Towards a Family of Digital Model/Shadow/Twin Workflows and Architectures

Randy Paredis[1][a], Cláudio Gomes[2][b] and Hans Vangheluwe[1,3][c]

[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*

randy.paredis@uantwerpen.be, claudio.gomes@ece.au.dk, hans.vangheluwe@uantwerpen.be

Keywords: Digital Model, Digital Shadow, Digital Twin, Architecture, Workflow, Variability Modeling

Abstract: Digital Twins (DTs) can be used for optimization, analysis and adaptation of complex engineered systems, in particular after these systems have been deployed. DTs make full use of both historical knowledge and of streaming data from sensors. DTs have been given numerous (distinct) definitions and descriptions in the literature. There is no consensus on terminology, nor a comprehensive description of workflows nor architectures. Following Multi-Paradigm Modelling principles, this paper proposes to explicitly model construction and use workflows of DTs as well as their architectures. We apply the concepts of variability (also known as product family) modeling, in particular to DT workflow and architecture. This allows for the de-/re-construction of the different DT variants in a principled, reproducible and partially automatable manner. To illustrate our ideas, two small use cases are discussed: a line-following robot (representative for an Automated Guided Vehicle) and an incubator (representative for an Industrial Convection Oven). The use cases focus on important systems in an industrial context.

1 INTRODUCTION

Digital Twins (DTs) are increasingly used Industry 4.0 and industrial processes for many purposes such as monitoring, analysis, optimization. While their definition has changed throughout the years, the concept stayed somewhat the same: there exists a digital counterpart of a real-world system that provides information about this system. Usually, this information concerns itself with optimizations and correctional behavior of this system.

Academic and industrial interest in DTs is growing, 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), Product Lifecycle Management (PLM) and many more.

Despite the many surveys on the topic (Rosen et al., 2015; Negri et al., 2017; Kritzinger et al., 2018; Cheng et al., 2018; Park et al., 2019; Zhang et al., 2019; Aivaliotis et al., 2019; Kutin et al., 2019; Bradac et al., 2019; Cimino et al., 2019; Lu et al., 2020; Tao et al., 2019), there is no general consensus on what characterizes a Digital Twin, let alone how it is constructed. There is no one-definition-fits-all and therefore anyone in need of a DT starts from scratch to build (what they believe to be) a DT. The main concern is to create "value" for the user and to ensure minimal re-use of workflows and architectures. To some, a DT defines the virtual counterpart of the system, while for others, it encompasses the full concept of having both virtual and real systems at the same time. For example, Lin and Low (2019) 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. (2019) define DT as "*an ultra-realistic virtual counterpart of a real-world object*".

This paper attempts to unify the most common definitions and viewpoints in the form of a *family* of problems solved by DTs as well as variant workflows and architectures.

Several of the above cited surveys propose reference architectures for DTs. For instance, Bevilacqua et al. (2020) proposes one that shares many commonalities with the other architectures. However, in a do-

[a] https://orcid.org/0000-0003-0069-1975

[b] https://orcid.org/0000-0003-2692-9742

[c] https://orcid.org/0000-0003-2079-6643

main with so much variability for definitions and usage contexts, it is paramount that researchers apply a systematic method to identify DT architectural patterns.

Tekinerdogan and Verdouw (2020) have tried to apply certain patterns to the design of DT and Garnier et al. (2020) identifies a set of patterns for a DT architecture in a specific context. Their work provides a basis which some of the ideas in this paper build on. Van Acker et al. (2021) introduce the notion of *validity frames* to support precise reasoning about valid DT contexts. In (Madni et al., 2019), four different levels for modeling DTs were described. Note that the patterns in the literature often mix problem domain, architecture domain and deployment domain. In our approach, we will separate and relate these.

This paper applies the variability modeling methodology proposed in (Kang and Lee, 2013), to systematically identify commonality and variability in workflows and architectures for Digital Twins. In order to illustrate our approach and make concepts concrete, we apply our approach to the development of Digital Twins for two complementary and small, but representative, use cases. Our approach should be generalizable towards larger systems.

The rest of this paper is structured as follows. Section 2 discusses variability and product family modeling within the context of DT. Section 3 introduces two simple use-cases that are representative for industrial cases. Next, Section 4 discusses DT, starting from variability modeling and presents a breakdown of a DT architecture. Section 5 provides the workflow for constructing DTs. Section 6 describes how engineering knowledge evolves over time and how this can be captured in a Knowledge Graph. Finally, Section 7 concludes the paper.

