# SysML model-driven development for digital twins

Kanza Ouari[1], Malika Ioualalen[1], and Ahmed Hammad[2]

[1] MOVEP, USTHB University, Algiers, Algeria
{kouari, mioualalen}@usthb.dz
[2] Franche-Comte, University of Bourgogne, Besançon, France
ahammad@femto-st.fr

**Abstract.** A Digital Twin is a virtual replica of a physical system that is used to simulate its behavior, and provide services to improve its operation. However, their rigorous development is a difficult task and requires considerable efforts especially for their design. In this paper, we introduce a SysML model-driven development methodology for digital twins, a way to design digital twins at a high level of abstraction, automatically generate an essential part of their implementation, and verify their fidelity to their physical counterpart. In this methodology, the architecture followed is independent from application domains, implementation technologies, and purposes of using DTs. The physical system behavior is integrated to a SysML model in an operational language, and the fidelity verification is performed using a data lake. We explain with a case study from the literature how to implement a prototype of the architecture in the open source tool Eclipse. The case study indicates that a SysML model can facilitate the design of digital twins, improving thus their development process.

**Keywords:** Digital twin · Architecture · Model-driven development · SysML · Simulation · Fidelity.

## 1 Introduction

A Digital Twin (DT) was first introduced by Michael Grieves [1] in a course for product lifecycle management in 2003 as the "virtual digital representation equivalent to physical products". Today, the advancement of digital technologies such as machine learning, cloud computing, IoT, and Big Data has led to wide adoption of DT [2]. These technologies have enabled the creation of additional services to the original system (the Physical Twin, PT) using DT such as: remote PT monitoring, user alerts, self-adapting mechanisms, etc.

Currently, the development of Digital Twins (DTs) is a difficult task due to the lack of a standard methodology for their development and their increasing complexity, which stems from the complexity of their physical counterpart. Adding to this, a DT must simulate the physical counterpart at a high level of fidelity. According to [3], a DT must behave exactly like its PT. It is important to ensure

that the DT behaves like the PT before creating any additional service.
Model-Based Systems Engineering (MBSE) is a way to address this complexity. Its visual diagrams allow a comprehensive understanding of a system, as well as document its characteristics and enhance interdisciplinary communication among stakeholders [4]. In addition, the automated transformation of models into software implementation can improve productivity, reduce complexity, and facilitate the testing of complex system [5]. The System Modeling Language (SysML) [6] is a general-purpose modeling language used by MBSE. It enables capturing requirements, defining system structure, and describing system behavior. In the context of DT, visual diagrams created with SysML can be easily used for allowing a comprehensive understanding of DT, documenting its characteristics, and enhancing interdisciplinary communication among stakeholders. However, the automated transformation of SysML models into software implementation in general and DT implementation in particular is a challenge [7], [4]. The authors in [8] refer to this utilization as "the technical utilization" of SysML models. Another challenge is to use MBSE, particularly SysML, for testing purposes in a DT context [3]. In the context of DT, the DT's behavior must be faithful to the PT's, i.e., the behavior of the DT must emulate with sufficient precision the behavior of its PT [9]. This requirement must be taken into account during the development of DTs and verified before exploitating DTs.
To the best of our knowledge, we are not aware of any comprehensive work on the development of DTs by defining a general architecture, with a focus on the automatic transformation of SysML models into DT implementations and fidelity verification before exploiting DTs. Previous works [7], [4], [10] have concentrated on the automatic transformation of SysML models into DT implementations in a specific context without addressing fidelity verification or proposed a standard architecture using other modeling languages with an emphasis on fidelity verification. The combination of the automatic transformation of SysML models into DT implementations with fidelity verification using a general architecture is a new approach to addressing the challenge of developing DTs. The use of a general architecture for the development of DTs has advantages in terms of making the technology of DTs, which remains promising, easy to understand, and adaptable to various contexts, regardless of application domains, implementation technologies, and purposes of using DTs. By combining the SysML language and fidelity verification, it is possible to rigorously develop a DT by modeling its characteristics, automatically generating its implementation, and ensuring its fidelity to its physical counterpart.
In this paper, we propose a new methodology to facilitate the development of DTs, structured in three steps: design, implementation, and verification. In the design step, we use a general architecture based on a data lake to represent the system components, and we use a SysML model to capture high-level information about the structure and behavior of the DT. In the implementation step, the automatic generation of usable code for the DT is possible in an operational language by directly transforming the SysML model. The behavior of the DT is integrated into the SysML model using the same operational language. In the

