Digital Twins in Manufacturing and the Use of Models

Alexander Hellwig[1,*,†], Judith Michael[1,2,†], Jérôme Pfeiffer[3,†], Bernhard Rumpe[1,†], Henrik Thillmann[1,†] and Andreas Wortmann[3,†]

[1]*Software Engineering, RWTH Aachen University, Germany*
[2]*Programming and Software Engineering, University of Regensburg, Regensburg, Germany*
[3]*Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Stuttgart, Germany*

Abstract

While models and digital twins are heavily related, how we use these models, how we understand those models, and even what these models represent for digital twins might change over time. Existing research provides model classifications from different angles, e.g., kind of usage, model content, nature of abstraction; however, a common understanding of which kinds of models could be useful and are used in digital twin engineering and during its lifetime is missing. In this work, we present an initial faceted classification to label models based on different properties relevant for digital twins in manufacturing. We show the application of the classification on different models of a digital twin in manufacturing. The classification can make finding and reusing models in and for digital twins easier, help to identify information gaps, and give inspiration about how models might change their purpose and nature over time.

Keywords

Digital Twin, Model Classification, Manufacturing, Cyber-physical Systems

1. Introduction

Digital Twins (DTs), as a software system representing an original system [1, 2], and the concept *model* are strongly related [3]: We can use models to represent the original system [4, 5, 6] and to derive and run the DT [7, 8]. What kinds of models are used at which point in time to represent which information, however, heavily rely on various factors, e.g., the different backgrounds of the engineers developing DTs [9, 10], the different application domains, the life-cycle phase when a model is used [3], and many more. This makes it challenging to reuse or adapt existing models effectively and to assess the model's relevance and validity as the digital twin evolves.

Up to now, no commonly used taxonomy or classification of models exists that can be used for the engineering of or in a running digital twin. Several works aim to classify models, e.g., Boyes and Watson [11] distinguish between different kinds of data models and physical entity models, Eramo et al. [12] distinguish between three types of purposes of models, or the Software Engineering Body of Knowledge (SWEBoK) [13] distinguishes three types based on the model content. The classification of a model, however, can change over time, e.g., a model describing the structure of a production machine during development can also be used to analyze the degradation over time during operation [14]. Hence, taxonomies, as hierarchical classifications, are not always suitable for models: not only might the model's purpose and how it is used change, but also the information and level of abstraction it considers could encompass a combination of different properties.

To cope with multidimensionality, we need a faceted classification [15, 16] of models in the life cycle of digital twins: Faceted classification is a way of classifying objects not into inflexible, hierarchical tree

ER2025: Companion Proceedings of the 44th International Conference on Conceptual Modeling: Industrial Track, ER Forum, 8th SCME, Doctoral Consortium, Tutorials, Project Exhibitions, Posters and Demos, October 20-23, 2025, Poitiers, France
*Corresponding author.
†These authors contributed equally.
hellwig@se-rwth.de (A. Hellwig); judith.michael@ur.de (J. Michael); jerome.pfeiffer@isw.uni-stuttgart.de (J. Pfeiffer); rumpe@se-rwth.de (B. Rumpe); thillmann@se-rwth.de (H. Thillmann); andreas.wortmann@isw.uni-stuttgart.de (A. Wortmann)
0009-0001-1698-4054 (A. Hellwig); 0000-0002-4999-2544 (J. Michael); 0000-0002-8953-1064 (J. Pfeiffer); 0000-0002-2147-1966 (B. Rumpe); 0009-0002-9001-0028 (H. Thillmann); 0000-0003-3534-253X (A. Wortmann)

structures, but one can link them to many different facets and their concrete manifestations called *foci*. For instance, a model can include structure and behavior information of a system under study, resulting in adding the foci structure and behavior; this can also be seen as some sort of labeling. One facet can have several foci, one foci belongs to one facet, and the foci of a facet can have a hierarchical structure, but do not have to. To an object, foci of several facets can be assigned (polyhierarchical classification system). We can assign several foci of a facet to an object (polydimensional usage of the classification). The classification of an object results from the combination of the assigned foci.

In this paper, we aim to answer the question of *what kinds of classifications of models are relevant for DTs in manufacturing*. This includes the questions of which model classifications are relevant for the engineering of DTs and which model classifications are rather important during runtime of a digital twin. Thus, we have analyzed commonly used classifications for models that are relevant for DTs and integrated them into a faceted classification grouped into facets and below their foci. We validated the faceted classification using examples from cyber-physical systems (CPSs) in manufacturing and discussed possible extensions in the future.

Structure: Sec. 2 explains related work for model classifications. Sec. 3 presents our suggestions for classification facets and their foci. Sec. 4 describes some case studies and their model classification. Sec. 5 discusses our method and results. Sec. 6 concludes by discussing the research agenda to move on with this new idea.

2. Related Work on Model Classifications

Models are classified based on various properties in the literature. Models can be classified based on the extensibility [17, 18, 19] or formality [18, 17] of the modeling language they conform to. Besides, in model-driven architecture [20], models are classified based on their level of abstraction, ranging from computation-independent to computation-specific models.