2 VARIABILITY MODELING

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 every sold car to be (often subtly) different due to small differences in *features*. Such variants can often be seen as different *configurations*.

To manage the often vast collection of variants, the notion of a *product family* is used. Kang and Lee (2013) separate the *Variability Space* in a *Problem Space* and a *Solution Space*, as shown in Figure 1. The variability in the Problem Space is broken down into variability of User Goals and Objectives, Quality Attributes and the Usage Context of the products. Variability in the Solution Space breaks down into variability in the Capabilities, the Operating Environment and the Design of a solution to the problem. The Capabilities define all different actors and their uses in a system. Goals and Objectives drive the Capabilities and Quality Attributes to be used in Quality Assurance. The results applicable in the Solution Space can be realized in an *Artifact Space*, that contains all architectures, workflows, deployment options, modules and components to be used in the realization (often deployment in the context of software) of the solution to the problem. As indicated by the “Drive”, “Mapped To” and “Implemented By” arrows in the figure, there is a natural flow from a Real World Problem to a solution, by making choices in each of the variability sub-components. Based on these choices, the transformations indicated by the arrows can be partially automated. This flow is called a (Software) Product Line in the context of Generative Product Development (Czarnecki et al., 2002).

Feature Modeling (Kang et al., 1990) 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 encode in the Feature Tree model. This leads to a configuration (feature selection) 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 and, in case the variability is mostly structural, Domain-Specific Languages may be used (Czarnecki, 2004).

We propose to use Feature Models to capture the variability in both the Problem Space and in the Solution Space. To illustrate our proposed approach, we apply it to the two simple use cases of Section 3.

3 USE CASES

As this work is meant to guide the creation of DT in a multitude of contexts, some exemplary use cases are included to demonstrate the proposed architectures (Section 4) and workflows (Section 5). A line-following robot and an incubator use case are presented. These cases were chosen as representative (exhibiting the essential complexity) for their indus-

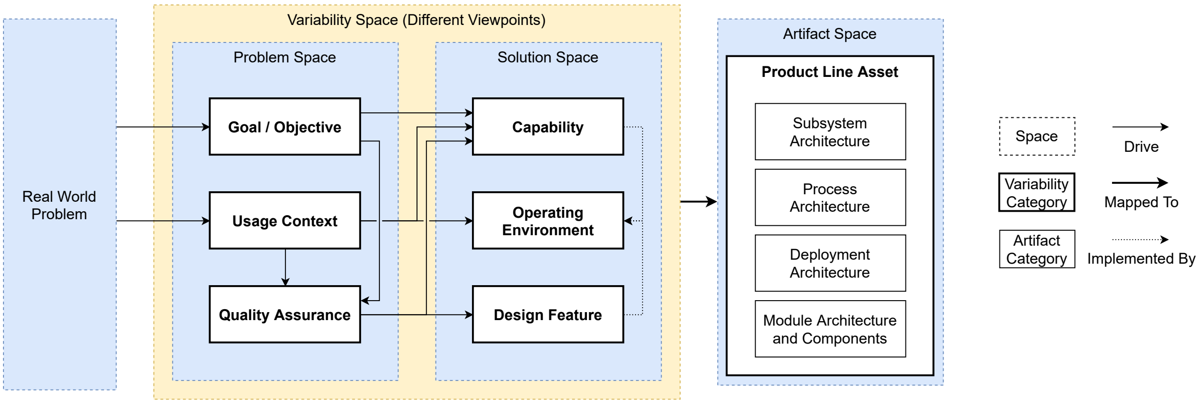

Figure 1: Variability Modeling Space (Kang and Lee, 2013).

trial counterparts: an automated guided vehicle and an industrial oven, respectively.

3.1 Line-Following Robot

A simple transportation device in an industrial setting is an *Automated Guided Vehicle* (AGV). This 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. Uncertainty about unforeseen changes in the LFR's environment are one of the scenarios where a Digital Twin provides a solution.

The proposed architecture and workflow have been applied to construct a Digital Shadow (DS) (see Section 4.1) for the LFR, which is described in detail in (Paredis and Vangheluwe, 2021). The robot drives on a predefined surface. Its position is objectively measured by a depth vision camera, mounted statically above the surface at a height to allow the camera's field of view to capture the entire driving range. In Figure 2, trajectory data for this system are presented. The blue, full line represents the the line to follow, the orange, striped line identifies the DS's simulation results and the green, dotted line represents the Physical Object's identified position.