verification step, the methodology verifies that the DT faithfully simulates the PT by exploiting the potential of the data lake. Figure 1 presents the steps of our methodology.

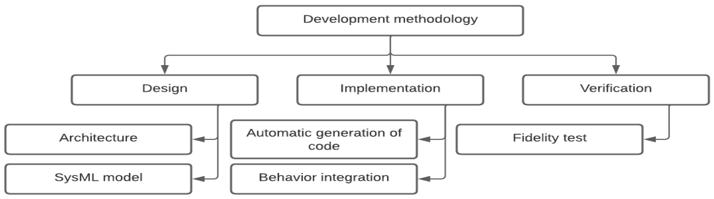

**Fig. 1.** Steps of the SysML model-driven development methodology.

The rest of this paper is structured as follows. After this introduction, Section 2 describes the background of our work. Next, Section 3 explicates the proposed DTs development methodology, and Section 4 explains the case study of a PT taken from the literature, use of the SysML model as a DT, creation of the data lake, and global implementation in Eclipse. Finally, Section 5 provides the related works and Section 6 presents the conclusion and an overview of future works.

## 2 Context

### 2.1 Digital Twin (DT)

A DT is a virtual copy that represents a physical system (its PT) and includes its characteristics. The virtual copy simulates the behavior of the PT through models and data. These models can be engineering models (e.g. CAD, Simulink) or software models (e.g. UML, SysML, MontiArc) [11]. The two twins are connected in a bidirectional way: the DT collects data from the PT and transfers it into services, and at the same time allows to interact with the PT and influence it.

Using a DT can offer many services for improving the PT operation [12]. For example, performing monitoring via real-time data collection to make better decisions and control the PT. A DT can also improve security and resilience by detecting malicious actions on a system. DT can predict future behaviors of the system and improve the process productivity using machine learning techniques. DT can also provide a testing platform for testing different scenarios to detect the most efficient one and to improve the system performances.

Additionally, the authors in [10] assert that DTs are software artifacts and, as such, require software engineering methods and practices, including rigorous

processes to enhance their quality and ensure their proper functioning. An important example of a test in this context is the fidelity test, which verifies that the traces produced by the two twins are similar [13], [9].
Although the meaning of DT may seem clear initially, the precise definition of a DT has been a topic of ongoing discussion and disagreement. Presently, there is no widely accepted and standardized definition for the term "DT". In our work, we consider a DT as a virtual representation that simulates the behavior of its PT. Simulation in our work represents the ability of DT to behave like its PT without providing any additional service. We define formally a concept of DT system as follows: "A system consisting of a DT connected in a bidirectional manner with its deployed PT. The purpose of this connection is to provide specific services for improving the operation of the PT. The entities involved in the system as software artifacts require specific tests to ensure that they behave as expected". In the design step of our methodology, we have defined an architecture that reflects our definition of a DT system. We consider the fidelity verification as a test to be performed on the behavior of the two twins, the objective is to have indications about the level of similarity between the behavior of the PT and the simulation of the DT.

