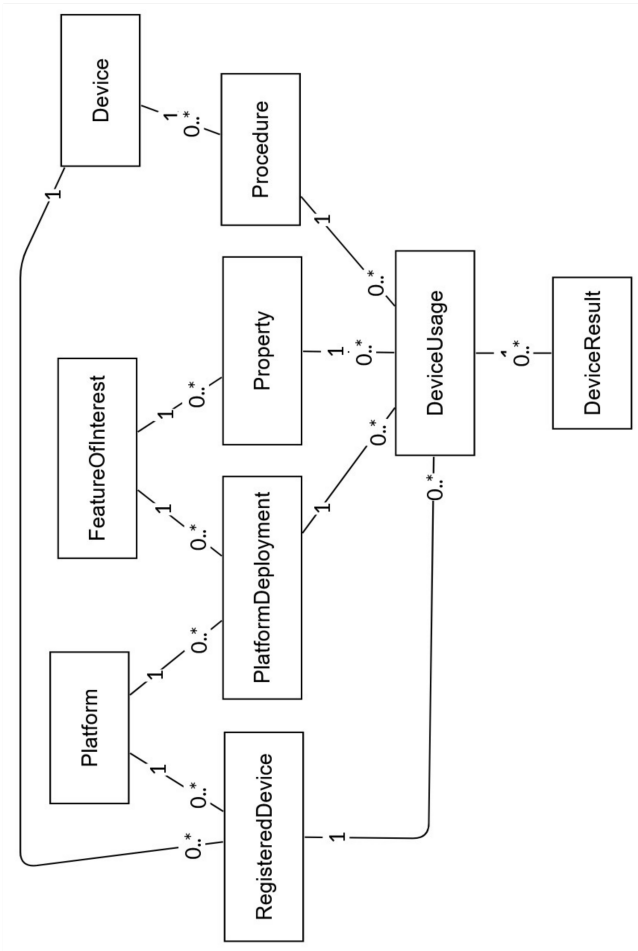
In this chapter, we first present a brief explanation of the use case on which the proposed approach was evaluated. Then, we check the usability of the approach focusing on its implementation and simulation. Finally, we acknowledge a set of limitations regarding the proposed approach.

A. Use Case: Leuven.cool

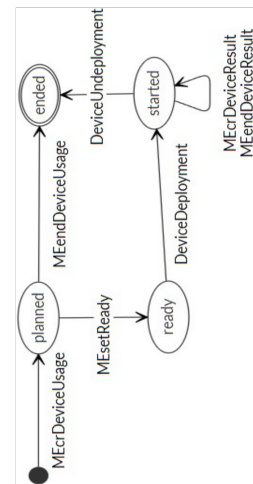
The Leuven.cool project, conducted by KU Leuven in collaboration with the climate NGO Leuven2030, Leuven City, and the Royal Meteorological Institute (KMI), aims to examine the urban microclimate of Leuven [3]. The project is centered around understanding how natural elements can help reduce negative impacts. To achieve this, weather stations have been installed in both private gardens and public locations throughout Leuven to measure the microclimate. Figure 8 depicts the network of the weather stations in Leuven and its stations.

The research project provides valuable insights into the optimal spatial configuration of green and blue elements, enabling managers and decision-makers to effectively plan and manage urban green (plants) and blue (water) areas to maximize their cooling effects. Each area is equipped with a weather station

