A Family of Digital T Workflows and Architectures: Exploring Two Cases

Randy Paredis[1] [0000−0003−0069−1975],
Cláudio Gomes[2] [0000−0003−2692−9742], and
Hans Vangheluwe[1,3] [0000−0003−2079−6643]

[1] University of Antwerp, Department of Computer Science, Middelheimlaan 1, Antwerp, Belgium
[2] Aarhus University, DIGIT, Department of Electrical and Computer Engineering, Åbogade 34, Aarhus N, Denmark
[3] Flanders Make@UAntwerp, Belgium

Abstract. Digital Models/Shadows/Twins/... have been given numerous definitions and descriptions in the literature. There is no consensus on terminology, nor a comprehensive description of workflows nor architectures. In this paper, we use the catch-all "Digital T" (pronounced "Digital Twinning") to refer to all concepts, techniques, architectures, ... related to the "twinning" paradigm. In this paradigm, virtual instances, known as twins, of a System under Study (SuS) are continually updated with the SuS's health, performance, and maintenance status, over its entire life-cycle. Digital T can be used for monitoring, analysis, optimization, and adaptation of complex engineered systems, in particular after these systems have been deployed. Digital T makes full use of both historical knowledge and of streaming data from sensors. Following Multi-Paradigm Modelling (MPM) principles, this paper proposes to explicitly model construction/use workflows as well as architectures and deployment of Digital T. Applying product family modelling allows for the de-/re-construction of the different Digital T variants in a principled, reproducible and partially automatable manner. Two small illustrative cases are discussed: a Line-Following Robot and an Incubator. These are representative for respectively an Automated Guided Vehicle and an Industrial Convection Oven, both important in an industrial context.

Keywords: Digital Model · Digital Shadow · Digital Twin · Architecture · Workflow · Variability Modelling

1 INTRODUCTION

Digital Twins (DTs) are increasingly used in Industry 4.0 and industrial processes for purposes such as condition monitoring, analysis, and optimization. While their definition has changed throughout the years, the concept of "twinning" has stayed the same: there exists a digital counterpart of a real-world (realized) system that provides information about this system.

Academic and industrial interest in DTs has grown steadily, as they allow the acceleration through digitization that is at the heart of Industry 4.0. Digital Twins are made

possible by technologies such as the Internet of Things (IoT), Augmented Reality (AR), and Product Lifecycle Management (PLM).

Despite the many surveys on the topic [26,17,12,4,23,30,1,13,3,5,15,28], there is no general consensus on what characterizes a Digital Twin, let alone how it is constructed. Some researchers state that a DT encompasses only the virtual counterpart of the system, while for others, it encompasses both virtual and real-world systems as well as the architecture connecting them. For example, Lin and Low [14] define DT as "*a virtual representation of the physical objects, processes and real-time data involved throughout a product life-cycle*", whereas Park et.al. [23] define DT as "*an ultra-realistic virtual counterpart of a real-world object*". The ISO 23247 standard "Automation systems and integration — Digital twin framework for manufacturing" defines a DT for manufacturing as a "*fit for purpose digital representation of an observable manufacturing element with a means to enable convergence between the element and its digital representation at an appropriate rate of synchronization*" [9]. Rumpe [27] observes that there are at least 118 different definitions in the literature that concern Digital Twins.

Additionally, many commonly used concepts such as Digital Shadows, Digital Models, Digital Passports, Digital Avatars, Digital Cockpits and Digital Threads, are closely related to "twinning".

In this paper, we introduce the catch-all "Digital T" (pronounced as "Digital Twinning") to refer to all concepts, techniques, architectures, ... related to the "twinning" paradigm. In this paradigm, one or more virtual instances, know as twins, of a System under Study (SuS) are continually updated with the SuS's health, performance, and maintenance status, and this over its entire life-cycle [16].

Our work focuses on the variability that appears when creating and managing Digital T systems, by unifying the most common definitions and viewpoints in the form of *product families* of problems solved by, and conceptual architectures and possible deployments for, Digital T. This allows for the de- and re-construction of the different Digital T variants in a principled, reproducible and partially automatable manner.

In order to illustrate our approach, we apply our approach to the development of Digital T systems for two complementary and small, but representative, use cases, a Line-Following Robot and an Incubator. Our approach is a first step towards generalization to relevant industrial systems.

The rest of this paper is structured as follows. Section 2 discusses variability and product family modelling. Section 3 introduces two simple examples that are representative for industrial systems. Next, Section 4 discusses the possible variations that may occur at the exploration stage. Section 5 then focuses on the design of a Digital T system and introduces some conceptual architecture models. In Section 6, possible deployment architectures are shown for both example cases. Section 7 presents a generic workflow for constructing Digital T systems. Finally, Section 8 concludes the paper.

2 VARIABILITY MODELLING

It is common for multiple *variants* of a product to exist. These variants share some *common* parts/aspects/features/... but do *vary* in others. In the automotive industry for

example, it is common for all sold cars to be (often subtly) different due to small differences in salient *features*. Such variants can often be seen as different *configurations*.

Feature Modelling [11] is widely accepted as a way to explicitly model variability. One possible representation to capture variability in a product family is by means of a Feature Tree (also known as a Feature Model or Feature Diagram). It is a hierarchical diagram that depicts the features that characterize a product in groups of increasing levels of detail. At each level, constraints in a Feature Tree model indicate which features are mandatory and which are optional. Traversing a Feature Tree from its root downwards, features are selected conforming to the constraints encoded in the Feature Tree model. This feature selection leads to a configuration which uniquely identifies an element of the product family. Note that Feature Trees are not the only way to model product families. Wizards can be used to traverse a decision tree. In case the variability is mostly structural, with many complex constraints, Domain-Specific Languages may be used [6].

The notion of a *product family* is used to denote the often vast collection of variants. Product families appear at multiple stages of a development process. The following will describe variability in the goal exploration stage, in the design stage and in the deployment stage.

To illustrate our proposed approach, we apply it to the two simple use cases of Section 3.

3 EXAMPLE CASES

As this work is meant to guide the creation of Digital T systems in a multitude of contexts, two cases are included as running examples: a Line-Following Robot and an Incubator. These cases were chosen as representative (*i.e.,* exhibiting the essential complexity) for their industrial counterparts, an Automated Guided Vehicle and an industrial oven, respectively.

