Explaining Cyber-Physical System Behavior with Digital Twins

Judith Michael
RWTH Aachen University
Aachen, Germany
michael@se-rwth.de

Maike Schwammberger
Karlsruhe Institute of Technology
Karlsruhe, Germany
schwammberger@kit.edu

Andreas Wortmann
University of Stuttgart
Stuttgart, Germany
wortmann@isw.uni-stuttgart.de

***Abstract*—Digital twins are increasingly being used for many purposes in various domains, including manufacturing, healthcare, transportation, and urban planning. To enable all of this, digital twins must be complex software systems that digitally represent and manipulate physical systems. Hence, they are equipped with extensive data, information, and models to reason about the represented system and are used by domain experts from various disciplines. For their proper use, it is vital to comprehend the wealth of data, information, and models intrinsic to them. Therefore, we present a model-driven software architecture of digital twins that combines this wealth with techniques for the self-explainability of software systems to support domain experts in configuring, deploying, operating, and maintaining them.**

***Index Terms*—Cyber-Physical System, Digital Twin, Explainability, Model-Driven Engineering**

Motivation

Digital twins [1] are increasingly being used for many purposes in various domains, including manufacturing, healthcare, transportation, and urban planning [2]. In automotive, digital twins of cars are used for improving data analytics [3]. In manufacturing, digital twins simulate and optimize production processes, *e.g.,* to automatically improve the results of industrial batch processes, such as injection molding [4]. Healthcare uses digital twins to create personalized patient models and design effective therapies. Transportation uses digital twins to monitor vehicle performance and predict maintenance needs.

To enable all of this, digital twins must be complex software systems that digitally represent and manipulate Cyber-Physical Systems (CPS) and provide further services in addition to the CPS. Digital twins are used, *e.g.,* to better understand the represented system (physical twin, PT) and to optimize its behavior. Therefore, digital twins use both digital data and models about the PT. The data can be collected by communicating with the PT, by observing it, and by using data sources that have information about the PT. The models used by digital twins can be models from the engineering time of the PT: for instance AutomationML models, SysML models, Simulink models, *etc.*, as well as dedicated models for the operation of digital twins, such as AI models. As a consequence, digital twins are equipped with extensive data, information, and models to reason about the represented system. This information and functionalities are beyond functionalities provided by the cyber-part of CPS.

Digital twins are used by domain experts from various disciplines, who employ them with different purposes, perspectives, and requirements. Hence, it is vital for the proper use of digital twins to comprehend the wealth of data, information, and models intrinsic to them.

In this article, we examine the integration of self-explainability techniques with digital twins. For this, we enrich digital twins by an explanation structure that is derived from different kinds of models of a CPS. We discuss this approach with examples from the automotive domain and a combination of system description, process and reasoning models. The explanation model can be used within different digital twin services to provide explanations about the twins decisions. To reach this, we combine already existing model-driven engineering methods for digital twins with existing techniques for self-explainability of software systems to support domain experts in configuring, deploying, operating, and maintaining digital twins.

The benefits of a self-explainable digital twin are manifold: Increased trust and acceptability of autonomous and self-learning systems, a better understandability and transparency of complex system decisions, as well as an increase in safety and practicability of human-machine cooperation, *e.g.,* in take-over situations in semi-autonomous driving [5].

Model-Driven Engineering of Digital Twins

For the creation of digital twins, we employ model-driven methods: different kinds of models, e.g., the domain model, a component-connector architecture model, and models describing graphical user interfaces, are the input for code generators that create a digital twin (see Fig. 1 for the process and generation results) [4]. A domain model describes the main domain concepts and their relationships and helps to generate data structures and communication functionalities. An architecture model describes the main components of the digital twin, connectors between components, and their behavior using embedded state charts. We can use GUI models in the generation process to create user interfaces for humans interacting with a digital twin. OCL models allow us to define

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 505496753.
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.

restrictions on the data structure and to generate validators for user input. Such generation approaches work very well for repetitive parts of the code. In contrast, very specialized parts of the code, e.g., for specific services, still have to be added as hand-written parts but work together with the generated parts.