Another possible classification is based on the model type. This can be a rather high-level categorization into behavior and structural models [13, 18, 21], *e.g.,* in UML or by Lehner et al. [22] in a more detailed classification that is conceived in a systematic mapping study investigating different model-driven engineering (MDE) techniques for DTs. The study differentiates architecture models, continuous behavior models, discrete behavior models, data models, ontology models, and UI models. Tao et al. [23] differentiates models into four dimensions: geometry, physics, behavior, and rule.

One important aspect of the usefulness of models is based on their executability. Executability can be achieved by interpreting and simulating models [18, 22], by transforming a set of source models into a set of target models to enable the composition of different views [18, 22], or by generating executable code [22].

Eramo et al. [12] investigate the purpose of the model in the twin. They classify models into descriptive models representing current or past information, predictive models that serve to understand and analyze the (potential) future of the system, and prescriptive models that serve to control the future behavior of the actual system.

Although a list of domains where models play a role in the creation of DTs is necessarily incomplete, prominent ones include manufacturing, energy, aerospace, business or healthcare [23]. Such lists are typically compiled in literature surveys or mapping studies about DTs [24].

In MDE [25] models are defined based on the properties introduced by Stachowiak [4], assuming that a model has an original, that it is an abstraction of this original, and the model fulfills a purpose with respect to the original. Besides, MDE categorizes models into two categories: (1) models that describe the internals of the software system itself, *e.g.,* its software architecture, and (2) models that describe their interaction with the environment, that, *e.g.,* the production plant to be controlled or monitored. Furthermore, models can be classified along their notation, which can be either textual or graphical [18, 26] or projectional [27].

Boyes and Watson [11] investigate the functional components of DTs and splits these components into four categories with several subcategories, which will be further discussed in Sec. 3. These categories

are (1) Digital Coupling (2) Digital Representation (3) Tools (4) Functional Output.

Another work [26] does not offer a structured overview over model categories, but some dimensions, along which models can be distinguished, as well as listing specific types of models, *i.e.,* whether a models is based on a formalism, is a physical or abstract model, whether it is descriptive or analytical, or whether it is domain specific or general purpose.

Finally, in practice, models are used in different life cycle phases of the actual system and the DT [3, 28], and, hence, models can be classified along these phases, *i.e.,* requirements analysis, design, implementation, test, release, and operate (including maintenance).

3. A Faceted Classification for Digital Twin Models

We present our faceted classification for DT models whereas each classification is based on related model categories from the literature presented in Sec. 2. The origins of the foci are listed after each facet name. For each classification, we describe its foci together with an example demonstrating in which cases they are applicable to a model. *The more detailed mapping can be found at our companion website [29].*

Purpose [12]: Models may serve different purposes for the user.

- *Descriptive models*, representing current or past information about the system (and maybe its environment) to enable analyzing it, e.g. geometry models [30].
- *Predictive models*, serve to understand and analyze the (potential) future of the system. Typical uses cases include predictive maintenance or what-if analyses, e.g. with simulation models [31], or diagnosis models [2].
- *Prescriptive models*, serve to control the future behavior of the actual system. In contrast to a predictive model about, *e.g.,* predictive maintenance, the prescriptive model controls how to initiate that maintenance. This includes models for optimization [23] as well as actuation [32].

Type [22]: The type of a model describes the subject that is described by the model. On an abstract level, we differentiate between models that describe structure or behavior.

- Structure
  - *Architecture model*, applies to models that describe the communication streams between different software systems. An example are SysML part definitions or MontiArc models.
  - *Data model*, applies to models that represent the structure of data that is emitted or received by physical or software systems. Examples for data models are AutomationML or the Asset Administration Shell.
  - *Ontology model*, applies to models that describe knowledge represented via structure and rules. Examples for ontology models are the Web Ontology Language (OWL) and Ontological Modeling Language (OML).
- Behavior
  - *Discrete behavior model*, applies to models that provide information about the discrete behavior of a physical or software system. Examples are SysML state machines models or BPMN.
  - *Continuous behavior model*, applies to models that describes the contiuous behavior of a physical or software system. Examples for this type of models are Simulink and Modelica models.

Executability [17, 18, 19]: Executability of a model refers to the ability to simulate or run the model to observe its behavior over time without needing to implement it in code first.

- *Not executable*, applies if a model is not executable. An example are UML class diagrams that do not have a formally defined semantic and, hence, cannot be executed.
- *Interpreted*, applies to models that are directly processed by a tool without translation. This applies, for example, to Simulink or workflow models.
- *Transformed*, applies to models that are translated to another kind of model or to code. An example could be an UML activity diagram that is transformed into Java.

Notation [18, 26, 27]: The notation describes how the modeler interacts with the model.

- *Textual*: The model is described in text form. A lexer and parser are then used to create the abstract syntax tree (AST), making the content processable by a computer.
- *Graphical*: The AST of the model can be manipulated via a graphical representation, such as diagrams.
- *Projectional*, if the AST is indirectly edited through different projections of the model. E.g., if parts of the model are edited via a table and other parts are graphical.

Modeled subject [11, 25]: Different parts of the overarching system –composed of a Actual System (AS), a DT, the connection between them, and the connection to the physical environment –can be subjects of the model.