3.1 Line-Following Robot

An *Automated Guided Vehicle* (AGV) is a simple transportation device in an industrial setting. . It is a computer-steered vehicle that allows the transportation of materials, resources and people. For the purposes of this use case, a *Line-Following Robot* (LFR) is used as a simplification of an AGV. The LFR drives over a surface that contains a line (which can be painted, reflective, fluorescent, magnetized, ...), with the sole purpose of following that line as closely as possible. However, unexpected situations (*e.g.,* the robot cannot find the line anymore, a forklift is blocking the robot's trajectory, ...) are difficult to all accommodate for during the (LFR controller) design phase. Unforeseen changes in the LFR's environment is one of the scenarios where twinning provides a solution as it may help detect the anomaly, identify its cause, and suggest adaptation.

Using AGVs reduces the need for conveyor belts, while being highly configurable at the same time. It is an easy-to-understand system that still provides enough essential complexity.

Our LFR is a *nonholonomic, wheeled, differential-drive robot* [25]. Nonholonomicity is due to the fact that the robot has only two controls, but its confiuguration space is three dimensional. The "differential-drive" (DD) aspect indicates that the robot has two wheels next to each other, driven by two distinct motors. In the middle of the LFR, there is a sensor that is able to detect the colour of the surface underneath the robot. From this information, it is possible to infer whether or not the LFR is on the line. The LFR is shown in Figure 1. It was described in detail in [21].

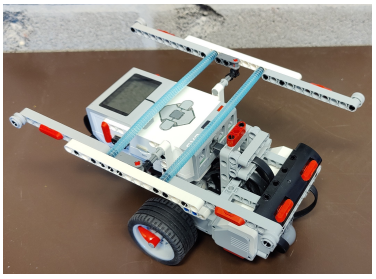

Fig. 1: The Line Following Robot.

The robot drives on a flat surface following a line. In Figure 2, trajectory data for this system are shown. The blue, full line represents the path to follow, as marked on the

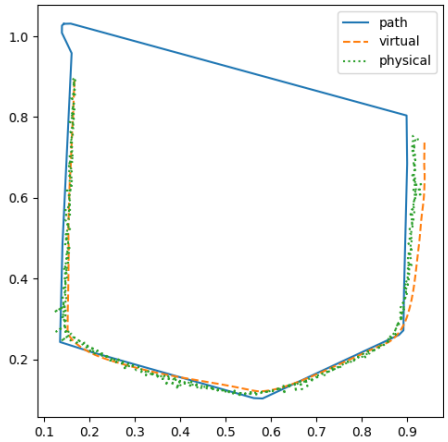

Fig. 2: Trace of an LFR experiment, as shown in a dashboard. Taken from [21].

floor, the orange, striped line identifies the twin's simulation trace and the green, dotted line represents a trace of the Physical Object's position. This position is obtained using machine vision on the data received from a depth vision camera, mounted statically in

a harness above the surface at such a height as to allow the camera's field of view to capture the entire driving range.

3.2 Incubator

A *heating chamber* (*i.e.,* an *industrial oven*) is commonly used in industry for curing, drying, baking, reflow,... It introduces high-temperature processes to the creation of a product. Some ovens allow this product to be transported through the heating chamber on a conveyor belt (or even an AGV).

The temperature in an industrial oven needs to be regulated, as a change in temperature could damage the product. For instance, glazed ceramics could have a completely different colour when baked at the wrong temperature. Additionally, such a system has to react to unpredictable changes in its environment (*e.g.,* complete a safety shutdown when a person enters the chamber during the baking process). This makes an industrial oven an excellent example for the use of a Digital T system.

Similar to the previous use case, a simplification of such a device is made for the purposes of this paper, to focus on the essential Digital T workflows and architectures. An incubator is a device that is able to maintain a specific temperature (or profile over time) within an insulated container. With an appropriate temperature profiel, microbiological or cell cultures can be grown and maintained.

The incubator (see Figure 3) consists of five main components: a thermally insulated container, a heatbed (for raising the temperature), a fan (for circulating the airflow, which, through air convention, allows a uniformly distributed temperature when in steady-state), three temperature sensors (two are used to measure the internal heat, one is used to measure the temperature outside the container – the environment, which is outside our control) and a controller. In our example this controller is similar to a *bang-bang* (or *on/off*) controller, but has to wait after each actuation, to ensure that the temperature is raised gradually.

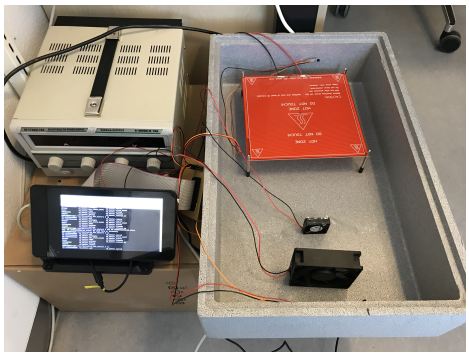

Fig. 3: The Incubator (with lid removed).

In [8], a full description of this incubator is given. Figure 4, adapted from [8], shows an example scenario where the lid of the incubator is opened. This is detected as an

anomaly by a Kalman (tracking) filter [10] (the purple temperature trajectory is the result of the Kalman filter; the blue trajectory is the real temperature as measured inside the incubator.). The Kalman filter uses a model for the prediction of the temperature. Such a model does not consider the thermal dynamics when the lid is open. As a result, when the lid is opened, the predictions start to perform poorly, a fact that can be the basis for anomaly detection. Note that the figure also shows a simulation that runs completely independently of the measured data. The reason the Kalman filter does not perform as poorly as this simulation is because it still takes into account real sensor data. This, in contrast to the simulation, which uses a model of the environment. Also note that, compared to the simulation, after the lid is closed, the simulation has difficulty returning to normal, whereas the Kalman filter, because it uses the measured data, quickly returns to tracking the system behaviour.