The main system architecture of a digital twin includes different services and a cockpit for visualization and user interaction [4]. For self-adaptive digital twins, services such as a data processor, evaluator, reasoner, and executor are important. The data processor prepares relevant parts of the data of the original system and transforms them into digital shadows, *i.e.,* contextualized data traces for a specific purpose. The evaluator takes this information and evaluates the current situation. If a need for self-adaption is detected, the reasoner identifies what should be changed to reach the desired situation and creates a plan. The executor takes the plan and transforms it into commands for the cyber-physical system or presents the plan to end users or operators. These then can decide if the plan should be executed or interact with the cyber-physical system themselves.

Digital twins mainly are operated not by software engineers but by domain experts familiar with the twinned system and its purpose. This is especially important in domains of long-living systems, such as automotive, avionics, or manufacturing; these systems degrade over time and evolve through maintenance. Hence, the longer the system lives, the more its behavior diverges from the idealized assumptions made during its design. For instance, the older a car gets, the more fuel it consumes for the same distance. Hence, potentially foiling tightly calculated trips.

Consequently, system behavior needs to be updated, tweaked, and refined *by the domain experts* knowing the system best. One particular means to integrate specific domain expert knowledge is case-based reasoning [6]. The cases represent domain knowledge about a specific twinned cyber-physical system. Therefore, they comprise three parts in the form

```
IF pattern THEN actions EXPECT result,
```

where

1) the *pattern* is a Boolean expression over the digital shadow data that describes a problematic situation, *e.g.,* low fuel relative to planned trip distance;
2) the *actions* are parametrizable commands being send to the system, *e.g.,* restrict maximum speed and optimize driving style for fuel efficiency;
3) the *expectation* is a Boolean expression over the expected system behavior after applying the actions, *e.g.,* prediction of extended range,

Given a problematic situation, the reasoner first tries to identify a predefined case that matches this situation to apply it. If no such case is found, the reasoner identifies the most similar case, adjusts that, and applies it. If an adjusted case solves the problematic situation, it is added to the knowledge base of cases, *i.e.,* the system has learned a new case. Hence, our model-driven digital twin architecture uses case-based reasoning AI planning to react to problematic situations [7] and adjust the system accordingly.

When designing the physical twin, we can describe its wanted behavior and interaction with human operators with process models, *e.g.,* using the Business Process Model and Notation (BPMN). These business process models can be used by digital twins to support solving problems [8]. We can use observed data, *e.g.,* from sensors, actuators, and related systems, as input for process discovery algorithms and identify how processes look like in reality. These identified processes can be compared to planned processes in the evaluator using process conformance algorithms. If lacking conformance, the digital twin can use case-based reasoning to identify needed changes in the process and either automatically execute them or suggest them to human operators.

Both, cases and business process models can be easily created by the domain experts operating the digital twins, however, they still fail to explain the behavior of the underlying system during operations in detail.

Explaining Software Decisions

Today, more and more tasks are performed by autonomous systems: from robot vacuum cleaners, via driving assistance systems to smart homes and smart factories. With this increased level of autonomy also comes more complex software. This is as an autonomous system's decision is the result of a complex decision making process including rules, desires, goals and environmental influences.

Now, what can we do, if our autonomous systems get so complex that their end-users and, potentially, even their engineers and experts fail to understand the system's decisions? We can make these systems *self-explainable*. The term self-explainability comprises any functionality that can explain reasons and causes for actions that the system decided (not) to take.

> **Example.** *Each day, the autonomous car takes the same route via a highway, because it is the shortest and fastest route for its passenger. Today, it took the longer route via the countryside. The reason for this was a heavy accident on the highway, which required rerouting.*

MAB-EX Framework

To enable the design of self-explainable systems, we developed the MAB-EX framework (Monitor, Analyze, Build, Explain) in previous work [9]. The key idea of MAB-EX is to adopt the core principles of the MAPE Loop [10] for self-adaptive systems to build self-explaining systems.