- *Software*, applies if the software of the system is modeled, e.g., the DT or the control part of the CPS.
- *Hardware*, applies if the model describes the hardware, such as a CAD model of the AS.
- *Interaction between AS and DT*, applies if the synchronization between hardware and software is modeled, for example, a model of the protocol between the control software and a programmable logic controller.
- *Interaction with the environment*, applies if the interaction with the physical environment is modeled, e.g., descriptions of the sensors or actuators.

Functional aspect [11]: The different functional components of a DT described in [11] can be the subject of a model. A faceted categorization of models along this axis can help find aspects of the DT that are not yet modeled.
Digital Coupling, if the model pertains to transfer of state data from the AS to the DT.

- *AS state*, if the model relates to the state of the AS
- *Communication*, if it models the connection between DT and AS
- *State data handling*, if it models the transformation of AS state data for use in the DT
- *Twinning*, if it models the push or pull of state data or the rate of data sync
- *Protocols*, if it models the protocols used for digital coupling of the two systems

Digital Representation, if the model describes a representation or data of the AS.

- *Data model*, if it models data structure used in the DT

- *Operational data storage*, it models data storage for the operation, condition, and use of the AS
- *Master & reference data*[1], if it models data to configure models of the AS
- *Physical entity models*, if it models the physical state during, or process of, the operation of the AS
- *Temporal*, if it models changing performance or data granularity over time

Tools, if a tool for decision making for the AS is modeled

- *Analysis*, if it models an analysis of the AS
- *Simulation*, if it models a simulation of the AS
- *Presentation*, if it models the presentation of analysis or simulation results

Functional output: Models describing the output of the DT to actors in the system, such as operators or other systems.

- *Digital twin output*, if it models the output of the DT to be used by an operator, the AS, or other system.
- *Configuration & control*, if it models by who or how the behavior of the DT can be changed
- *User interface*, if the interaction with an operator is modeled, e.g., models of a GUI

Level of abstraction [20]: Models may describe the DT or the AS at different levels of abstraction:

- *Computation-independent model (CIM)*, describe the business context and requirements of the system without considering how the system will be implemented. An example can be Use Case models that describes the requirements and potential interactions with a DT.
- *Platform-independent model (PIM)*, describes the system's structure and functionality without specifying the technology platform. A typical example are class diagrams, that model the designed system architecture.
- *Platform-specific model (PSM)*, enriches the PIM with platform-specific information, such that it can be implemented. This could be, for example, a Java class implementing the class diagram described in the PIM.

Domain [33, 34, 23]: The wide range of domains in which DTs are applied makes any list inherently incomplete. Notable disciplines include mechanics, electricity, civil engineering, chemistry, hydraulics, kinematics or material science. Models from those disciplines can be used in DTs for application fields such as manufacturing, energy, aerospace, business, engineering construction or healthcare [33]. Some types of models used in DTs are domain-specific, while others have more universal applicability. The models can be used to represent entities ranging from injection molding machines [34], over valves, to hearts [23].

Life cycle phase [35]: Both the DT and the AS go through different life cycle phases from the inception of the system to its eventual shutdown. Since the life cycles of the twins are not necessarily synchronized, e.g., a simulation of the AS can be used in the DT long before the physical part is deployed, we define facets for each of the twins. Each of the facets can be applied if the model is used during the corresponding life cycle phase. For the AS these phases are considered: (1) *AS Design* (2) *AS Construction* (3) *AS Operation* (4) *AS End-of-life.* The DT goes through the phases: (1) *DT Requirement analysis* (2) *DT Design* (3) *DT Construction* (4) *DT Test* (5) *DT Release* (6) *DT Operation.*

The way models are created and used during a project is influenced by the features of the language itself. In UML and SysML it is commonplace to customize the language to the problem at hand by defining profiles, while other languages do not support customization.

[1]Refers to the meta level, i.e., models that describe or configure other models of the asset.

Extensibility [17, 18, 19]: Extensibility expresses the degree to which the employed modeling language are extensible.

- *Extensible*, applies if a modeling language is extensible. For general purpose languages, *e.g.,* SysML, UML, this can be enabled by stereotypes, or for domain-specific languages (DSLs) this is possible via composition operators [19], *e.g.,* embedding.
- *Not extensible*, applies if a modeling language is not extensible. For instance, OPC UA [36] is a standardized language that has no extension mechanism.

Formality [18, 17]: The formality of a modeling language refers to the degree to which its syntax and semantics [37] are precisely defined, typically using mathematical or logical foundations. A formally defined modeling language allows unambiguous interpretation and rigorous analysis, whereas informal ones rely more on natural language and are open to interpretation.

- *Formally defined syntax*, applies if a language provides a metamodel or grammar that specifies its syntax. An example for a modeling language with formally defined syntax is UML that uses MOF (Meta Object Facility) to describe its metamodel.
- *Formally defined semantics*, applies when a language specifies its semantics formally, operationally or mathematically. An example is Z that is based on set theory and first-order predicate logic, with both its syntax and semantics rigorously defined using mathematical constructs.
- *Informal*, applies if a language is not defined formally. For instance, altough, BPMN a standardized visual syntax, its semantics—especially for complex behaviors—are not fully mathematically defined, leading to potential ambiguities in interpretation and implementation.