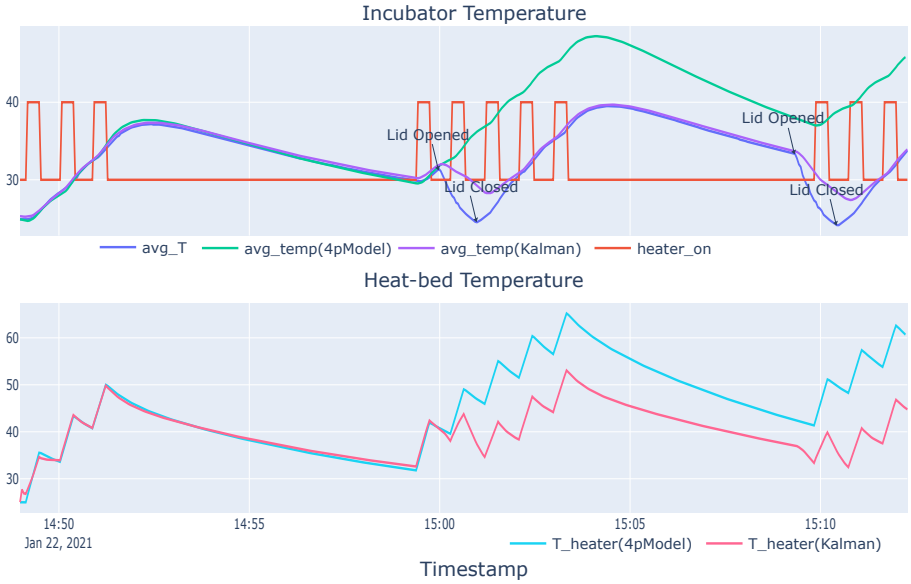

Fig. 4: Example experiment where an anomaly is detected using the Kalman filter. Adapted from [8].

4 EXPLORATION STAGE

Based on our experience with the two use cases and on the extensive Digital T literature, we built feature models for each of the development stages in creating Digital T systems. Note that these are by no means complete, but are rather meant as a starting point, to illustrate our approach. They can therefore be seen as a “base” feature model that can be critiqued and extended by others.

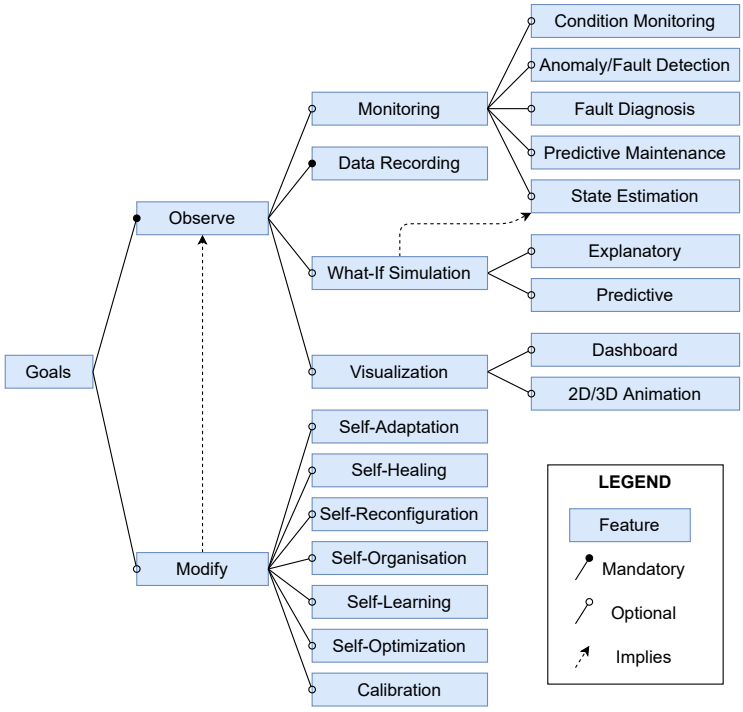

Fig. 5: Feature model of the Goals of a Digital T System.

When creating a solution for a problem (that is to be solved by twinning), it is important to first describe the individual *Properties of Interest* (PoIs) for the problem. These PoIs are attributes (or descriptors) of an artifact that are either logical (*i.e.,* the LFR has wheels) or numerical (*i.e.,* the LFR has 2 wheels). They can be *computed* (or derived) from other artifacts, or *specified* (*i.e.,* defined by a user) [24]. These PoIs describe the *goals* of the system that will solve the problem. Each specific goal corresponds to a specific choice in the variability models for that system. They define the problem space.

A (sub)set of the most common Goals for a Digital T system is shown in Figure 5. Notice the separation of the mandatory "*Observe*" and the optional "*Observe and Modify*". This separation identifies the split between analysis, whereby the real-world system is not modified, and adaptation, where it is.

While the LFR case allows for additional PoIs to be added, we will only focus on the "Dashboard" section for the purposes of this work. The incubator case also supports "Anomaly Detection".

In addition to the selection of the Goals, there is also the Context in which the Digital T system is to be used. This is shown in the feature model of Figure 6. In the first layer, under Context, all features are mandatory. There is always some User, always a Scale, always a Product Lifecycle Stage, ... This is an indication that these are orthogonal dimensions. Feature choices in each of these dimensions can be combined.

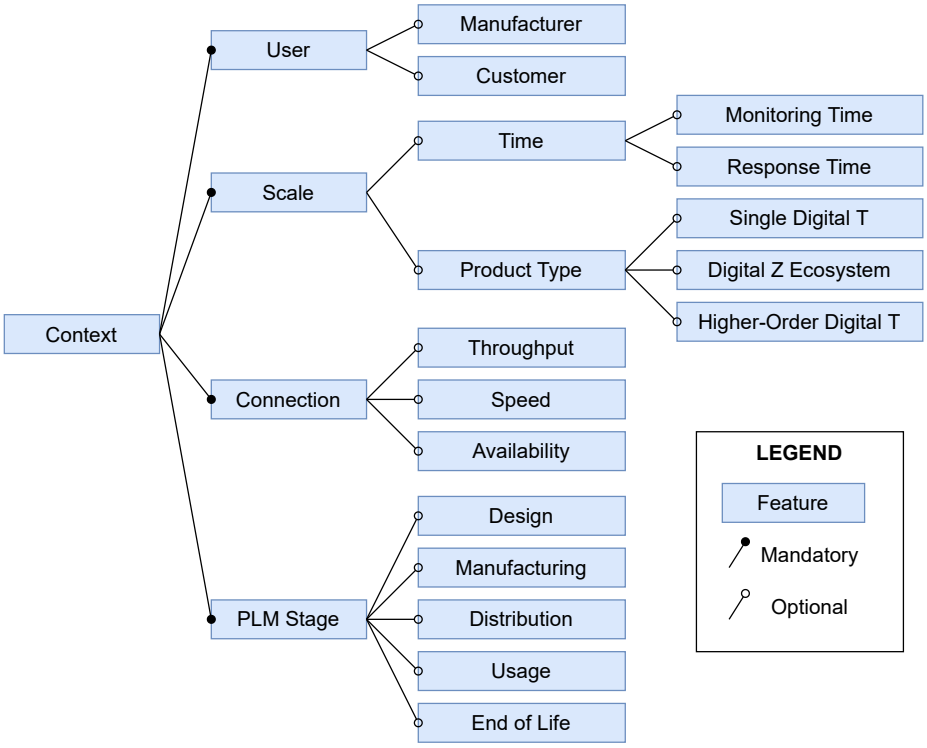

Fig. 6: Feature model of the Contexts for a Digital T System.