	Device	PlatformDeployment	DeviceUsage	Property	Platform	FeatureOfInterest	DeviceResult	Procedure	RegisteredDevice
EVcrRegisteredDevice	A/M				A/M			O/C	O/C
EVendRegisteredDevice	A/M				A/M			O/E	
EVcrProcedure	A/M						O/E		
EVendProcedure	A/M						O/E		
EVcrDeviceResult	A/M			A/M	A/M	A/M	O/C	A/M	A/M
EVendDeviceResult	A/M			A/M	A/M	A/M	O/E	A/M	A/M
EVcrFeatureOfInterest	A/M					O/C			
EVendFeatureOfInterest	A/M					O/E			
EVcrPlatform	A/M				O/C				
EVendPlatform	A/M				O/E				
EVcrProperty	A/M			O/C					
EVendProperty	A/M			O/E					
EVcrDeviceUsage	A/M			A/M	A/M	A/M	A/M	A/M	A/M
EVendDeviceUsage	A/M			A/M	A/M	A/M	A/M	A/M	A/M
DeviceDeployment	A/M			A/M	A/M	A/M	A/M	A/M	A/M
DeviceUndeployment	A/M			A/M	A/M	A/M	A/M	A/M	A/M
EYsetReady	A/M			A/M	A/M	A/M	A/M	A/M	A/M
EVcrPlatformDeployment	A/M			A/M	A/M	A/M	A/M	A/M	O/C
EVendPlatformDeployment	A/M			A/M	A/M	A/M	A/M	A/M	O/E
EVcrDevice									O/C
EVendDevice									O/E



RegisteredDevice FSM



DeviceUsage FSM

Fig. 6: The MERODE Domain Model derived from the SSN/SOSA Ontology.

Derived EDG	SSN/SOSA Ontology	Description
Platform	sosa:Platform	A Platform is an entity that hosts other entities, particularly Sensors, Actuators, and other Platforms.
Feature Of Interest	sosa:FeatureOfInterest	The thing whose property is being estimated or calculated in the course of an Observation to arrive at a Result.
Device	sosa:Sensor sosa:Actuator	Device, agent (including humans), or software (simulation) involved in, or implementing, a Procedure. A device that is used by, or implements, an (Actuation) Procedure that changes the state of the world.
Procedure	sosa:Procedure	A workflow, protocol, plan, algorithm, or computational method specifying how to make an Observation, an Actuation, or a Sampling.
Property	ssn:Property	A quality of an entity. An aspect of an entity that is intrinsic to and cannot exist without the entity.
Device Usage	sosa:Observation sosa:Actuation	Act of carrying out a Procedure to estimate or calculate a value of a property of a Feature Of Interest. An Actuation carries out an (Actuation) Procedure to change the state of the world using an Actuator.
Result	sosa:Result	The Result of an Observation, Actuation, or act of Sampling.
Registered Device	✗	This class allows for the registration to a specific instance of a Device in a specific Platform.
Platform Deployment	✗	This class allows to align a specific instance of Feature of Interest within a specific Platform.

TABLE I: Mapping Between the SSN/SOSA Ontology and the EDG Domain Model.

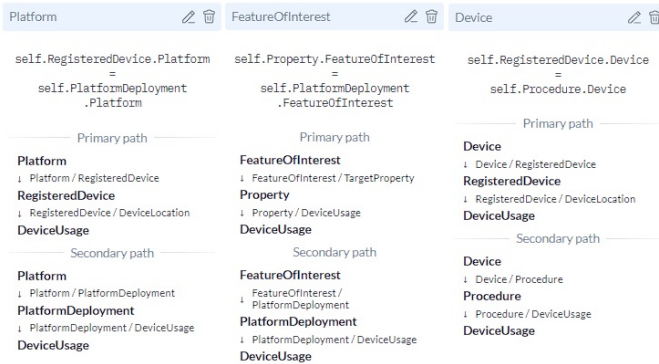


Fig. 7: Multiple Path Constraints in Device Usage.

that includes various sensors, such as temperature, humidity, wind speed, UV radiation, and precipitation. To access real-time urban climate data, it is possible to visit the Leuven.cool website³, which provides access to the map of weather stations and where each dot represents a DT of a garden or public area.

B. Feasibility of the Approach

To assess the feasibility of our approach, we tested the usability of the derived domain model by populating it with instances representing a weather station of the Leuven.cool project. To discuss how the data management works, we consider the five components of a DT architecture, discussed in Section II-B. The domain model derived from the SSN/SOSA ontology, the java application prototype, and the web application are available here⁴.