### 2.2 System Modeling Language (SysML)

SysML is a general-purpose modeling language for systems engineering, based on UML. SysML reuses a subset of UML diagrams and extends it with new ones to model aspects of complex systems. It distinguishes three categories of diagrams: four structure diagrams (block definition diagram, internal block diagram, parametric diagram, package diagram), four behavior diagrams (state machine diagram, activity diagram, sequence diagram, use case diagram), and requirement diagram. Over these diagrams, in our work, we are interested in the block definition and package diagrams.
In the design phase of our methodology, we are interested in the Block Definition Diagram (BDD). The BDD is a structural diagram. It goes beyond the UML class diagram, which primarily focuses on classes and associations in an object-oriented context, by providing a more comprehensive structural representation. The BDD structures the system in blocks, sub-blocks and links. A block represents a complete system, a sub-system, or an element of a system. It can include values, properties, parts, references to other blocks, ports (properties specifying the types of interactions allowed between blocks), constraints (properties specifying constraints on other properties), and operations. Operations represent functionalities described in the behavioral part of the system. The use of the BDD has a dual objective: 1) Ensuring a comprehensive understanding of the DT structure. 2) Enabling the automatic transformation into DT implementation. In the case study section, we are interested in the Package Diagram (PD). The PD is a diagram that is used to organize and represent the various elements of a system model in a hierarchical manner. We use the PD to provide a visual representation of the possible modules that could be delivered from our proposal. It can help stakeholders to better understand the system.

## 3 SysML model-driven development methodology

### 3.1 Design

**Architecture:** To achieve our work, we defined a general architecture which reflects our definition of a DT system. It is composed of six components (see Figure 2): PT component, DT component, Data Lake component (DL), Connection Components (CC), Service Components (SC), and Analysis Components (AC). The PT component represents the deployed physical device. The DT component is the virtual copy of the PT component. The DL component plays the role of a storage space to manage data collected from the other components of the DT system. The CC represent possible interactions between the other components, this component is represented in the figure as arrows. The SC represent additional services and functionalities offered by the DT component to improve the operation of the PT component. The AC represent tests on the other software components of the architecture to ensure that they behave as expected. It's worth noting that all components of the architecture can interact with the DL component through CC, which are implemented by the architecture developer.

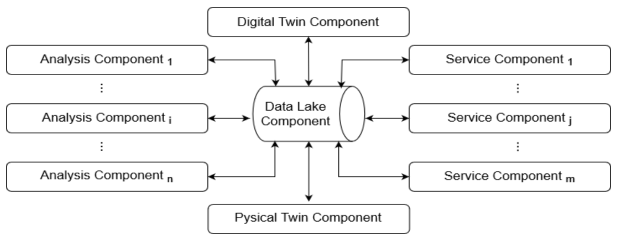

**Fig. 2.** Architecture for developing DT syetms.

The purpose of the methodology proposed in this work is to develop a DT that simulates the behavior of its PT and verify the fidelity, which requires the use of the PT component, DT component, DL component, a single analysis component, which we call Fidelity Test Component (FTC), CC with three interactions, between PT and DL, DL and DT, as well as between DL and FTC, without any service component. The restricted architecture that we are interested in is illustrated in Figure 3. It is composed of five components: PT, DT, DL, CC and FTC. We consider it as an initial and essential architecture for verifying the fidelity of DT before extending it with any service component.

**SysML model:** In our proposal, the developer works at a high level of abstraction by creating a SysML model as BDD to model the DT in a SysML modeling tool. Working at a high level of abstraction eliminates low-level details and takes only specific features of the PT, instead of taking the physical system as a whole, the BDD shows thus only the structural elements that interest a developer. The

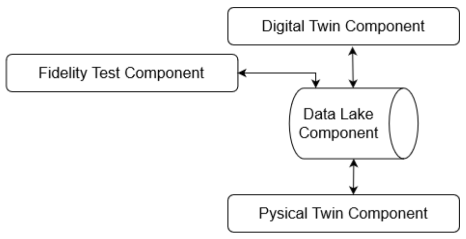

**Fig. 3.** Initial architecture for verifying DT fidelity.

resulting BDD is an abstraction of the PT that can include other elements required by the simulation that the DT performs. The behavior of the DT, which reflects the behavior of the PT, can be indicated by a list of operation in each BDD block at a high level, without going into the details of how the behavior is implemented. The behavioral part of the operations is integrated in the implementation step of our methodology as we will see in the next section.