Note that the context in which the Digital T system is active constrains downstream choices in the Solution Space.

The LFR case focuses on “Customers”, only requires “Monitor Time” and is a “Single Digital T System”. The “Connection” is not critical, and the Digital T system is meant to work at the “Usage PLM Stage”. The incubator case focuses on “Customers”, requires a certain “Response Time” and is also a “Single Digital T System”. The “Speed” of the “Connection” is important, as it must adapt to anomalies in a timely manner. It is also meant to be used at the “Usage PLM Stage”.

Many different Ilities can be listed and discussed. According to [29], the Ilities are desired properties of systems, such as flexibility or maintainability (usually but not always ending in “ility”), that often manifest themselves after a system has been put to its initial use. These properties are not the primary functional requirements of a system’s performance, but typically concern wider system impacts with respect to time and stakeholders than are embodied in those primary functional requirements.

The main Ilities to be focused on in our example cases are “Testability”, “Repeatability”, “Replicability” and “Usability”.

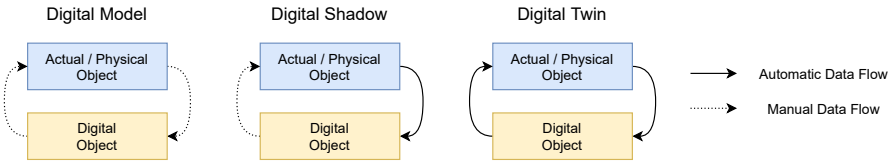

Fig. 7: Three main Design variants of DTs, adapted from [12].

5 DESIGN STAGE

There is variability in the kind of Digital T system we wish to build. [12] defines three Digital T variations, as outlined in Figure 7.

As shown in the figure, each variant contains a *Physical Object* (PO) and a *Digital Object* (DO). The PO represents the System under Study (SuS) within its Environment. The DO represents a virtual copy of the SuS (often in the form of a real-time simulator), trying to mimic its behaviour, *assuming* it is active in the same Environment as the PO. Depending on one's viewpoint and on the application domain, "*physical*" may be an ambiguous term as not all SuS are constructed from what we typically call physical (mechanical, hydraulic, ...) components. The SuS may for example also contain software components. Hence, the more general term "*Realized Object*" (RO) will be used. The same logic can be applied to the DO. For instance, a DO of a train may be modelled using a scale model of the train, instead of a mathematical model used in a simulator. In this case, the DO will be an "*Analog Object*" (AO) or *Analog Twin* instead. For the purpose of this paper, the focus will be on DOs.

The three main Design variants of DTs follow from the nature of the data/information flow between RO and DO. In a *Digital Model* (DM), there is no automated flow of data/information between the objects. The only way data/information is transferred is by a human user. If something changes in the RO, the DO must manually be updated, and vice-versa.

In a *Digital Shadow* (DS) however, the data flow from the RO to the DO becomes automated (*i.e.,* without human intervention). This data, more specifically, is environmental information captured by sensors. This, to ensure that the RO and the DO "see" the same Environment. This is an important step, as it allows the DO to "track" the RO. In Figure 5, the Observe family of goals leads to a DS solution.

Finally, the *Digital Twin* (DT) closes the loop between RO and DO. If something now changes in the DO, the RO will receive an automated update corresponding to this change. This is usually optimization information, fault tolerance notification, predictive maintenance instructions, etc. Note that in the DT case, automation may also refer to the inferencing and decision making without human intervention.

When building a DT, these variants are typically all traversed in different "stages" in of the Digital T workflow. When a system exists as a DM, the introduction of an appropriate data communication connection yields a DS. When the DO of a DS is now expanded to do system analysis and optimizations (and the RO is able to evolve to allow for system adaptation), we obtain a DT. Note that there needs to be an initial plan to build a DT, so the architecture of the RO can be engineered to allow for adaptation.

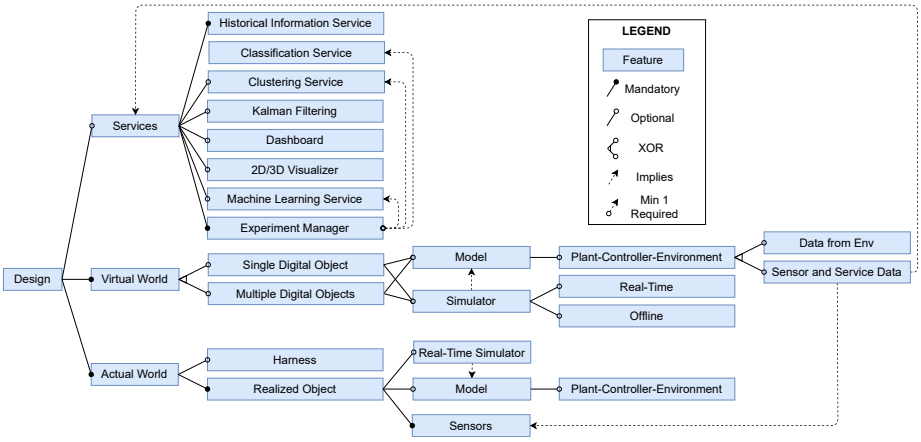

Fig. 8: Variability for the Design of a Digital T System.

If there was no original plan to build a DT, modifying an existing RO may be hard. Making the RO configurable from the outside as an afterthought may introduce security risks.

Similar to the Exploration Stage, we present a Feature Tree to model the variability in the conceptual Design of a Digital T system. This is shown in Figure 8.