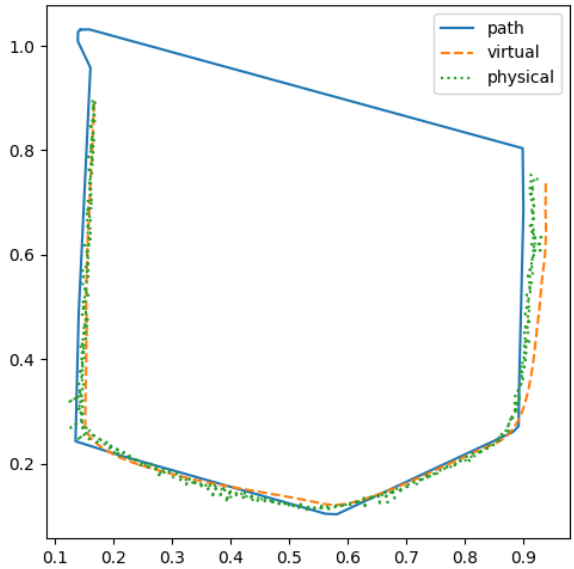

Figure 2: Example experiment results for the LFR.

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 color 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 DT.

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 DT workflows and architectures. An incubator is a device that is able to maintain a specific (variable) temperature within an insulated container. When the temperature is in the right range, a baking process can be performed.

The incubator consists of five main components: an 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 for the temperature outside the container – the environment, which is outside our control) and a controller. This controller is similar to a *bang-bang* (or *on/off*) controller, but it has to wait after each actuation, to ensure that the temperature is raised gradually.

As with the LFR, this case was used to investigate DT architecture and workflow. In (Feng et al., 2021), a full description of this incubator is given. Figure 3, adapted from (Feng et al., 2021), shows an example where the lid of the incubator is opened and that is detected as an anomaly by a Kalman filter (Kalman, 1960) (purple temperature trajectory is 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 dynamics of the temperature when the lid is open. As a result, when the lid is opened, the predictions start to perform poorly, a fact that can be leveraged to perform 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 a harder time returning to normal, whereas the Kalman filter, because it uses the measured data, quickly returns to tracking the system behavior.

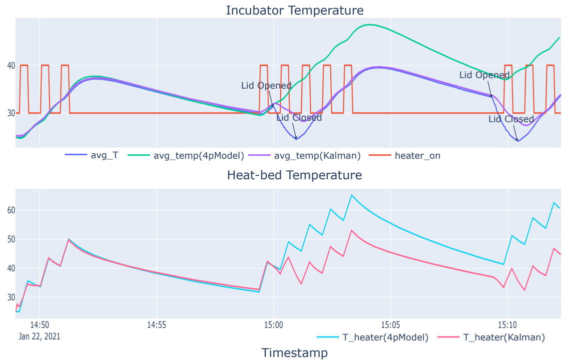

Figure 3: Example experiment where an anomaly is detected using the Kalman filter. Adapted from (Feng et al., 2021).

4 DIGITAL TWINS

The concept of the Digital Twin dates back to a University of Michigan presentation to industry in 2002 (Grieves and Vickers, 2017). Since its origin, the concept has changed such that many research groups and companies introduced their own definition(s), despite many surveys on the topic.

There is, however, a consensus on the goals and objectives a DT can accomplish. Visualization, safety monitoring and analysis, predictive maintenance, fault tolerance, self-adaptation, self-reconfiguration and many more analysis-oriented concepts can be solved using DTs. What is important is that the use of DTs usually concerns complex systems with a certain level of uncertainty due to limited knowledge about the environment (*e.g.,* human interaction).

This leads to the main challenges of this work (and DT research in general): how can a DT be systematically engineered such that variability is allowed, quality ensured and the real-world problem solved?

4.1 DT Variability

Based on our experience with the two use cases and on the extensive DT literature, we built feature models for each of the blocks in Figure 1. Note that these are by no means complete, but rather meant as a starting point to illustrate our approach.

A (sub)set of the most common Goals for a DT is shown in Figure 4. 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.