4. Case studies

This section applies our classification to two real-world case studies from the manufacturing domain and demonstrates its usefulness and applicability. The models of two case studies were classified with the categories presented in Sec. 3 and result of this classification can be found at our companion website [29]. In the following, we present a subset of this classification and how models of the actual system and the digital twin can switch their foci based on the life cycle.

4.1. Application of the classification

4.1.1. Fischertechnik

Using a model-driven DevOps approach, a mock production line and accompanying DT are developed [38]. It consists of a sorting line, which sorts tokens by color, a conveyor belt, which is used as an input buffer, and a multi-processing station, which applies a hardening and milling step to the workpiece. Two vacuum gripper robots are used to move the tokens between all other machines, and the output is stored in a high bay storage. The digital twin of this system keeps track of the inventory, all production steps performed on each workpiece, and the state of all machines. Additionally, an optimization service increases the throughput of the production line , and a predictive maintenance service schedules replacement of mill tools as well as maintenance of the vacuum grippers. All services are accessible to the operators through a dashboard.

Class diagram A MontiCore class diagram is used to analyze the *data structure* of the domain, *manufacturing*, during the *AS design* life cycle phase. The model is *extensible* through the application of stereotypes to model elements, but the base language has *defined semantics*, defined as the set of all conforming object structures [21]. The language notation is *textual* and the models are executable by *transformation* into Java code via a generator. During analysis, the model is *descriptive* of the

data and relationships between the *hardware*, *software*, *environment*, and other actors. It is also a *Computation-Independent Model*, since the implementation and platform are considered during initial analysis.

MontiArc MontiArc component & connector diagrams are used to model the functional *architecture* of the production line, by hierarchically decomposing the top-level components. Thus, this model is used during the *AS design* live cycle phase and is *prescriptive* for the engineered system. The language has a *textual* notation and is *extensible*, which allows for adaptation to the use case using MontiCores language composition operators. FOCUS [39] *defines the semantics* of MontiArc, which can be used to execute the models via *transformation*. Since the functions describe the full CPS, the modeled subjects are *hardware*, *software*, and *interaction with the environment*. The models belong to the domain of *manufacturing*, since a production line is modeled, and are *platform independent*, as the implementation technologies and platform are not specified.

Statecharts To model the *discrete behavior* of the machines and controllers of the production line, MontiCore Statechart models are used during the *AS design* life cycle phase. Since the same language workbench is used for MontiArc and Statecharts, they share some facets: The modeling language is also *extensible*, has a *textual* notation, and can be executed through *transformation* to Java code. Also, the domain *manufacturing* stays unchanged. The models are *prescriptive* of the behavior of the machines during operation and have *defined semantics* through automata theory. The model has multiple subjects: the states of *hardware* and *software* are described, and the stimuli of the transitions have their origin within the system as well as from the *interaction with the environment*. The models depend on the concrete machines used and thus are *platform dependent*.

4.1.2. Five-X

The second case study of our classification is based on the Five-X [40], a drilling machine that possesses five degrees of freedom and a tool changer with multiple milling tools for different materials. With the Five-X, we realize a predictive maintenance use case for our DT, where, based on its use, the hardware tool is changed automatically.

Asset Administration Shell In the context of the Five-X, asset administration shell (AAS) models are used for storing technical properties, *e.g.,* hardware interface, how the machine is constructed, and relevant documentation. During operation, the AAS stores attributes like energy consumption and the wear index of the connected tool. The language of AAS is neither extensible nor formally defined. AAS models can be classified as data models since they store static information, although the referenced models within the AAS, of course, can be of other types, too. AAS models are interpreted and can be formulated both textually and graphically, *e.g.,* with Eclipse BaSyx [41]. Concerning the modeled subject, the AAS models properties of the hardware in our case. The purpose is to prescriptively describe the operational data of the AS. In our case, it models the subject hardware. Regarding the level of abstraction, our AAS model is at the level of a platform-specific model.

Simulation models For the virtual commissioning of the Five-X, we employ the virtual commissioning tool ISG virtuos, which utilizes a 3D model with kinematics based on a CAD import and a block diagram describing the machine's behavior. The language describing the 3D model is neither extensible nor formally defined. The model's type is data, and it is executed by the simulation tool. For the DT it describes a simulation of the AS and a description of the physical entities. Since it is a visual 3D model, its notation is graphical. During operation, the model is synchronized with the AS acting as a descriptive model. The modeled subject is the hardware of the machine. Since the model is contained in the tool environment of Virtuos, it is a platform-specific model.

The language of the block diagrams is also not extensible and not formally expressed. The models are interpreted by virtuos and describe discrete behavior. It accompanies the 3D model and, hence, also

describes the simulation and the physical entity. It is also defined graphically and is a descriptive model of the AS. Like the 3D model the modeled subject is hardware, and since it is defined within Virtuos itself, it is also platform-specific.

OPC UA OPC UA describes the communication interface provided by the AS to the DT. The language itself is defined by a standard document; hence, it is not formally defined and is not extensible. OPC UA models are interpreted and describe the communication aspect of the DT. Depending on the editor, OPC UA can be edited in graphical or textual notation. OPC UA models have a descriptive purpose by modeling the interaction with the AS. The domain of this language is manufacturing. Concerning the level of abstraction of OPC UA models, they depend on the underlying system and are, thus, platform-specific models.