### 3.2 Implementation

**Automatic generation of code:** After modeling the DT using the BDD, the developer automatically generates the SysML model implementation in an operational language. This activity automatically transforms the SysML model from a graphical format to a machine-interpretable textual format, or DT implementation. This allows enhancing the productivity, saving time and minimizing coding errors and mistakes. There are tools that provide interfaces allowing code generation, thus enabling the technical utilization of SysML. In this activity, a tool that allows code generation is needed.

**Behavior integration:** The behavior of the DT is completed by the developer manually in the same operational language by implementing the body of the operations modeled in the SysML model (BDD). This allows to obtain a DT that simulates the behavior of its PT. He also creates the DL in an appropriate database, and implements the CC that allow interaction between the other components. At this stage, the DL interacts with the PT that executes commands, and with the textual SysML model that simulates the behaviors of the PT.

### 3.3 Verification

The objective of the verification step is to ensure that the behavior of the PT and the DT are similar. To achieve this, the developer implements the FTC which uses the DL to measure the fidelity. It queries the database to retrieve the results produced following the execution of a command by the PT and the simulation of the DT. The same input data is fed into both the PT and DT, and then the traces of their behavior are compared to determine if there are any differences. At this stage, the developer also implements the CC that allows the interaction between the DL and FTC.

## 4 Case Study

### 4.1 Physical Twin (PT): Lego Mindstorms vehicle

This case study is taken from the literature [10]. The Lego Mindstorms vehicle, shown in Figure 4, consists of three captures: an ultrasound sensor to measure distances, a tactile sensor to detect collisions, and a light sensor to differentiate colors. The vehicle has a motor to advance and turn at a specific angle. It includes

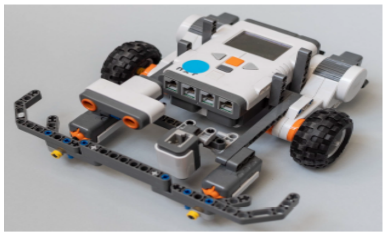

**Fig. 4.** Lego Mindstorms vehicle [10].

also a pose-provider to detect its position. The vehicle interacts with computers through a Bluetooth connection. The Lego software installs itself on a computer and controls it through various behaviors. This software represents the firmware used by the vehicle, and it is implemented in our case in the Java programming language. An example of a behavior to be simulated by the DT and verified by the FTC is the following: Lego follows a black line on the ground, if the line disappears, it continues to turn until it finds it.

### 4.2 Digital Twin (DT)

**Graphical SysML model** The BDD diagram is used as a SysML model to specify the physical vehicle structure in Papyrus tool with additional information required by the simulation. The BDD serves as the DT component in graphical format. Papyrus is a software system modeling tool developed by the Eclipse community. It allows software engineers to model software systems using UML and SysML. Papyrus is used as a modeling tool because it is open source and offers interfaces with other tools to directly transform SysML models. The BDD diagram used to model the vehicle is presented in Figure 5. The main block presented in the BDD diagram is called Vehicle. The other main blocks are Motor, UltrasonicSensor, LightSensor and TouchSensor. The signature of each operation is indicated in the block operation compartment.