With MAB-EX, the behavior of a system is **M**onitored and **A**nalyzed and it is decided whether a recipient requires an explanation. This decision can depend on different factors: on the situation and system context, on the user's experience with similar situations, or on the characteristics of the situation (*e.g.,* rarity of a situation). In the **B**uild phase, the observed behavior is used to identify the current system state in a pre-built *explanation model* and to identify the events that

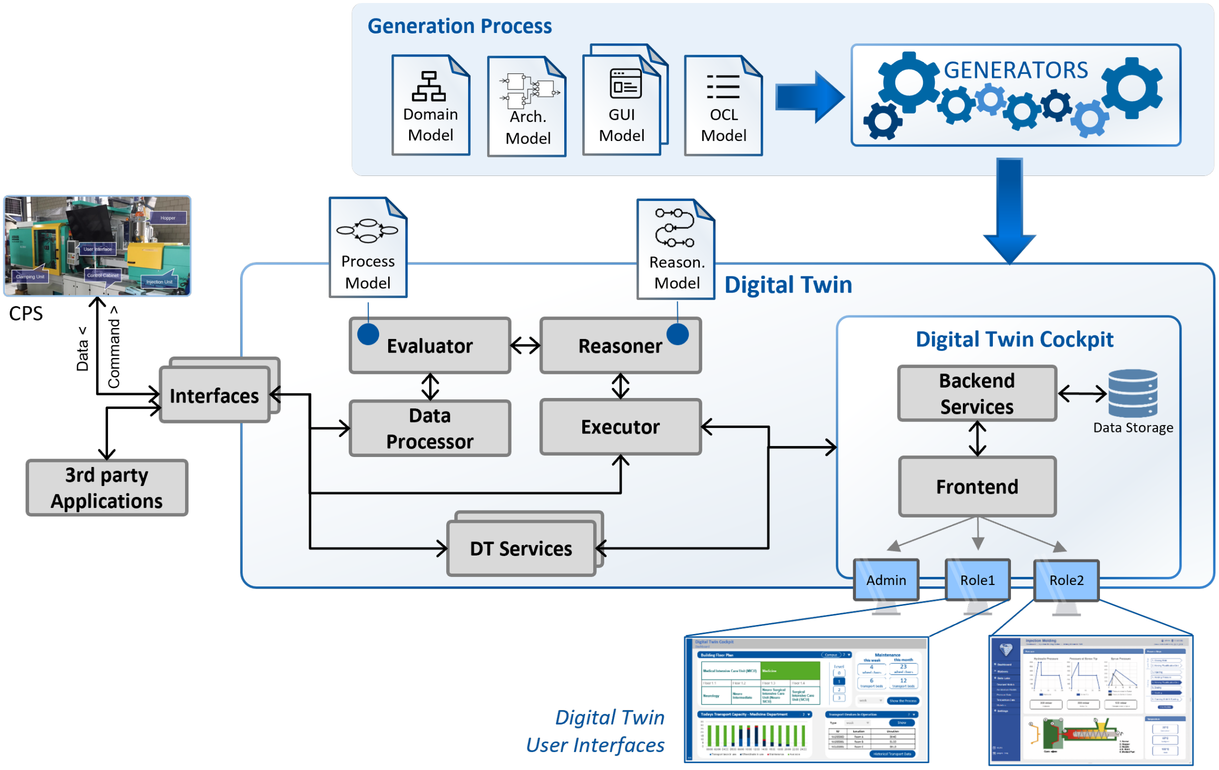


Fig. 1. Generation process and main architectural components of a digital twin

led to this state. Possible implementations for an explanation model are decision trees [11]. This is as their structure already includes decision incentives for mapping possible actions to different reasons, which we also need for explanation models. For more details on the structure of explanation models, we refer to [9]. The identified events form a trace, which we refer to as an *explanation path*. This path describes an internal representation of an explanation. It is further processed into a suitable presentation format that is given to the *explainee* in the **EX**-phase of the loop. By using the MAB-EX framework, it is possible to add an explanation module to an existing system. With that, it is not necessary to include explanation capabilities directly into the system development process and also older systems can be made self-explainable.

Deriving Explanation Models

Providing a suitable, correct and preferably minimal explanation model is crucial to provide explanations fast during run-time. This explanation model should also be tailored towards specific *explainee types*. This is as, *e.g.,* end-users need differently detailed explanations than engineers, lawyers, or interacting autonomous systems.

In previous work [12], we discuss a procedure for deriving an explanation model from an existing *system model*. By system model, we mean a formal description of a computer system: *e.g.,* an automaton or a diagram-style model with locations and transitions, or some program code.

Informally, the idea for extracting an explanation model from a system model is to connect actions of the system with their yet abstract reasons.