4.2. Changing foci in the system life cycle

Over time, the foci related to models could change. In the following, we provide some practical examples of how these changes could appear.

4.2.1. Fischertechnik

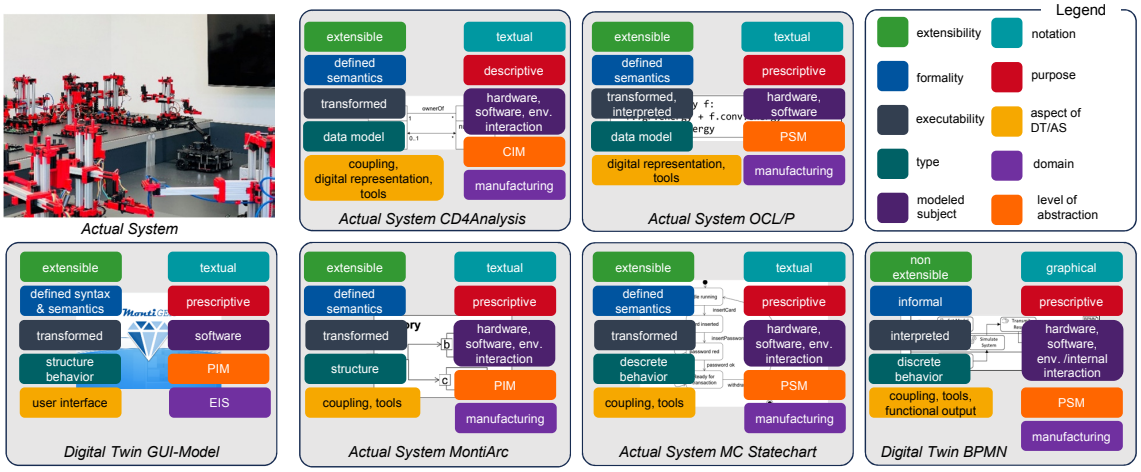

Figure 1: Models used in the DT and AS in the Fischertechnik case study.

The class diagram created during domain analysis is reused and evolves through different life cycle phases of both the DT and the AS. During the creation of the MontiArc model, parts of the analysis class diagram are reused as the data structure for the messages sent between the components of the system. Thus, the model foci changes from *descriptive* to *prescriptive*. While the class diagram is *descriptive* of the actual production line machines during the *AS design*, they can be reused in the *DT design* as the schema for a digital shadow, where they are now *prescriptive* of the digital representation of the machine.

The MontiArc model, created during AS design, describes how its function is decomposed, and as such, it is a prescriptive model in this life cycle phase. The same model can be reused as a descriptive model in the DT, by monitoring the messages passed between the component implementation and comparing them to the specification. Operators are alerted of detected deviations to handle implementation or operation errors.

4.2.2. Five-X

In the course of the Five-X's life cycle, different models are used in different phases for either modeling the functionality of the DT or the AS. For creating the DT, a GUI model describing the DT dashboard

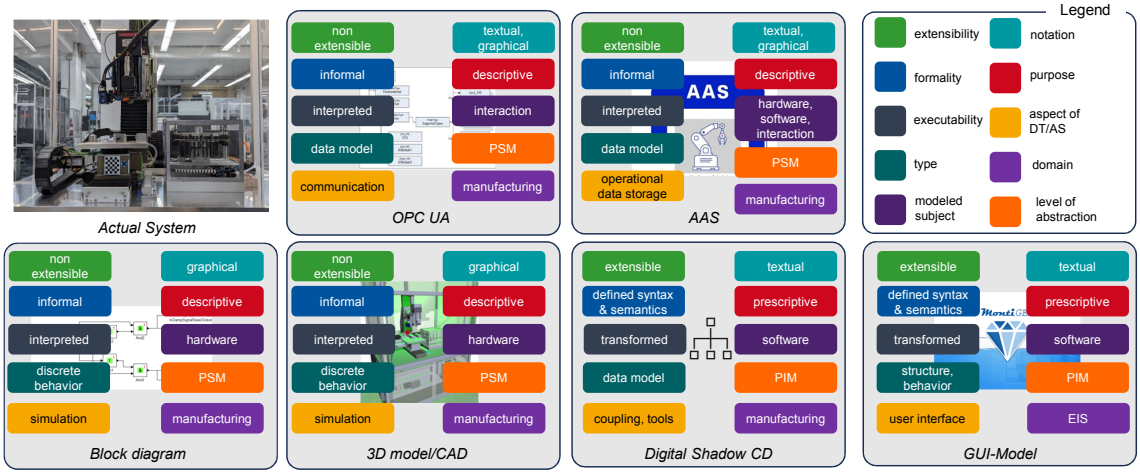

Figure 2: Models used in DT and AS in the Five-X case study.

structure and behavior is utilized, as well as a class diagram that describes the digital shadows that the DT requires for working properly. In the creation of the AS, a virtual commissioning model consisting of a 3D model and a block diagram is created. Based on this, the AS and its control are configured and fine-tuned. Furthermore, an AAS stores design-time information, *e.g.*, wiring plans and safety constraints. When moving to the operation phase, the virtual commissioning models created during the AS design phase become relevant for the DT. They can represent the current state of the AS in the DT and the DT can compute analysis, *e.g.*, a wear analysis of the connected drilling tool. For the AS the AAS created in design also becomes relevant for its operation, storing information such as time of operation and energy consumption. The OPC UA model of the Five-X is created at the operation phase of the AS, where it acts as a communication interface between DT and AS. Hence, it must be established before the operation of the DT.