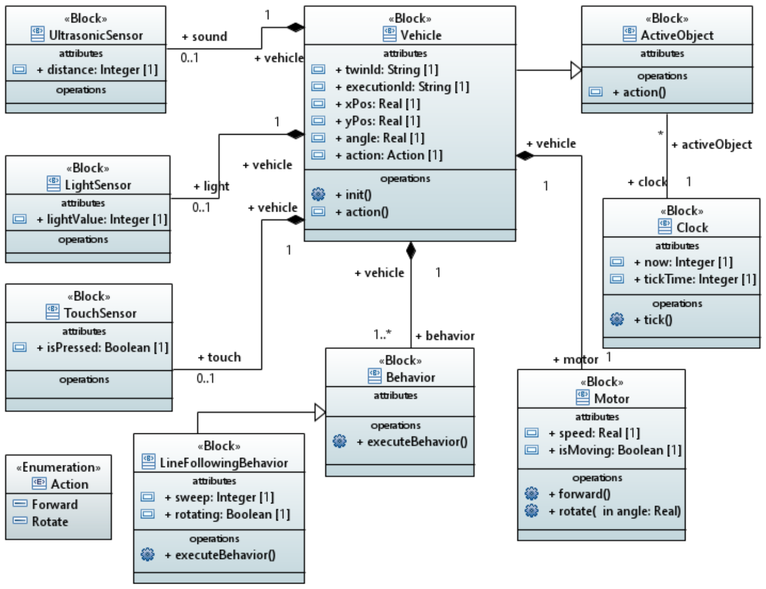

**Fig. 5.** Graphical SysML model of the DT of Lego.

**Textual SysML model** We have installed Papyrus in Eclipse to allow the direct exchange of SysML model, from a graphical model to a textual model, expressed as Java code. The Papyrus modeling tool features an interface that enables automatic code generation. We have chosen the Java programming language because, as mentioned before, the firmware of the vehicle uses Java. This activity presents the automatic generation of the code in the implementation step of our methodology. The code includes all the model information (block names, block properties, operation signatures and links between blocks), which ensures a lossless integration. This code is completed by the behavior of the vehicle and other behavior related to the simulation, by implementing the body of the operations. This process illustrates the behavior integration in the implementation step of the proposed methodology. To illustrate the relation between Papyrus and Eclipse, Figure 6 shows the SysML modeling in Papyrus, automatic generation of code in Eclipse, and manual integration of behavior.

## 4.3 Data Lake (DL)

In order to manage all data on the DT system and ensure the interaction between all components of the architecture, we used Neo4j to create the DL. Neo4j is a graph-oriented NoSQL database. It uses nodes and arcs to store data instead of tables as in relational databases. Neo4j allows complex queries, necessary when efficient data analysis is required through its expressive Cypher language, which is one of the reasons why we chose this database, since we want to ensure an

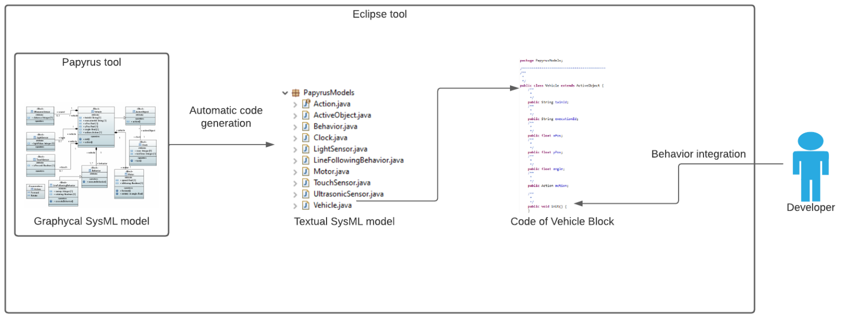

**Fig. 6.** The relation between Papyrus and Eclipse.

easy and efficient interaction with the storage space. In our work, we did not implement the FTC, because we did not synchronize the SysML model with the real vehicle, we only show how to implement a prototype of the architecture proposed in Eclipse to demonstrate our ideas. The DT framework that allows the SysML model to be connected to a physical system for simulation and fidelity test purposes can be implemented in Eclipse using the Java programming language by creating a project, named for example DigitalTwinFramework, with five packages. We have used the PD to illustrate these packages or modules in Figure 7. The required knowledge to implement this project includes SysML in Papyrus, advanced concepts in Java programming language such as collections, exceptions, threads, and sockets, as well as the ability to create and interact with a Neo4j database, a real system, and the FTC using Java language.