Many options exist. Does the realized Digital T system contain a Testing Harness? Is there a single Digital Object in the Virtual World, or are multiple Digital Objects required, one for each Property of Interest. Finally, there are the Services that provide solutions for the Goals presented earlier. There will always be a Historical Information Service (HIS) to store and access historical data of experiments carried out in the past. Additionally, there will be an Experiment Manager that orchestrates Services and Digital Objects.

Figure 7 presents a very high-level view on Digital T architectures. Going into more detail, there must be some component to set up and handle any and all experiments being done with the RO and DO. This will be done by a set of Experiment Managers (EMs) or Assimilation Objects (in case only data if collected from RO and DO). An Experiment can therefore be seen as an execution of a Digital T system such that one or more PoIs can be analyzed. All this information is communicated to the HIS, over which one or more Reasoners may make inferences. Such inferences are important, both in the design of new Digital T systems and in the optimization of future SuS designs.

In Figure 9, a high-level architecture is shown for the DM. The SuS is modelled containing a *Plant-Controller* feedback loop. Both in the RO and in the DO, this SuS interacts with an *Environment*. For the DO, however, this Environment is a modelled mock-up of the real environment. It should interact with the SuS in exactly the same way as the real environment. In practice, the environment models a typical "duty cycle" of the SuS and is based on historical (measured) data. At the heart of a DO is a *Simulator* to produce behaviour traces from the SuS and mock-up Environment models. Note that there is no direct link between RO and DO (only an indirect one, via

the Experiment Managers). Simulation will hence typically not be real-time, but rather as-fast-as-possible.

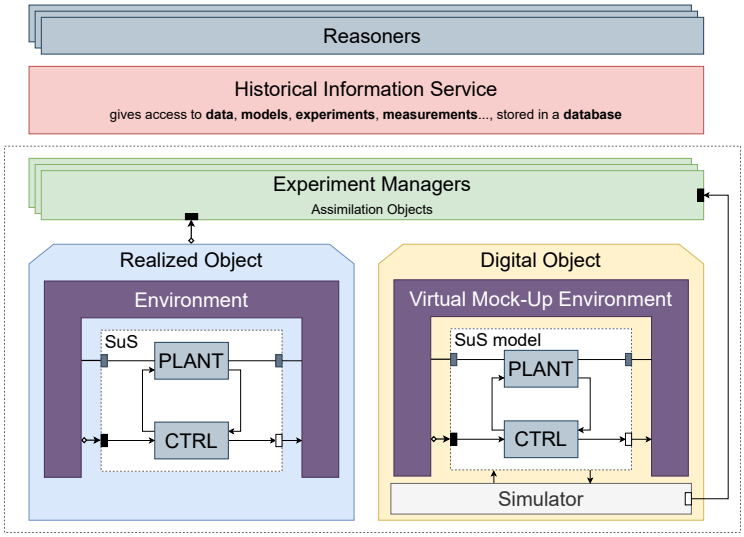

Fig. 9: High-level DM architecture, originally presented in [22].

When taking a closer look at Figure 10, a high-level architecture for a DS, it becomes clear that (in the DO) the mock-up Environment is now (partially) replaced by a data connection from the input of the SuS in the RO. This provides both RO and DO sub-systems with the exact same input and should, therefore, yield identical behaviour in both. Note that deviations are still possible as the plant's behaviour may be influenced by the environment in ways that are not recorded by the sensors in the RO. In that case, the DO still relies on a mock-up (model, possibly based on historical data) of the Environment [4].

Figure 11 shows a similar architecture for a DT. Compared to the DS, a connection from the EM goes towards the inputs of both RO and DO of the SuS. If the EO identifies the need for a change in the RO, it will ask the SuSs in both RO and DO to apply this change.

Notice how Figures 9, 10 and 11 describe variant architectures. They are models in an appropriate Architecture Description Domain-Specific Language (AD DSL). DSLs are more appropriate than Feature Trees when the variability is structural.

[4] The authors are grateful to Francis Bordeleau for pointing this out during Dagstuhl Seminar 22362 on Model Driven Engineering of Digital Twins.

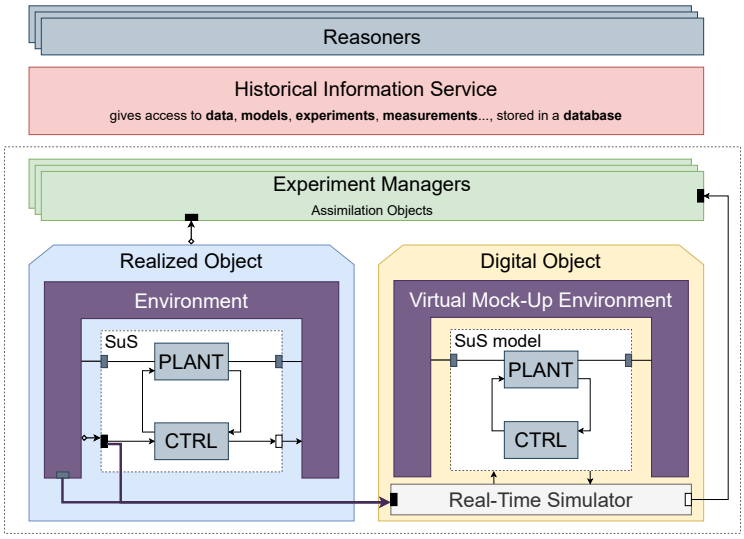

Fig. 10: High-level DS architecture, originally presented in [22].

6 DEPLOYMENT STAGE

For the deployment stage, no feature model has been created. Due to the vast number of possible options, this is left as future work. We plan to still provide feature models, but at an appropriate level of abstraction, in particular, distinguishing between commonly used architectures. Here, deployment diagrams are presented for the current implementation of the two cases. Each component type is the result of a specific choice that has been made in the creation of these systems. Unmarked, dashed arrows denote data flow. For the LFR, the deployment diagram is given in Figure 12. The LFR was constructed using the *LEGO Mindstorms EV3 Core Set (313131)*, which connects to a *Computer* (for which the specifications are not mentioned here) via a TCP/IP connection. Above the LFR, there is a depth vision camera, whose image needs to be processed before a valid representation of the system can be shown in a Dashboard.

Figure 13 shows the deployment diagram for the incubator case. The core of the incubator is a Raspberry Pi that controls the temperature in a Styrofoam box. Inside, there are two temperature sensors to measure the inside temperature. Additionally, there is one temperature sensor on the outside of the box. Control information is communicated to the heatbed and the fan through a relay. On a *Computer* (for which the specifications are not mentioned), multiple OS applications and processes implement anomaly detection and data management.