5. Discussion

Insights from the case studies. When looking in depth into different foci of models, it became clear that there is more potential in reusing models or at least model information that was created in the design phase, also for other life cycle phases or for generating the DT. This results in a change of the purpose of the model usage and its context. This would result in having to change the foci or having different sets of facets and foci bound to a model at a certain point in time, related to the model usages. More comprehensive analyses are needed to draw further conclusions, e.g., which types of models are needed in which life cycle phase, which information in models overlaps, and what typical paths are in which model types are evolving.

Application methods of the classification. Having this facetted classification, we see several possible ways in which it can be usefully applied in practice:

1. This classification could be added to existing models as some sort of metadata describing them. This makes it easier to find and reuse models, as their initial usage mode, the way of using them, or their domain range is made explicit.

2. When creating a model about a CPS that shall be used for digital twin engineering, the classification could give an idea about which type of information might be missing in the models to cover certain aspects in a digital twin.

3. The typical paths of foci additions and changes could give developers inspiration about how their models might be used over time and what they could do with them within the digital twin's

lifetime. This could help them to further consider what to model at the beginning as, e.g., the purpose of a model might change over time and further information could be needed later on.

Limitations. We are not domain experts in mechanical engineering, so models from this particular domain, e.g., for the engineering process of a production machine, must be classified by domain experts. In our analysis, we may have missed an important classification. Thus, further evaluations such as expert panels or interviews are needed, to get a more holistic picture of what model classifications are relevant. Further aspects that could be worth discussing further with examples are:

- separating the model, its representation, and the usage of a model.
- defining relations between foci and facets to restrict combinations (similarly to restricting variants in feature models) or mark mandatory combinations (i.e., simulation as a purpose might require the models also to be executable). This could require using a representation as feature diagrams with OCL for defining these relations or describing the classification as an ontology.

What is additionally missing is concrete guidance on how to implement this classification system in practice using particular tooling. However, this can be suggested after further discussion with digital twin engineers from different application domains.

Generalization to other application domains and classification extension. Even though we have started to apply this facetted classification for models used in the manufacturing domain, we envision an application also in other domains where digital twins are of interest. This might mean, that additional facets and foci might have to be added to also consider concerns from these domains. We see our classification, therefore as an initial discussion point for further research and as a point where the different digital twin engineering communities can share and unify their perspectives [42].

Model classifications as a base for reuse. In addition, this work on classification can contribute to the the National Research Data Infrastructure for and with Computer Science (NFDIxCS)[2] project, which works on FAIR Data Principles for sharing and reusing computer science research data and software artifacts, i.e., models. Having such a classification within platforms managing research data encapsulating models, together with relevant data, software and context in Research Data Manageement Containers [43, 44] is one step further to foster the reuse of models. The foci in the classification can make the search for particular model types and properties easier on such platforms. In addition, in the context of digital twins, one can connect model types usable during DT runtime to particular software components in this digital twin using these models, i.e., model managers for specific modeling languages [45], simulators or analysis components. This connection would make reuse of models easier as they come along with the software components to use them.

6. Conclusion

We have introduced an initial faceted classification for models used in and for digital twins in manufacturing. We have integrated different notions in literature and added missing concepts relevant for digital twins. Our application of the classifications on two industially inspired cases demonstrates that such model foci can change over the asset lifetime.

Moving on with this research requires further discussions within the community on additional case studies to refine the classification, formalize constraints between facets, extend the coverage beyond manufacturing, and provide tooling to support the classification process, but also querying of existing models that can be reused within open or closed industrial settings.

[2] https://nfdixcs.org/

Acknowledgments

Partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2023 Internet of Production - 390621612. Website: https://www.iop.rwth-aachen.de.
Partly funded by the DFG – Model-Based DevOps – 505496753. Website: https://mbdo.github.io.
Partly funded by the DFG under NFDI 52/1 – NFDIxCS – 501930651.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, Gemini, and DeepL in order to: Grammar and spelling check, Paraphrase and reword. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] ISO/IEC, ISO/IEC 30173: Digital twin – Concepts and terminology, Technical Report, International Standardization Organization (ISO), 2023.

[2] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn, Digital twin in manufacturing: A categorical literature review and classification, IFAC-PapersOnLine 51 (2018) 1016–1022. doi:https://doi.org/10.1016/j.ifacol.2018.08.474, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

[3] J. Michael, L. Cleophas, S. Zschaler, T. Clark, B. Combemale, T. Godfrey, D. Khelladi, V. Kulkarni, D. Lehner, B. Rumpe, M. Wimmer, A. Wortmann, S. Ali, B. Barn, I. Barosan, N. Bencomo, F. Bordeleau, G. Grossmann, G. Karsai, O. Kopp, B. Mitschang, P. Muñoz Ariza, A. Pierantonio, F. Polack, M. Riebisch, H. Schlingloff, M. Stumptner, A. Vallecillo, M. van den Brand, H. Vangheluwe, Model-Driven Engineering for Digital Twins: Opportunities and Challenges, INCOSE Systems Engineering (2025). URL: https://doi.org/10.1002/sys.21815. doi:10.1002/sys.21815.