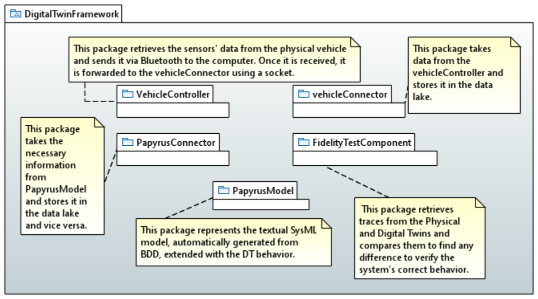
DigitalTwinFramework
This package retrieves the sensors' data from the physical vehicle and sends it via Bluetooth to the computer. Once it is received, it is forwarded to the vehicleConnector using a socket.
This package takes data from the vehicleController and stores it in the data lake.
VehicleController
vehicleConnector
This package takes the necessary information from PapyrusModel and stores it in the data lake and vice versa.
PapyrusConnector
FidelityTestComponent
PapyrusModel
This package represents the textual SysML model, automatically generated from BDD, extended with the DT behavior.
This package retrieves traces from the Physical and Digital Twins and compares them to find any difference to verify the system's correct behavior.

**Fig. 7.** PD showing how to implement a prototype of the initial architecture.

## 5 Related Work

We can link our work to several previous works conforming to three axes: 1) Architecture. 2) Automated transformation of models into DT implementation. 3) Fidelity verification.
Several works have described a possible architecture for developing DTs. [1] proposed an initial architecture with three dimensions: physical entity, virtual model, and connection. This same number of dimensions has been considered by other authors, such as [3]. Based on the initial architecture, works like [14], [15] have proposed five dimensions: physical entity, virtual model, services, data, and connection. [10] proposed an architecture with five components: physical entity, digital twin, data lake, analysis, and services. We have taken inspiration from this work and included the AC in our architecture. Generally, there are several architectures proposed in the literature, but they do not illustrate the possibility of using the DT for fidelity verification. In our work, we introduced a formal definition for DT systems in general and associated it with an architecture consisting of six components: PT, DT, DL, CC, SC, and AC. Our architecture is independent from application domains, implementation technologies, and purposes of using DTs. We also demonstrated how to restrict this architecture to simulate the PT behavior and verify the fidelity using five components: PT, DT, DL, DL, and FTC.
The automated transformation of models into DT implementation is already used by several authors. [11], [16], [17], [18] have used an architectural model of components and connectors in MontiArc with UML class diagrams specifying the data types of the objects exchanged between components as a starting point to facilitate the automatic generation of DTs, the behavior of DTs is integrated by a domain-specific language. In the context of using SysML, we have identified two works [4], [7] that use a SysML diagrams to design DT and transfer them directly into machine-recognizable code in a specific context, where part of the DT behavior is generated by the SysML state machine diagram, without following any architecture or separated steps. Our work differs from these works by two main novelties: 1) Our work uses SysML as a starting point for the systematic development of DTs in general by separating the architecture from SysML usage and implementation. 2) Our methodology requires the fidelity verification before integrating services to ensure the quality of DT.
Some works have implicitly considered fidelity verification, such as [3], [13], [19]. [3] mentioned the possibility of performing specific tests on the DT. [13] has proposed a method for validating the behavior of the two twins. [19] has shown the importance of defining DTs at different levels of abstraction for performing tests according to the properties to be validated. Concrete proposals for measuring the fidelity between the two twins are introduced in [10] [9]. These works use the UML language to introduce a model-driven methodology for testing DTs. In [10], a verification method similar to the one proposed in our work is treated. In [9], a bioinformatics algorithm is adapted to validate that the behavior of the two twins is the same or sufficiently similar. These works use class diagrams in a tool called USE [20] to model the DT at a high level of abstraction. The behavior

of the DT is integrated with an imperative language called SOIL [21]. Our work exploits the SysML language instead of the UML language to launch another research methodology for developing DTs by focusing on the fidelity verification before integrating additional services.