Even with only two example cases, some commonality is apparent. This will form the basis for a feature model of the Deployment Stage.

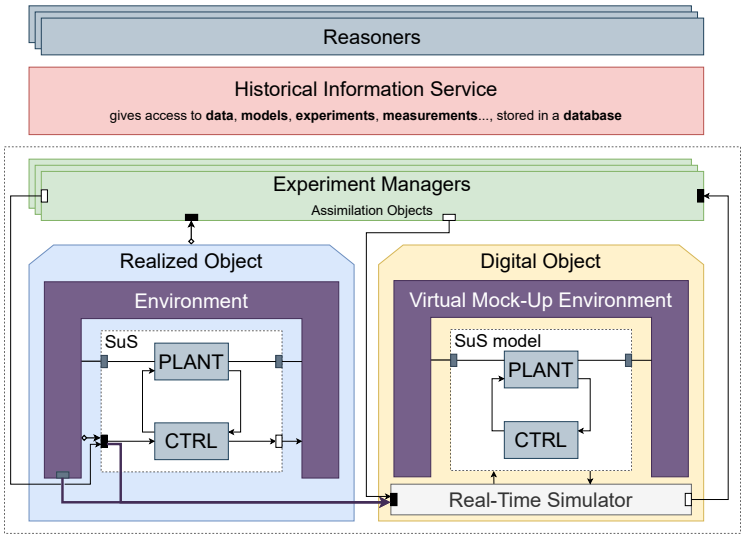

Fig. 11: High-level DT architecture, originally presented in [22].

7 WORKFLOW

Section 4 looked at the required setup and features in the problem space and Section 5 presented the general architectures of Digital T systems. Finally, the two example cases were deployed as described in Section 6. Each of these stages have their own *worflows* and final results. On top of that, the overall creation of the Digital T system also follows a workflow. Such a workflow describes in which order which activities are carried out, on which artifacts. A well-chosen workflow may optimize the overall development time. Note that a workflow or Process Model (PM) follows partly from the constraints imposed by the variability models. Figure 14 shows a PM that yields a Digital T system, going through the previously defined stages. It has been created by analyzing and unifying the workflows followed for both cases from Section 3. The PM shown is modelled in a language similar to a UML Activity Diagram, but following the notations defined in [19]. It describes the order of activities (bold, blue control flow), annotated with their in- and output artifacts (thin, green data flow). The blue roundtangles represent the activities carried out and the green rectangles identify the input/output artifacts. All activities and artifacts have a name and a type, denoted as "`name : Type`". Some of the activities can be automated. This depends on the modelling languages and tools that were used.

In the following, we detail some of the process steps.

First, starting from some "Requirements", the environment in which the Digital T system is active is modelled. Once the environment is available, the plant and controller models can be constructed in the hierarchical "PlantControllerModelling" activity. To allow for more possibilities and variability, no details will be provided on the exact realization of this activity (*i.e.,* , this is a Variant Subsystem). The "CommunicationMod-

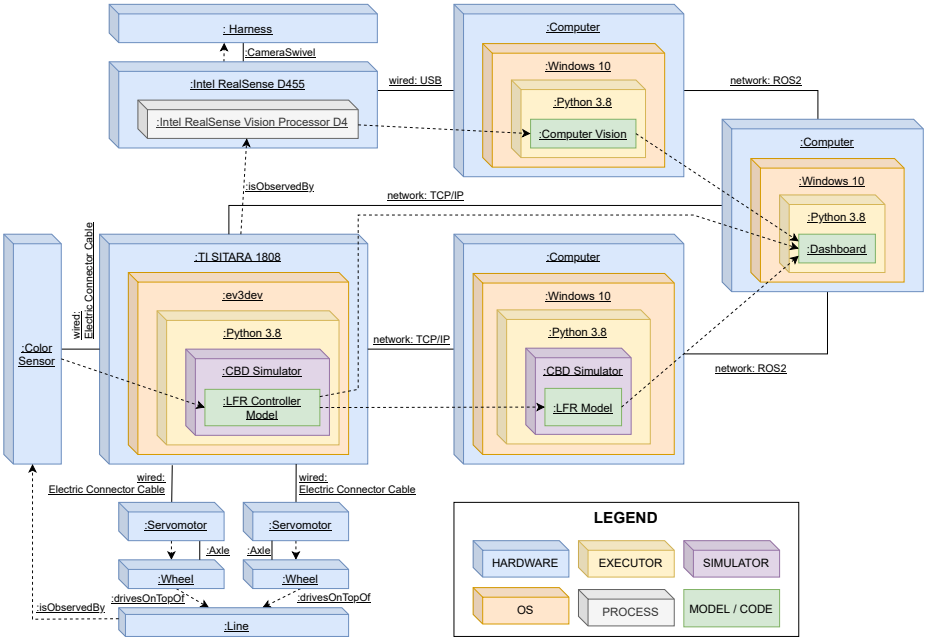

Fig. 12: Deployment mapping diagram for the LFR case.

elling" activity provides for communicating data to an external instance. At the same time as "CommunicationModelling" and "PlantControllerModelling", it is possible to concern the services required for creating Digital T systems.

Next, the system is built, potentially also resulting in a 3D CAD model as an intermediate artifact. Once the system is built, all components are deployed as described by a corresponding deployment diagram. After deployment, the overall functionality is validated and (if need be) new requirements are constructed for a next iteration of this process.