The Usage Context for a DT is shown in the feature model of Figure 5. In the first layer under Usage 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. The context in which the DT is active will constrain downstream choices in the Solution Space. Figure 6 shows a selection of Quality features. As with the Usage Context, in the first layer under Quality, all features are orthogonal and hence mandatory. The first example dimension concerns the quality of the various data connections in the DT architecture. What these connections are depends on the chosen architecture. The second ex-

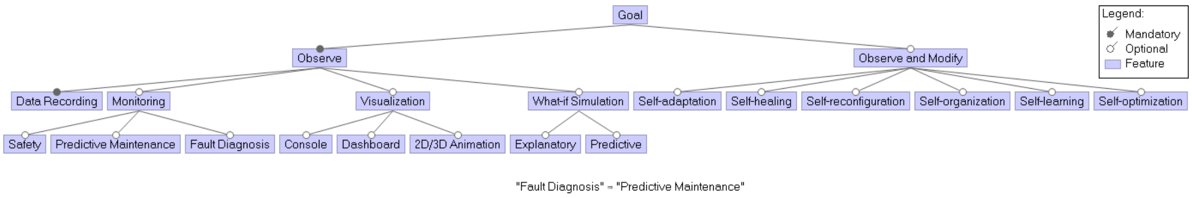

Figure 4: Feature model of the goals of a DT.

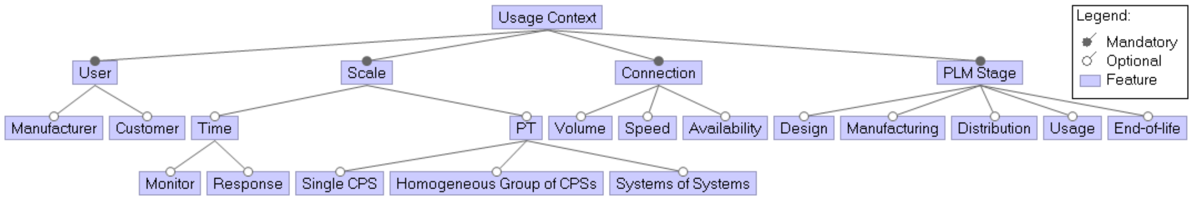

Figure 5: Feature model of the usage contexts of a DT.

ample dimension concerns the *Ilities*. According to (de Weck et al., 2011), 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 Ilities do not include factors that are always present.

4.2 DT Design Variants

For the Solution Space, we currently do not use Feature Trees. Rather, in Figure 7, we show the three main DT Design variants (Kritzinger et al., 2018). As shown in Figure 7, 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 behavior, *assuming* it is active in the exact same Environment as the PO. Depending on one's viewpoint and 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 also contain software components. Furthermore, imagine a (fictional) 3D virtual world in which a SuS exists for which a DT needs to be built. The SuS is not physical in this example; hence, the more general term "*Realized Object*" (RO) is proposed. Alternatively, the same logic can be applied on the DO. For instance, a DO of a train may be modeled using a scale model of the train, instead of a simulation. 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 to conform to the new RO, 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 a precise analysis of the accuracy and validity of both objects. In Figure 4, the Observe family of goals will lead 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 the creation process. 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. If there was no origi-

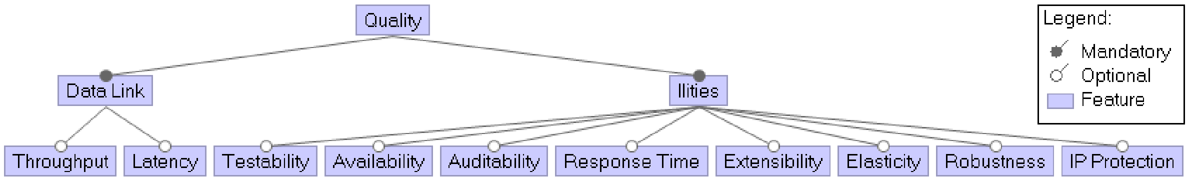

Figure 6: Feature model of the quality assurance of a DT.

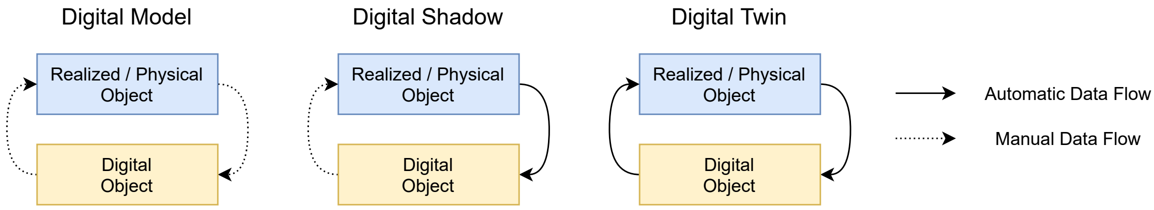

Figure 7: Three main Design variants of DTs.

nal 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.