Physical Entity (PE). The physical entities in this scenario encompass the weather station itself, along with its set of sensors, and the surrounding environment. Specifically, within the Leuven.cool project, a weather station consists of various sensors designed to monitor a range of weather phenomena. These include a temperature sensor, a humidity sensor, two wind sensors for measuring wind speed and direction, a radiation sensor, and a UV sensor.

Virtual Entity (VE). Virtual entities are represented and described by the domain model derived from the ontology.

³<https://leuven.cool/>

⁴<https://anonymous.4open.science/r/ModDIT-23>

A minimal representation of the instances in each class of the model is depicted in Figure 9. To begin, we established instances of the top-level classes: *Platform*, *Feature of Interest*, and *Device*. We defined a general platform instance on which devices would be hosted, designated the garden as a Feature of Interest for which measurements would be collected, and created a collection of sensors as instances of the Device class. Next, we proceeded to create instances for the second-level classes: *Procedure*, *Property*, *Platform Deployment*, and *Registered Device*. We generated five instances of the *Property* class to represent measurable attributes of real-world objects (e.g., temperature, relative humidity, wind speed, wind direction, solar radiation, UV index), and an additional five instances for *Procedure* to describe the operational methods of the devices. Instances of *Platform Deployment* and *Registered Device* were created by referring to the master object instances. Subsequently, by creating an instance of the *Device Usage* class, it became possible to set up the deployment of a device by referencing the instances of the second-level classes. Finally, for each device, we defined the *Result* class containing attributes related to the values measured by the device.

Service. The services provided in this approach consist of a set of APIs generated in the web application. The entities involved in the IoT system can perform a set of actions by directly executing an API to trigger a business event or inspect data. The execution of these APIs is orchestrated following the rules and constraints defined during the design of the domain model.

Data. Data management in the context of MERODE is based on interconnected object types, which are effectively addressed by the EDG domain model. To cater to the persistence of data, the EDG model is transformed into a database schema. In this process, each object type will by default be mapped to a corresponding database relation. To cater to proper implementation, associations will by default be converted to foreign keys referring to the ID of the corresponding master object type. API services rely on the database schema to ensure compliance between services and data.

Connection. The bi-directional communication between the Physical and Digital entities is provided using of API services. The API-oriented communication mechanism allows for reliability and robustness in communications since business



Fig. 8: The Leuven.cool Network and its Weather Stations.