8 CONCLUSIONS AND RELATED WORK

This paper has discussed the variability in creating Digital Ts (*e.g.,* (a family of) Digital Models/Shadows/Twins/...) using Multi-Paradigm Modelling principles [2], at the exploration, design and deployment stage. For the design stage, an architecture was also presented. The feasibility of the presented workflow has been demonstrated by means of two distinct, exemplary cases that are simplifications of often used industrial components (potentially used together). We plan to further explore the various product family models introduced here using the recent Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins [7] as a starting point. In particular, the relationships between the features will be further investigated, with as ultimate goal, to chart and automate as much as possible of the workflow. The link will be made with the Asset

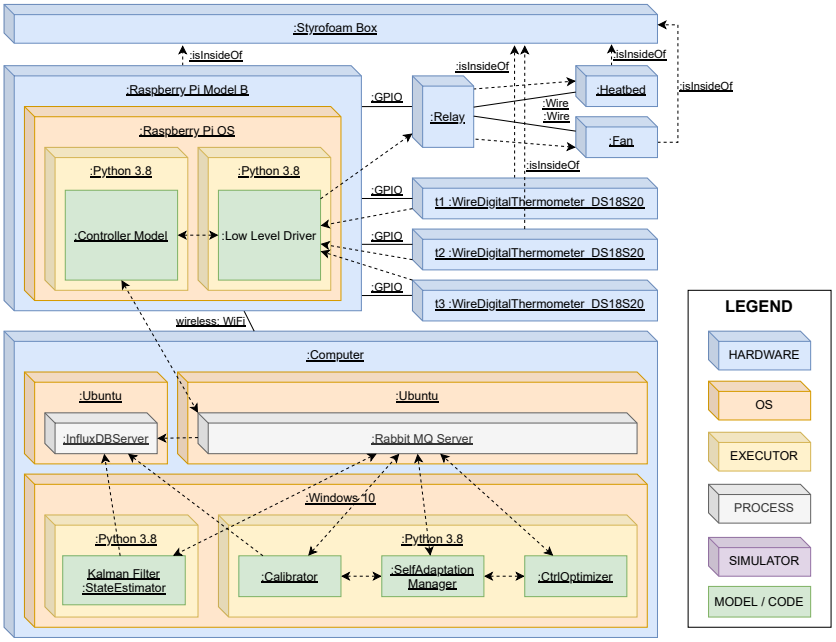

Fig. 13: Deployment mapping diagram for the incubator case.

Administration Shell (AAS). Oakes et.al. [18] provided a mapping of their Digital Twin framework onto the Asset Administration Shell.

ACKNOWLEDGEMENTS

This research was partially supported by Flanders Make, the strategic research center for the Flemish manufacturing industry and by a doctoral fellowship of the Faculty of Science of the University of Antwerp. In addition, we are grateful to the Poul Due Jensen Foundation, which has supported the establishment of a new Center for Digital Twin Technology at Aarhus University.

References

1. Aivaliotis, P., Georgoulias, K., Alexopoulos, K.: Using digital twin for maintenance applications in manufacturing: State of the Art and Gap analysis. In: 2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC). pp. 1–5. IEEE, Valbonne Sophia-Antipolis, France (Jun 2019). https://doi.org/10.1109/ICE.2019.8792613
2. Amrani, M., Blouin, D., Heinrich, R., Rensink, A., Vangheluwe, H., Wortmann, A.: Multi-paradigm modelling for cyber–physical systems: a descriptive framework. Software and Systems Modeling (2021). https://doi.org/10.1007/s10270-021-00876-z
3. Bradac, Z., Marcon, P., Zezulka, F., Arm, J., Benesl, T.: Digital Twin and AAS in the Industry 4.0 Framework. IOP Conference Series: Materials Science and Engineering **618**, 012001 (Oct 2019). https://doi.org/10.1088/1757-899X/618/1/012001

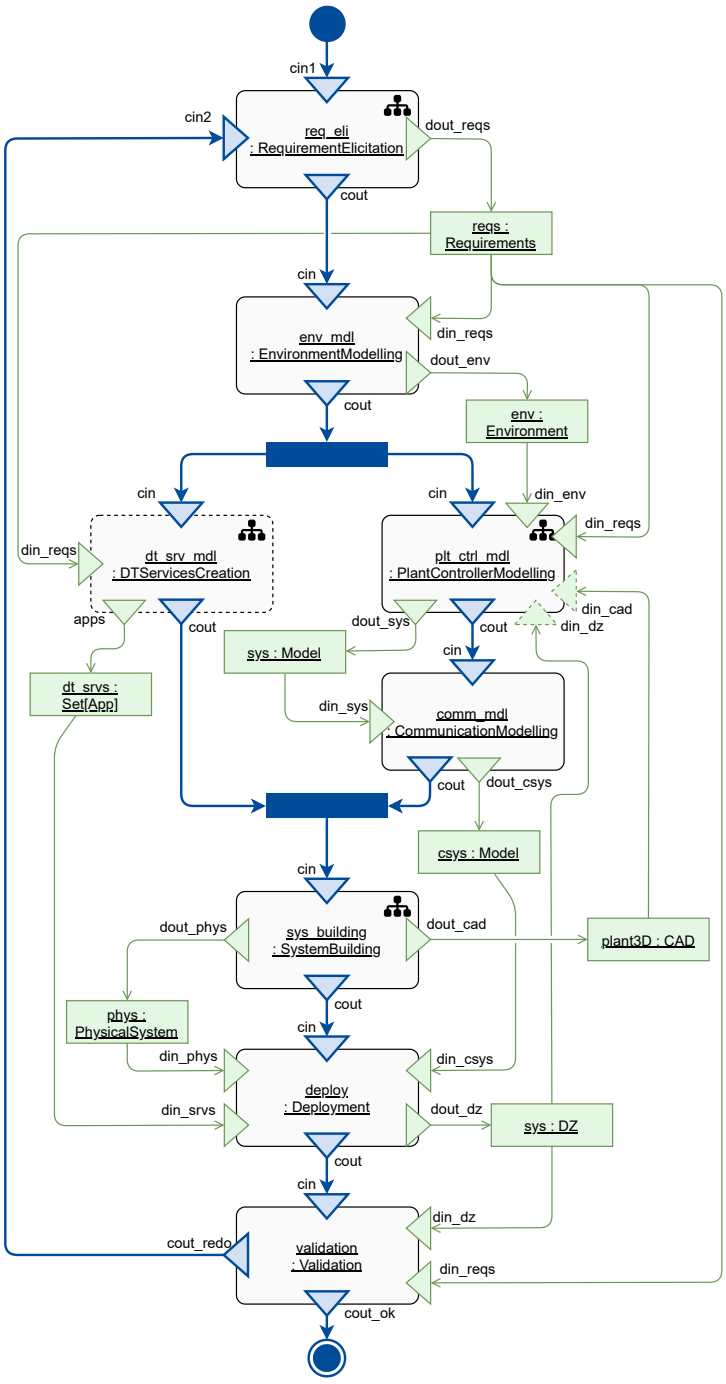

Fig. 14: Workflow model for the construction of a Digital T system. Adapted from [20].