[4] H. Stachowiak, Allgemeine Modelltheorie, Springer Verlag, Wien New York, 1973.

[5] X. Hu, R. H. Assaad, A bim-enabled digital twin framework for real-time indoor environment monitoring and visualization by integrating autonomous robotics, lidar-based 3d mobile mapping, iot sensing, and indoor positioning technologies, Journal of Building Engineering 86 (2024) 108901. doi:10.1016/j.jobe.2024.108901.

[6] E. Ferko, L. Berardinelli, A. Bucaioni, M. Behnam, M. Wimmer, Towards interoperable digital twins: Integrating sysml into aas with higher-order transformations, in: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), 2024, pp. 342–349. doi:10.1109/ICSA-C63560.2024.00063.

[7] T. Brockhoff, M. Heithoff, I. Koren, J. Michael, J. Pfeiffer, B. Rumpe, M. S. Uysal, W. M. P. van der Aalst, A. Wortmann, Process Prediction with Digital Twins, in: Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C), ACM/IEEE, 2021, pp. 182–187.

[8] D. Bano, J. Michael, B. Rumpe, S. Varga, M. Weske, Process-Aware Digital Twin Cockpit Synthesis from Event Logs, Journal of Computer Languages (COLA) 70 (2022). doi:10.1016/j.cola.2022.101121.

[9] M. Amrani, D. Blouin, R. Heinrich, A. Rensink, H. Vangheluwe, A. Wortmann, Multi-paradigm modelling for cyber–physical systems: a descriptive framework, Software and Systems Modeling 20 (2021) 611–639.

[10] A. Bucchiarone, B. Combemale, A. Pierantonio, N. Bencomo, M. van den Brand, J.-M. Bruel, A. Cicchetti, J. Di Rocco, L. Lambers, J. Michael, B. Rumpe, M. Sjödin, G. Taentzer, M. Tichy, H. Vangheluwe, M. Wimmer, S. Zschaler, Modeling: The Heart and Soul of Engineering Smart Ecosystems, in: Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C): SAM Conference, ACM/IEEE, 2025.

[11] H. Boyes, T. Watson, Digital twins: An analysis framework and open issues, Computers in Industry 143 (2022) 103763. doi:https://doi.org/10.1016/j.compind.2022.103763.
[12] R. Eramo, F. Bordeleau, B. Combemale, M. van Den Brand, M. Wimmer, A. Wortmann, Conceptualizing digital twins, IEEE Software (2021).
[13] H. Washizaki, Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), IEEE Computer Society, 2024. URL: www.swebok.org, version 4.0.
[14] Y. Kang, H. Yan, F. Ju, Performance evaluation of production systems using real-time machine degradation signals, IEEE Transactions on Automation Science and Engineering 17 (2020) 273–283. doi:10.1109/TASE.2019.2920874.
[15] K. La Barre, Facet analysis, Annual Review of Information Science and Technology 44 (2010) 243–284. doi:10.1002/aris.2010.1440440113.
[16] K. Crowston, B. Kwasnik, A framework for creating a facetted classification for genres: addressing issues of multidimensionality, in: 37th Annual Hawaii Int. Conf. on System Sciences, 2004, pp. 9 pp.–. doi:10.1109/HICSS.2004.1265268.
[17] J. Seabra, L. Fernandes da Silva, Classifying software modeling languages: The clupir model approach, IEEE Access 13 (2025) 80759–80774. doi:10.1109/ACCESS.2025.3567005.
[18] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. C. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. Stikkolorum, J. Whittle, The relevance of model-driven engineering thirty years from now, in: J. Dingel, W. Schulte, I. Ramos, S. Abrahão, E. Insfran (Eds.), Model-Driven Engineering Languages and Systems, Springer International Publishing, Cham, 2014, pp. 183–200.
[19] J. Pfeiffer, B. Rumpe, D. Schmalzing, A. Wortmann, Composition operators for modeling languages: A literature review, Journal of Computer Languages 76 (2023) 101226.
[20] A. W. Brown, Model driven architecture: Principles and practice, Software Systems Modeling 3 (2004) 314–327.
[21] B. Rumpe, Modeling with UML: Language, Concepts, Methods, Springer International, 2016. URL: https://mbse.se-rwth.de/.
[22] D. Lehner, J. Zhang, J. Pfeiffer, S. Sint, A.-K. Splettstößer, M. Wimmer, A. Wortmann, Model-driven engineering for digital twins: a systematic mapping study, Software and Systems Modeling (2025).
[23] F. Tao, B. Xiao, Q. Qi, J. Cheng, P. Ji, Digital twin modeling, Journal of Manufacturing Systems 64 (2022) 372–389. doi:https://doi.org/10.1016/j.jmsy.2022.06.015.
[24] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, A. Wortmann, A cross-domain systematic mapping study on software engineering for Digital Twins, Journal of Systems and Software (JSS) 193 (2022).
[25] B. Combemale, R. France, J.-M. Jézéquel, B. Rumpe, J. Steel, D. Vojtisek, Engineering Modeling Languages: Turning Domain Knowledge into Tools, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, 2016.
[26] S. Friedenthal, Systems Engineering Body of Knowledge (SEBoK): Types of Models, 2025. https://sebokwiki.org/wiki/Types_of_Models.
[27] M. Voelter, J. Siegmund, T. Berger, B. Kolb, Towards user-friendly projectional editors, in: Int. Conf. on Software Language Engineering, Springer, 2014, pp. 41–61.
[28] R. Paredis, H. Vangheluwe, Modelling and simulation-based evaluation of twinning architectures and their deployment, in: 14th Int. Conf. on Simulation and Modeling Methodologies, Technologies and Applications SIMULTECH-Volume 1, 2024, pp. 170–182.
[29] Companion website, https://github.com/iswunistuttgart/edtconf-dts-and-the-use-of-models/, 2025. Accessed: 2025-07-09.
[30] F. Tao, M. Zhang, Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing, IEEE Access 5 (2017) 20418–20427. doi:10.1109/access.2017.2756069.
[31] S. Boschert, R. Rosen, Digital Twin—The Simulation Aspect, Springer International Publishing, Cham, 2016, pp. 59–74. doi:10.1007/978-3-319-32156-1_5.
[32] A. Fuller, Z. Fan, C. Day, C. Barlow, Digital twin: Enabling technologies, challenges and open research, IEEE Access 8 (2020) 108952–108971. doi:10.1109/ACCESS.2020.2998358.