Figure 7 gives a high-level view on a DM/DS/DT. Technically, this can be expanded by introducing an *External Object* (EO). The EO implements one or multiple external processes that require information from or need to send data to the RO and the DO. The purpose may be visualization, predictive maintenance, analysis, optimization,... Whereas, originally, these processes were typically part of the DO, the system breakdown becomes much more streamlined by logically including them in the EO. The EO's functionality may still, at deployment time, be included in the DO. The EO almost always contains an observation "*harness*" that allows the objective (independent from the DO) collection of data from the RO and its environment. In the LFR, this is an H-bar on which a depth-vision camera is mounted, supplemented with a Kalman filter for a moving window estimator of the RO's state (position and heading). For the incubator, this harness only consists of the sensors that were used. In (Paredis and Vangheluwe, 2021), this harness is called the "*External System Analyzer*" (ESA).

In Figure 8, a high-level architecture is shown for the DM. The SuS is modeled as a *Plant-Controller* model (the faint, dotted arrow identifies an optional feed-back loop). Both in the RO and in the DO, this SuS interacts with an *Environment*. For the DO, however, this Environment is a modeled 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 will model a typical "duty cycle" of the SuS. The DO also includes a *Real-Time Simulator* used to simulate the SuS and Environment models.

Every component of this figure is implemented with the techniques and technologies that best match the selected Goals, Usage contexts and Quality requirements of the system.

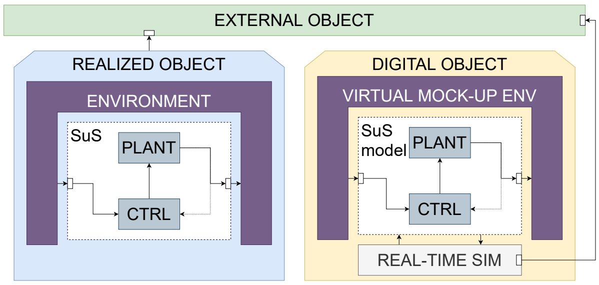

Figure 8: High-level DM architecture.

When taking a closer look at Figure 9, a high-level architecture for a DS, it becomes clear that (in the DO) the mock-up Environment is now 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 behavior in both.

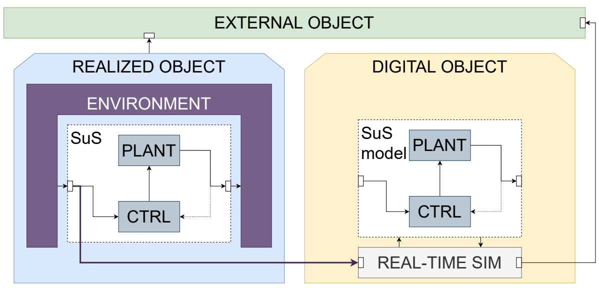

Figure 9: High-level DS architecture.

Figure 10 shows a similar architecture for a DT. Compared to the DS, a connection from the EO goes towards the inputs of both SuS. If the EO identifies the need for a change in the RO, it will ask the SuSs in both RO and DE to apply this change.

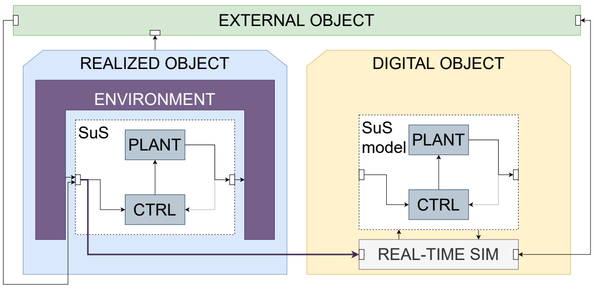

Figure 10: High-level DT architecture.

Multiple technologies exist to deploy communication links. RabbitMQ (https://www.rabbitmq.com/), Ditto (https://www.eclipse.org/ditto/), RTI (https://www.rti.com/), to name a few. Depending on the selected Goals and Quality requirements of the system, a specific technology can be chosen.