Example. *For instance, an action could be that an autonomous car brakes and an abstract reason for this action to be found in the system model could be that a data variable h exceeded a value* 70.

This abstract reason is still not enough that end users would understand it. This requires to add further explanatory information to concepts taken from the system model. Thus, after extracting the explanation model, we further clean-up and tailor the model towards the different explainee types. Such explainee types can, *e.g.,* be end-users, engineers, lawyers, or other technical systems.

Example. *Depending on the explainee type that receives the explanation, the information $h > 70$ needs to be further refined, e.g., to "the autonomous car stopped moving, because it's battery was overheating".*

The explanation model extraction and refinement procedure from [12] comprises three phases:

1) Extraction of the explanation model directly from the system model, as well as preparing the model for the following steps;
2) Tailoring of the model towards different explainees; and
3) Run-time updates of the explanation model after it was

DIGITAL TWINS IN A NUTSHELL

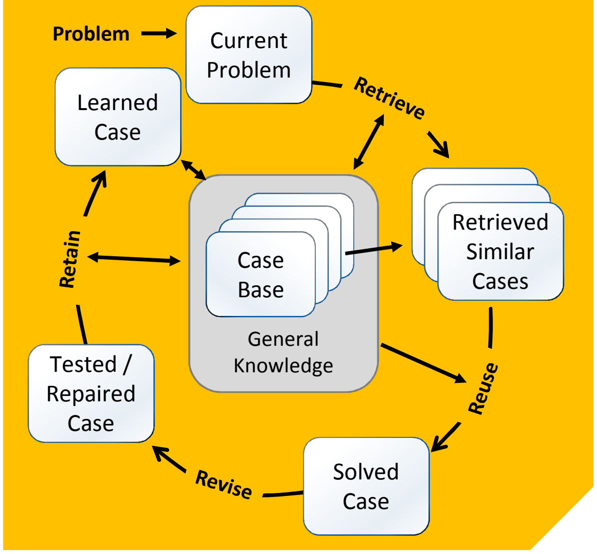
There are many different understandings of what a digital twin is. According to the most popular definition [1], **digital twins are software systems** that receive data from the physical twin and send commands to that system. To this end, **digital twins use models and data from and about that system.**

They are already **used in many different disciplines**, including agriculture, automotive, avionics, construction, healthcare, logistics, mining, production, and more to optimize, monitor, validate, and predict the behavior of the physical twin [2].

Most often, digital twins represent the physical system as operated, i.e., during its operations and based on observed data, they **produce plans to optimize the physical twin's behavior** during operations [4]. To ensure having the right combination of information at hand for such optimizations, digital twins employ **digital shadows** [5], which are instances of data structures tailored for digital twin decision making.

For such planning, the **digital twin needs domain knowledge**, which, in our model-driven digital twin architecture is encoded in **process models** and **case-based reasoning** models that can be defined by domain experts.

CASE-BASED REASONING CYCLE



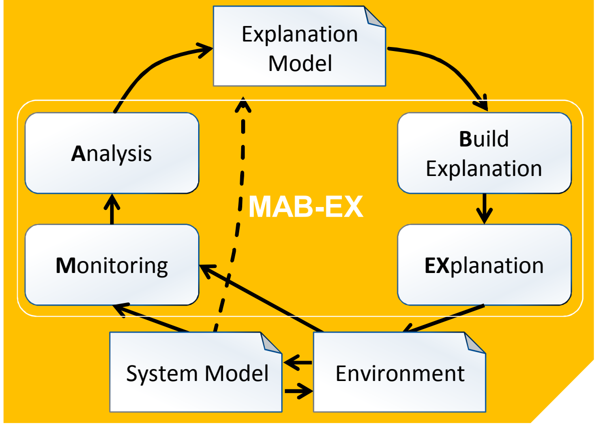
SELF-EXPLAINABILITY IN A NUTSHELL

The main characteristic of a self-explainable system is its capability to **explain its decisions and specific observables to an explainee** (i.e. recipient of an explanation), without the need of an external explanation entity. Observables that need explaining can be, e.g., unexpected system decisions or unexpected environmental information. Especially in systems **where humans and machines work closely together, explainability is vital**: let it be factory robots that assist a worker, a human driver who needs to take-over control in a semi-automated vehicle in a critical situation, or an end-user that wishes to understand why their smart home system unexpectedly closes all windows. Hence, self-explainability is **highly interdisciplinary**: computer science, psychology, philosophy, linguistics, law, social sciences and many other disciplines are working together to engineering safe, secure and trustworthy explanations.