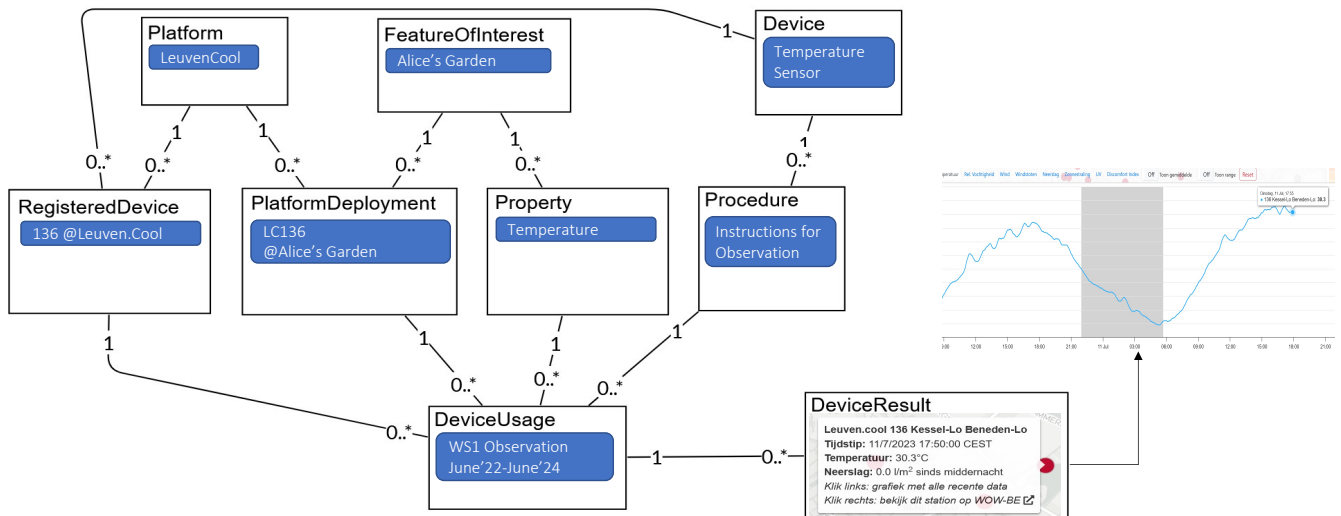


Fig. 9: Minimal Example of Instances Allocation in the EDG Domain Model.

rules of entities (business events) have been already managed in the domain model. Thus, through the web application and APIs, users can interact with both physical and digital entities. For instance, it is possible to initiate pull operations, such as measuring the temperature of a garden, and then the web application retrieves data from the physical environment and presents it to the user in a digital format. On the other hand, also push operations are allowed, since users can trigger actions that have an impact on the physical world. For example, users can utilize the web application’s API to send a command for turning off a device in the garden.

C. Limitations

Currently, this study has some limitations concerning the domain model derived from the SSN/SOSA ontology and regarding some technical aspects.

Firstly, the attributes in the EDG domain model still need to be defined. The EDG model was derived by mapping classes from the SSN/SOSA ontology, without considering yet the

inclusion of attributes. In the current implementation, only a single generic attribute “name/id” has been provided in each class, to allow for simulating and evaluating the proposed approach. In addition, the bidirectional communication between the physical and digital entities can only occur through the manual execution of API services, which poses a constraint in the automation of communication processes.

Lastly, the proposed approach allows for an “out of the box” representation of a DT system, without considering the actual technical implementation of the system. A full-fledged implementation of a DT will require more complex details than those described in this study.

VII. CONCLUSION & FUTURE WORK

In this work, we have introduced an approach that leverages the MERODE modeling method along with a generic domain model derived from the SSN/SOSA ontology to take a significant step forward in engineering DT systems. Our approach demonstrates the effectiveness of leveraging a domain model

from standard ontologies in specifying the digitalization of IoT technologies. By employing the domain model, we can accurately represent and describe the digital aspects of IoT technologies, facilitating their seamless integration into DT systems. This solution enables the verification and monitoring of expected behavior with minimal computational costs and development efforts. The strength of our approach lies in the modeling method provided by MERODE, which allows for rapid prototyping and validation using real-life cases. In addition, ontologies facilitated the definition of semantic data models combined with domain knowledge and the formulation of inference strategies, further enhancing the effectiveness of our approach. This promoted data integration and interoperability, allowing for seamless management of heterogeneous data into DT systems. To evaluate the feasibility of our approach, we applied it to a real-world use case from the Leuven.cool project. Through the use of web APIs, we achieved manual bi-directional communication between physical and digital entities, effectively replicating real-world actions in the digital realm.

However, implementing full-fledged DTs can be a non-trivial task. While our approach does offer robust software engineering methods and practices to ensure the effectiveness and reliability of data exchange, there is still ongoing work required to address technical implementation aspects within the scope of our study.

We have identified several areas of focus that we intend to address in the near future. One of our goals is to enhance the usability of the RESTful web application by implementing a web interface.

Then, we plan to replace the current manual communication with bidirectional automated communication between physical and digital entities. The study conducted by [16], provides evidence of the effectiveness of integrating MERODE with BPMN and DMN, resulting in a modeling approach that is process, data, and decision-aware. The study also demonstrates the successful operationalization of this approach, resulting in an executable process and decision system. This integration enables the realization of an integrated view of data and processes, ensuring enhanced coherence and alignment between operational activities and domain concepts. Building upon these findings, our objective is to automate the communication between physical and digital entities.