Notice how Figures 8, 9 and 10 describe variant architectures. They are models in an appropriate Architecture Domain-Specific Language (DSL). DSLs are more appropriate than Feature Trees when the variability is structural.

Finally, a "Digital X" (where X stands for Model, Shadow, or Twin) is a System in its own right, which means that all Model-Based System Engineering (MSBE) techniques can, and should, be used to design it. Conversely, DTs can be used as components in a system. Each DT component corresponds to a particular aspect of interest of the system. Note that this recursive combination of the specialization and containment relationships implies higher-order DTs.

5 WORKFLOW

The previous section presented the general architecture of DMs/DSs/DTs. In addition to architectures and their deployment, the *worflow* describes in which order which activities are carried out, on which artifacts. A well-chosen workflow may optimize the overall development process time. A workflow or Process Model (PM) follows partly from the constraints imposed by the variability models and their relationships (Drives, Mapped To, Implemented By) in Figure 1. Figure 11 shows a PM that yields a DT with an architecture described in Figure 10. It has been created by analyzing and unifying the workflows followed for both cases from Section 3. The PM shown is modelled as a UML Activity Diagram, describing the order of activities (control flow), annotated with their in- and output artifacts (data flow).

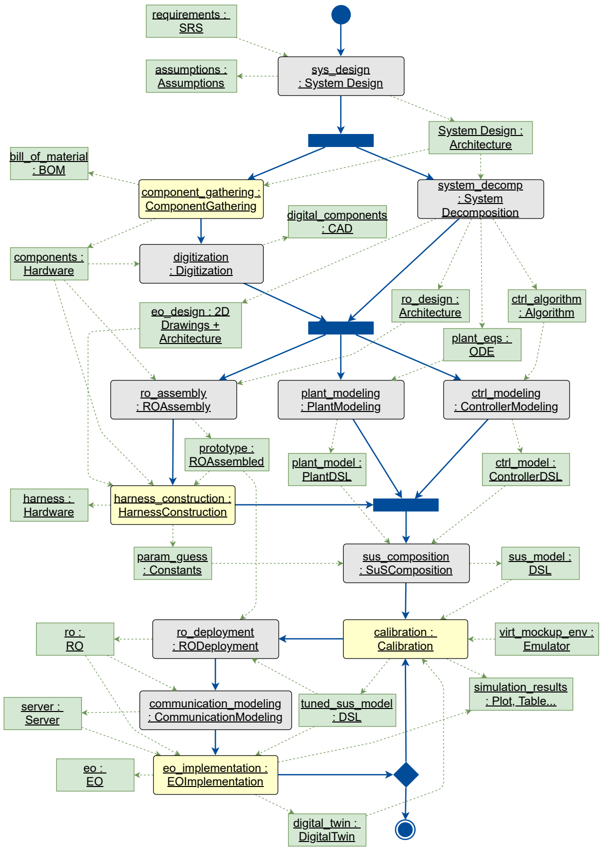

Figure 11: Workflow model for the construction of a DT.

The roundtangles represent the activities carried out and the 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 modeling languages and tools that were used. Yellow roundtangles represent hierarchical activities, indicating that they encapsulate a sub-workflow. For instance, "`component_gathering`" could entail collecting all necessary components from a set of usable parts (*e.g.,* the LFR was built using the *LEGO Mindstorms EV3 Core Set (313131)*) or it could consist of defining the component requirements and buying the corresponding parts (as was the case for the incubator).

In the following, we detail some of the process steps. The "`digitization`" activity may be skipped if there is no need for the *CAD* models of the system. Alternatively, this can be done after the DT has been constructed.

For all DT systems, an external (to the RO) set of sensors is required to obtain the true current state of the SuS. This is called the DM/DS/DT "*harness*". For the incubator, this consists of the temperature sensors. The LFR, however, requires an H-bar construction, on which a depth-camera is mounted. When constructing this harness (the "harness_construction" activity), a set of benchmark tests is performed to help in the calibration of the harness, but also to yield an *initial guess* for the parameters of the system, which will be altered by the "calibration" activity. This activity uses a "virt_mockup_env" (called the "*emulator*" in (Feng et al., 2021)) in which the DO runs. This is a virtual abstraction of the environment that interacts with the DO, just like the RO interacts with the real environment (see Figure 8). This yields a DM that allows testing and continuous integration of the DT before it is connected to the real system. Once an acceptable result is obtained, the RO is deployed and a communication link established between the DO and the RO. Real-world experiments may provide new insights that cause the "calibration" activity to be entered again.