MAB-EX FOR SELF-EXPLAINABLE SYSTEMS

The modular MAB-EX Framework [9] enables making existing systems self-explainable. It adopts features of the MAPE-K loop [4] and comprises the following cycle:

- **M**onitoring: of system and environmental data
- **A**nalysis: of the monitored data, to investigate the need for an explanation
- **B**uild: an internal version of an explanation from a pre-generated explanation model
- **EX**plain: transform the internal explanation into an explanation format understandable w.r.t. the explainee type.



deployed.

In the first phase, unnecessary data (*i.e.,* that is not needed for the explanation context), is removed. *The explanation context could be that the robot's movement should be explained, not it's communication habits.* In the second phase, unnecessary details are removed, with respect to different explainee-types. Equally, the model is enriched with additional information.

> **Example.** *For an engineer, the information that an internal data variable had an overflow is of interest, while for an end-user, such an internal variable might not be of interest, as it is not visible from the outside.*

Our explanation model is complete, w.r.t. the existing system model and known data. Of course, in real-world applications, there will always exist situations requiring explanations, which are new and impossible to consider during the building phase of the explanation model (e.g. a completely new type of an intersection that has been engineered to solve a new type of traffic problems in the future). For this, we refer to the third phase of the explanation model creation procedure, where run-time updates of the explanation model are discussed. Such a run-time update of the explanation model could, e.g., be triggered by an update of the system itself.

Explainable Digital Twins

To make digital twins explainable, we use the MAB-EX framework. For the **M**onitoring and **A**nalysis phases, the approach is straight-forward: The system can be monitored and analyzed through its digital twin, and the digital twin itself can be monitored and analyzed by an environmental entity. With this, we get both: an explainable system, where an outside entity explains the system, as well as a self-explainable system, through the digital twin explaining the system.

Also, the **EX**plain phase does not change, as it only comprises the translation of an internal explanation path as it was extracted from the explanation model. Thus, this phase is system independent: for the system type "digital twin", this is not different than for other systems. However, we perceive a challenge in the **B**uild phase, where we need to extract an explanation model from a system model: what is our system model in the case of digital twins?

For digital twins, we suggest that the explanation model must be built not from one single system model, but from (1) formal system description models, *e.g.,* an automaton or a diagram, (2) process models, *e.g.,* BPMN or UML Activity Diagrams, and (3) reasoning models, *e.g.,* case-based reasoning or planning rules. This entails translating these models into explanation models, merging the resulting models, removing redundancies and adding information, as we describe in the following.

We illustrate the combination of explanation models for digital twins in Fig. 2: First, we use the explanation model derivation procedure from the previous section independently on each of the three digital twin model types to obtain the corresponding explanation models. These resulting explanation models might contain equivalent sub-trees that describe behavior leading to the same explanation trace.

> **Example.** *Both the system description and a process model describe that, when the car's battery overheats, it must stop immediately.*

To avoid redundancy in the explanation models, we need to identify all those equivalent sub-trees throughout the three explanation models. For doing so, bisimulation relations within sub-trees of the explanation models must be detected and the equivalent sub-trees must be removed through graph transformation techniques. We then combine the cleaned explanation models. This is done through tree-merging methods which are, e.g., described for decision trees in [13].

Again consider our picture in Fig. 2. There, we can see that Y is a condition (*i.e.,* a pattern over data obserable by the digital twin) for X in both explanation models for a process model and for a reasoning model. This means that in the merged explanation model, the two sub-trees containing $Y \rightarrow X$ are merged into one sub-tree. Ultimately, this merging results in a single explanation model containing relations between nodes of different input models or merged nodes.

All of this preparation happens before the digital twin starts operating. Once running, its `Reasoner` uses the single explanation model to explain why it chooses certain actions to be performed. To this end, it receives digital shadows from the `Evaluator` and derives the current node in the explanation model. By selecting an action based on the current situation represented in the digital shadow, it navigates from that node to a child node containing this action. Through this, it tracks an explanation path and how it lead to the current decision. The explanation path relevant to the current explanation is provided to the `Executor`, which persist these explanations in the digital twin's `DataStorage` and provides these to the digital twin cockpit, where it is presented to an explainee.