4. Cheng, Y., Zhang, Y., Ji, P., Xu, W., Zhou, Z., Tao, F.: Cyber-physical integration for moving digital factories forward towards smart manufacturing: A survey. The International Journal of Advanced Manufacturing Technology **97**(1-4), 1209–1221 (Jul 2018). https://doi.org/10.1007/s00170-018-2001-2
5. Cimino, C., Negri, E., Fumagalli, L.: Review of digital twin applications in manufacturing. Computers in Industry **113**, 103130 (Dec 2019). https://doi.org/10.1016/j.compind.2019.103130
6. Czarnecki, K.: Overview of generative software development. In: Banâtre, J., Fradet, P., Giavitto, J., Michel, O. (eds.) Unconventional Programming Paradigms, International Workshop UPP, Revised Selected and Invited Papers. LNCS, vol. 3566, pp. 326–341. Springer (2004). https://doi.org/10.1007/11527800_25
7. Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., Wortmann, A.: A Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins. Journal of Systems and Software **193**, 111361 (2022). https://doi.org/https://doi.org/10.1016/j.jss.2022.111361, https://www.sciencedirect.com/science/article/pii/S0164121222000917
8. Feng, H., Gomes, C.a., Thule, C., Lausdahl, K., Iosifidis, A., Larsen, P.G.: Introduction To Digital Twin Engineering. In: 2021 Annual Modeling and Simulation Conference (ANNSIM) (2021)
9. International Organization for Standardization (ISO/DIS): ISO 23247: Automation systems and integration — Digital Twin framework for manufacturing. Tech. rep. (2020)
10. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering **82**(1), 35–45 (03 1960). https://doi.org/10.1115/1.3662552, https://doi.org/10.1115/1.3662552
11. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Carnegie-Mellon University (1990)
12. Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W.: Digital Twin in Manufacturing: A Categorical Literature Review and Classification. IFAC-PapersOnLine **51**(11), 1016–1022 (Jan 2018). https://doi.org/10.1016/j.ifacol.2018.08.474
13. Kutin, A.A., Bushuev, V.V., Molodtsov, V.V.: Digital twins of mechatronic machine tools for modern manufacturing. IOP Conference Series: Materials Science and Engineering **568**, 012070 (Sep 2019). https://doi.org/10.1088/1757-899X/568/1/012070
14. Lin, W.D., Low, M.Y.H.: Concept and implementation of a cyber-physical digital twin for a SMT line. In: 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). pp. 1455–1459 (2019). https://doi.org/10.1109/IEEM44572.2019.8978620
15. Lu, Y., Liu, C., Wang, K.I.K., Huang, H., Xu, X.: Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. Robotics and Computer-Integrated Manufacturing **61**, 101837 (Feb 2020). https://doi.org/10.1016/j.rcim.2019.101837
16. Madni, A.M., Madni, C.C., Lucero, S.D.: Leveraging Digital Twin Technology in Model-Based Systems Engineering. Systems **7**(1), 7 (2019)
17. Negri, E., Fumagalli, L., Macchi, M.: A Review of the Roles of Digital Twin in CPS-based Production Systems. Procedia Manufacturing **11**, 939–948 (Jan 2017). https://doi.org/10.1016/j.promfg.2017.07.198
18. Oakes, B.J., Parsai, A., Meyers, B., David, I., Van Mierlo, S., Demeyer, S., Denil, J., De Meulenaere, P., Vangheluwe, H.: A Digital Twin Description Framework and its Mapping to Asset Administration Shell. arXiv preprint arXiv:2209.12661 (2022)
19. Paredis, R., Exelmans, J., Vangheluwe, H.: Multi-Paradigm Modelling for Model-Based Systems Engineering: Extending the FTG+PM. In: 2022 Annual Modeling and Simulation Conference (ANNSIM). SCS (2022)

20. Paredis., R., Gomes., C., Vangheluwe., H.: Towards a Family of Digital Model/Shadow/Twin Workflows and Architectures. In: Proceedings of the 2nd International Conference on Innovative Intelligent Industrial Production and Logistics - IN4PL,. pp. 174–182. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010717600003062
21. Paredis, R., Vangheluwe, H.: Exploring A Digital Shadow Design Workflow By Means Of A Line Following Robot Use-Case. In: 2021 Annual Modeling and Simulation Conference (ANNSIM) (2021)
22. Paredis, R., Vangheluwe, H.: Towards a Digital Z Framework Based on a Family of Architectures and a Virtual Knowledge Graph. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS) (Dec 2022)
23. Park, H., Easwaran, A., Andalam, S.: Challenges in Digital Twin Development for Cyber-Physical Production Systems. In: Chamberlain, R., Taha, W., Törngren, M. (eds.) Cyber Physical Systems. Model-Based Design, vol. 11615, pp. 28–48. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-23703-5_2
24. Qamar, A., Paredis, C.: Dependency Modeling And Model Management In Mechatronic Design. In: Proceedings of the ASME Design Engineering Technical Conference. vol. 2. Chicago, IL, USA (08 2012). https://doi.org/10.1115/DETC2012-70272
25. Rajamani, R.: Vehicle Dynamics and Control. Springer Science & Business Media (2011)
26. Rosen, R., von Wichert, G., Lo, G., Bettenhausen, K.D.: About The Importance of Autonomy and Digital Twins for the Future of Manufacturing. IFAC-PapersOnLine **48**(3), 567–572 (Jan 2015). https://doi.org/10.1016/j.ifacol.2015.06.141
27. Rumpe, B.: Modelling for and of Digital Twins. Keynote (Oct 2021)
28. Tao, F., Zhang, H., Liu, A., Nee, A.Y.C.: Digital Twin in Industry: State-of-the-Art. IEEE Transactions on Industrial Informatics **15**(4), 2405–2415 (Apr 2019). https://doi.org/10.1109/TII.2018.2873186
29. de Weck, O.L., Roos, D., Magee, C.L., Vest, C.M.: Life-Cycle Properties of Engineering Systems: The Ilities, pp. 65–96. MIT Press (2011)
30. Zhang, H., Ma, L., Sun, J., Lin, H., Thürer, M.: Digital Twin in Services and Industrial Product Service Systems:. Procedia CIRP **83**, 57–60 (2019). https://doi.org/10.1016/j.procir.2019.02.131