Finally, we plan to focus on implementing the attributes in the EDG domain model, using an iterative approach that involves testing and applying improvements to the model. By employing this iterative methodology, we aim to achieve a more comprehensive and refined representation of the ontology, enhancing its accuracy in describing and representing the real-world aspects within the context of a DT.

REFERENCES

[1] van der Aalst, W.M.P., Hinz, O., Weinhardt, C.: Resilient digital twins. *Bus. Inf. Syst. Eng.* **63**(6), 615–619 (2021)

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

[3] Beele, E., Reyniers, M., Aerts, R., Somers, B.: Replication Data for: Quality control and correction method for air temperature data from a citizen science weather station network in Leuven, Belgium (2022)

[4] Borky, J.M., Bradley, T.H.: *Effective Model-Based Systems Engineering*. Springer (2019)

[5] Correia, J.B., Abel, M., Becker, K.: Data management in digital twins: a systematic literature review. *Knowl. Inf. Syst.* **65**(8), 3165–3196 (2023)

[6] Haller A., Janowicz K., Cox S., Le Phuoc D., Taylor K., Lefrançois M.: *The Semantic Sensor Network Ontology* (2017), online: <https://www.w3.org/TR/vocab-ssn/> (Accessed on 20 July 2023)

[7] Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. *MIS Quarterly* **28**(1), 75–105 (2004)

[8] Huang, Y., Dhoub, S., Medinacelli, L.P., Malenfant, J.: Enabling semantic interoperability of asset administration shells through an ontology-based modeling method. *MODELS '22*, New York, NY, USA (Nov 2022)

[9] Jackson, M.: The world and the machine. In: *Proceedings of the 17th International Conference on Software Engineering*. p. 283–292. ICSE '95, Association for Computing Machinery, New York, NY, USA (1995). <https://doi.org/10.1145/225014.225041>, <https://doi.org/10.1145/225014.225041>

[10] Janowicz, K., Haller, A., Cox, S.J.D., Phuoc, D.L., Lefrançois, M.: SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *J. Web Semant.* **56**, 1–10 (2019)

[11] Munir, K., Sheraz Anjum, M.: The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics* **14**(2), 116–126 (2018)

[12] Platenius-Mohr, M., Malakuti, S., Grüner, S., Goldschmidt, T.: Interoperable Digital Twins in IIoT Systems by Transformation of Information Models: A Case Study with Asset Administration Shell. In: *Conference on the Internet of Things, IoT 2019*. pp. 2:1–2:8. ACM (2019)

[13] Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., Wang, L., Nee, A.Y.C.: Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems* **58**, 3–21 (2021)

[14] Snoeck, M., Dedene, G.: Existence dependency: The key to semantic integrity between structural and behavioral aspects of object types. *IEEE Transactions on Software Engineering* **24**(4), 233–251 (1998)

[15] Snoeck, M.: *Enterprise Information Systems Engineering - The MERODE Approach*. The Enterprise Engineering Series, Springer (2014)

[16] Snoeck, M., Verbruggen, C., De Smedt, J., De Weerd, J.: Supporting data-aware processes with MERODE. *Software and Systems Modeling* (Mar 2023)

[17] Tao, F., Zhang, M., Nee, A.Y.C.: *Digital Twin Driven Smart Manufacturing*. Academic Press (2019)

[18] Tekinerdogan, B.: On the notion of digital twins: A modeling perspective. *Systems* **11**(1), 15 (2023)

[19] Wurm, B., Becker, M.C., Pentland, B.T., Lyytinen, K., Weber, B., Grisold, T., Mendling, J., Kremser, W.: Digital twins of organizations: A socio-technical view on challenges and opportunities for future research. *Commun. Assoc. Inf. Syst.* **52**, 26 (2022)