Challenges

Model-driven digital twins and the twinned systems are usually created based on many different kinds of models for their cyber-physical parts, such as system structure models, kinematics models, CAD models, software architecture models, data models, deployment models, commissioning models, traceability models, and more. Our approach towards making digital twins self-explainable relies on discrete models, through which the digital twin or the cyber-physical part take dedicated steps at run-time. Deriving explanations from other kinds of models, such as continuous mathematical models, logical constraint models, or AI models is subject to ongoing research.

By using the approach from [12], we suggest to tailoring explanations towards different explainees. To automate this, a definition of explainee classes is necessary. These can, *e.g.,* be obtained from Personas [14], as they are known from software design processes. Such Personas can be used to define different user groups of digital twins and connect them with different configurations for the explanation model. For

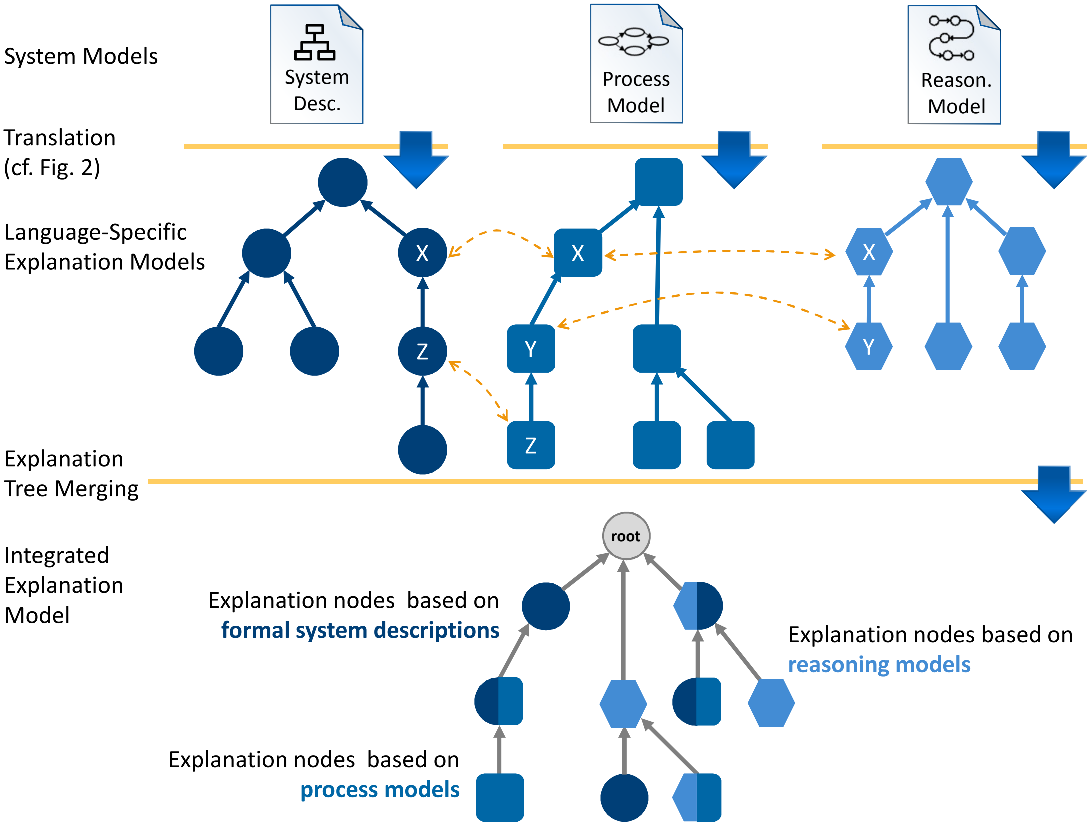

System Models
System Desc.
Process Model
Reason. Model
Translation (cf. Fig. 2)
Language-Specific Explanation Models
X
X
X
Y
Y
Z
Z
Explanation Tree Merging
Integrated Explanation Model
root
Explanation nodes based on formal system descriptions
Explanation nodes based on reasoning models
Explanation nodes based on process models

Fig. 2. Combined explanation model derivation procedure for a collection of system models.

instance, considering different explainee types entails that the internal explanation paths from the explanation model must be translated with different degrees of detail. More experienced users might request more details in explanations whereas less experienced users might only need high level explanations. Another issue to be considered is *when* an explanation should be provided; in a time-critical emergency situation, only a brief explanation might be useful.