[33] European Parliament, Regulation (EC) No 1893/2006 of the European Parliament and of the Council of 20 December 2006 establishing the statistical classification of economic activities NACE Revision 2 and amending Council Regulation (EEC) No 3037/90 as well as certain EC Regulations on specific statistical domains, 2006. URL: https://eur-lex.europa.eu/eli/reg/2006/1893/2019-07-26.

[34] M. Heithoff, N. Jansen, J. Michael, F. Rademacher, B. Rumpe, Model-Based Engineering of Multi-Purpose Digital Twins in Manufacturing, Springer Nature Switzerland, 2024, pp. 89–126. doi:10.1007/978-3-031-67778-6_5.

[35] 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, Systems Engineering (2025).

[36] S. Leitner, W. Mahnke, OPC UA - service-oriented architecture for industrial applications, Softwaretechnik-Trends 26 (2006).

[37] D. Harel, B. Rumpe, Meaningful Modeling: What's the Semantics of "Semantics"?, IEEE Computer Journal 37 (2004) 64–72.

[38] B. Combemale, N. Jansen, J.-M. Jézéquel, J. Michael, Q. Perez, F. Rademacher, B. Rumpe, D. Vojtisek, A. Wortmann, J. Zhang, Model-Based DevOps: Foundations and Challenges, in: Model-Driven Engineering for Digital Twins (MoDDiT) WS at ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion, IEEE, 2023. doi:10.1109/MODELS-C59198.2023.00076.

[39] H. Kausch, M. Pfeiffer, D. Raco, A. Rath, B. Rumpe, A. Schweiger, A Theory for Event-Driven Specifications Using Focus and MontiArc on the Example of a Data Link Uplink Feed System, in: I. Groher, T. Vogel (Eds.), Software Engineering 2023 Workshops, Gesellschaft für Informatik e.V., 2023, pp. 169–188.

[40] Stuttgarter Maschinenfabrik, https://www.isw.uni-stuttgart.de/forschung/projekte/Stuttgarter-Maschinenfabrik/, 2025. Accessed: 2025-07-09.

[41] S. Kannoth, J. Hermann, M. Damm, P. Rübel, D. Rusin, M. Jacobi, B. Mittelsdorf, T. Kuhn, P. O. Antonino, Enabling smes to industry 4.0 using the basyx middleware: A case study, in: 15th Europ. Conf. on Software Architecture (ECSA'21), Springer, 2021, pp. 277–294.

[42] J. Michael, D. Bork, M. Wimmer, H. C. Mayr, Quo Vadis Modeling? Findings of a Community Survey, an Ad-hoc Bibliometric Analysis, and Expert Interviews on Data, Process, and Software Modeling, Journal Software and Systems Modeling (SoSyM) 23 (2024) 7–28. doi:10.1007/s10270-023-01128-y.

[43] F. Al Laban, J. Bernoth, M. Goedicke, U. Lucke, M. Striewe, P. Wieder, R. Yahyapour, Establishing the research data management container in nfdixcs, in: Proceedings of the Conference on Research Data Infrastructure, volume 1, 2023. doi:10.52825/cordi.v1i.395.

[44] J. Bernoth, S. I. Ayon, F. A. Laban, S. Bavendiek, H. Federrath, M. Striewe, M. Goedicke, Workflow for creating and sealing a research data management container (rdmc), in: Software Engineering 2025 – Companion Proceedings, Gesellschaft für Informatik, Bonn, 2025. doi:10.18420/se2025-ws-26.

[45] J. Pfeiffer, J. Zhang, B. Combemale, J. Michael, B. Rumpe, M. Wimmer, A. Wortmann, Towards a Unifying Reference Model for Digital Twins of Cyber-Physical Systems, in: Int. Conf. on Emerging Technologies and Factory Automation (ETFA'25), IEEE, 2025.