## 6 Conclusion

In this work, we have proposed a new methodology based on SysML model for systemically developing DTs. The methodology supports three main steps: design, implementation and verification. In the design step, we have adapted an independent architecture from application domains,implementation technologies, and purposes of using DTs. The DT is described using a BDD. In the implementation step, an essential part of the code was automatically generated. In the verification step, we are required to verify the fidelity of the DT before developing any other service component in order to ensure its quality in the future development steps. In the future directions of this work, we intend to synchronize the textual SysML model with the real vehicle. Also, we want to show how to model the FTC using a SysML model and automatically derive its code, following the process indicated in our methodology. In addition, we plan to analyze the traces of the two twins using the bioinformatics algorithm used in the literature. Additionally, we intend to use a requirement diagram to capture and model the requirement of our system in the design step. Also, we want to automatically generate a part of the behavior using the SysML behavior diagrams. Finally, we want to integrate other AC and SC to our initial architecture.

## References

1. M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, no. 2014, pp. 1–7, 2014.
2. A. Sharma, E. Kosasih, J. Zhang, A. Brintrup, and A. Calinescu, "Digital twins: State of the art theory and practice, challenges, and open research questions," *Journal of Industrial Information Integration*, p. 100383, 2022.
3. M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," *Transdisciplinary perspectives on complex systems: New findings and approaches*, pp. 85–113, 2017.
4. F. Wilking, C. Sauer, B. Schleich, and S. Wartzack, "Sysml 4 digital twins–utilization of system models for the design and operation of digital twins," *Proceedings of the Design Society*, vol. 2, pp. 1815–1824, 2022.
5. R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 37–54.
6. S. Friedenthal, A. Moore, and R. Steiner, "Omg systems modeling language (omg sysml™) tutorial," in *INCOSE international symposium*, vol. 18, no. 1. Wiley Online Library, 2008, pp. 1731–1862.
7. F. Wilking, C. Sauer, B. Schleich, and S. Wartzack, "Integrating machine learning in digital twins by utilizing sysml system models," in *2022 17th Annual System of Systems Engineering Conference (SOSE)*. IEEE, 2022, pp. 297–302.

8. H. A. Handley, W. Khallouli, J. Huang, W. Edmonson, and N. Kibret, "Maintaining the consistency of sysml model exports to xml metadata interchange (xmi)," in *2021 IEEE International Systems Conference (SysCon)*. IEEE, 2021, pp. 1–8.
9. P. Muñoz, M. Wimmer, J. Troya, and A. Vallecillo, "Using trace alignments for measuring the similarity between a physical and its digital twin," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 503–510.
10. P. Muñoz, J. Troya, and A. Vallecillo, "Using uml and ocl models to realize high-level digital twins," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2021, pp. 212–220.
11. P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, and A. Wortmann, "Model-driven development of a digital twin for injection molding," in *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings*. Springer, 2020, pp. 85–100.
12. M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, p. 5396, 2022.
13. F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *Systems Modelling and Management: First International Conference, ICSMM 2020, Bergen, Norway, June 25–26, 2020, Proceedings 1*. Springer, 2020, pp. 43–54.
14. F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, pp. 3563–3576, 2018.
15. Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, and A. Nee, "Enabling technologies and tools for digital twin," *Journal of Manufacturing Systems*, vol. 58, pp. 3–21, 2021.
16. J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 90–101.
17. M. Dalibor, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Towards a model-driven architecture for interactive digital twin cockpits," in *Conceptual Modeling: 39th International Conference, ER 2020, Vienna, Austria, November 3–6, 2020, Proceedings*. Springer, 2020, pp. 377–387.
18. T. Bolender, G. Bürvenich, M. Dalibor, B. Rumpe, and A. Wortmann, "Self-adaptive manufacturing with digital twins," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2021, pp. 156–166.
19. A. Arrieta, "Multi-fidelity digital twins: a means for better cyber-physical systems testing?" *arXiv preprint arXiv:2101.05697*, 2021.
20. M. Gogolla, F. Büttner, and M. Richters, "Use: A uml-based specification environment for validating uml and ocl," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27–34, 2007.
21. F. Büttner and M. Gogolla, "On ocl-based imperative languages," *Science of Computer Programming*, vol. 92, pp. 162–178, 2014.