Example. *The autonomous vehicle enters an unknown system state and explains to the human driver that they have to take-over the situation immediately.*

On the other hand, for an explanation after an event, there might be time enough to even start an explanation dialogue, where the explainee might react to an explanation to, *e.g.,* demand more details [15].

Providing explanations to human users requires us to translate the explanation models into natural language. To do so, information from various information sources is helpful. Digital twins have the advantage that they already include a set of models, data, and meta-data describing different aspects of the physical twin. Especially the models and meta-data provide a good information source to enrich explanations with natural language concepts.

The explanation models are also interesting in combination with the assistive functionalities of digital twins: Explanation models can be used in addition to step-by-step support for human users based on process model information and, thus, increase the acceptance of and trust in support services.

Moreover, creating explanations at run-time of the digital twin introduces additional computational complexity, which might be challenging if running the twin on the edge or less computationally powerful devices. This challenge becomes amplified if the system models change often at run-time of the twin, as this demands adapting the explanation model often as well. For this, it is beneficial to have the pre-built explanation model, as its' key idea is that only parts of it will have to be adaptated during run-time.

References

[1] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *Ifac-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018.

[2] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, and A. Wortmann, "A cross-domain systematic mapping study on software engineering for digital twins," *Journal of Systems and Software*, p. 111361, 2022.

[3] D. Piromalis and A. Kantaros, "Digital twins in the automotive industry: The road toward physical-digital convergence," *Applied System Innovation*, vol. 5, no. 4, p. 65, 2022.

[4] M. Dalibor, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits," in *Conceptual Modeling*, pp. 377–387, Springer, 2020.

[5] L. Chazette, V. Klös, F. Herzog, and K. Schneider, "Requirements on explanations: A quality framework for explainability," in *2022 IEEE 30th Int. Requirements Engineering Conf. (RE)*, pp. 140–152, 2022.

[6] J. Kolodner, *Case-based reasoning*. Morgan Kaufmann, 2014.

[7] F. Becker, P. Bibow, M. Dalibor, A. Gannouni, V. Hahn, C. Hopmann, M. Jarke, I. Koren, M. Kröger, J. Lipp, J. Maibaum, J. Michael, B. Rumpe, P. Sapel, N. Schäfer, G. J. Schmitz, G. Schuh, and A. Wortmann, "A Conceptual Model for Digital Shadows in Industry and its Application," in *Conceptual Modeling, ER 2021*, pp. 271–281, Springer, 2021.

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

[9] M. Blumreiter, J. Greenyer, F. J. C. Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, and A. Wortmann, "Towards self-explainable cyber-physical systems," in *22nd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion*, pp. 543–548, 2019.

[10] D. Sinreich, "An architectural blueprint for autonomic computing," *IBM Autonomic Computing – White Paper*, 2006.

[11] S. B. Kotsiantis, "Decision trees: a recent overview," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 261–283, 2013.

[12] M. Schwammberger and V. Klös, "From specification models to explanation models: An extraction and refinement process for timed automata," in *4th Int. Workshop on Formal Methods for Autonomous Systems (FMAS) and 4th Int. Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE), FMAS/ASYDE@SEFM 2022, and 4th Int. Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE)* (M. Luckcuck and M. Farrell, eds.), vol. 371 of *EPTCS*, pp. 20–37, 2022.

[13] C. Fan and P. Li, "Classification acceleration via merging decision trees," in *ACM-IMS on Foundations of Data Science Conference*, FODS '20, (New York, NY, USA), p. 13–22, ACM, 2020.

[14] S. Faily and J. Lyle, "Guidelines for integrating personas into software engineering tools," in *5th ACM SIGCHI Symp. on Engineering Interactive Computing Systems*, EICS '13, (New York, NY, USA), p. 69–74, ACM, 2013.

[15] L. A. Dennis and N. Oren, "Explaining BDI agent behaviour through dialogue," *Auton. Agents Multi Agent Syst.*, vol. 36, no. 1, p. 29, 2022.