6 KNOWLEDGE EVOLUTION

One may wonder where the knowledge used to construct a DM/DS/DT comes from. This knowledge, when explicitly represented, takes the form models in various formalisms, including historical data from earlier experiments. We use a Knowledge Graph (KG) to store this knowledge (and made available in a KG server). The KG is used in the design of a DM/DS/DT Experiment, by means of the variability models and workflow described earlier. Figure 12 shows an example of the interaction between the KG

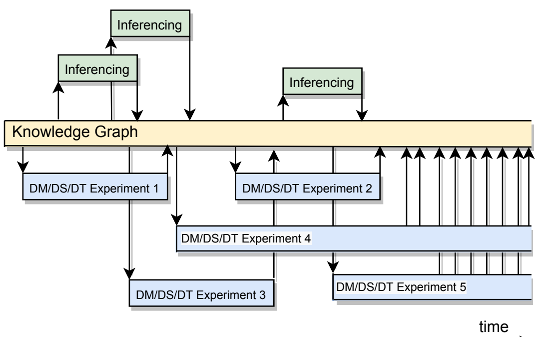

Figure 12: KG and DM/DS/DT Experiment Interaction.

(server) and DM/DS/DT experiments. Given a particular user question Q, the data/information/knowledge in the KG is used to design a DM/DS/DT experiment architecture that, when deployed as experiment E, can provide an answer A to the question. The DM/DT/DS may use historical information and/or it may interact in real-time with real-world physical assets. Experiments 1, 2 and 3 in the figure are bounded in time (also known as terminating). When they finish, their results (*i.e.,* the triple (Q,E,A) is stored in the KG. Long-running "streaming" experiments 4 and 5 will store intermediate, partial answers in the KG. Experiment 5 stores periodically, whereas Experiment 4 does so when deemed relevant (*e.g.,* whenever sufficient new information has become available). It is noticed that many experiments can be performed in parallel. When setting up an experiment to answer a question, all information available in Knowledge Graph at that time, included that which was added by earlier experiments, can be used. Asynchronous inferencing may also update the KG, by for example turning data into information into knowledge.

7 CONCLUSIONS

This paper has presented an architecture and a workflow for creating (a family of) Digital Models/Shadows/Twins using Multi-Paradigm Modelling principles (Amrani et al., 2021). The feasibility of the approach has been demonstrated by means of two distinct use cases (which may be combined in an industrial setting) that are simplifications of often used industrial components. A Knowledge Graph has been introduced as a knowledge repository from which an experiment is constructed to answer questions. This DM/DS/DT experiment then provides an answer to the question, which get added to the Knowledge Graph. We plan to further explore the various product family models introduced here. More importantly, the relationships between the features will be further investigated, with as ultimate goal, to automate as much as possible of the workflow.

ACKNOWLEDGEMENTS

This research was partially supported by Flanders Make, the strategic research centre 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 Centre for Digital Twin Technology at Aarhus University.

REFERENCES

Aivaliotis, P., Georgoulias, K., and Alexopoulos, K. (2019). 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)*, pages 1–5, Valbonne Sophia-Antipolis, France. IEEE.

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

Bevilacqua, M., Bottani, E., Ciarapica, F. E., Costantino, F., Donato, L. D., Ferraro, A., Mazzuto, G., Monteriù, A., Nardini, G., Ortenzi, M., Paroncini, M., Pirozzi, M., Prist, M., Quatrini, E., Tronci, M., and Vignali, G. (2020). Digital twin reference model development to prevent operators' risk in process plants. *Sustainability (Switzerland)*, 12(3).

Bradac, Z., Marcon, P., Zezulka, F., Arm, J., and Benesl, T. (2019). Digital Twin and AAS in the Industry 4.0 Framework. *IOP Conference Series: Materials Science and Engineering*, 618:012001.

Cheng, Y., Zhang, Y., Ji, P., Xu, W., Zhou, Z., and Tao, F. (2018). 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.

Cimino, C., Negri, E., and Fumagalli, L. (2019). Review of digital twin applications in manufacturing. *Computers in Industry*, 113:103130.

Czarnecki, K. (2004). Overview of generative software development. In Banâtre, J., Fradet, P., Giavitto, J., and Michel, O., editors, *Unconventional Programming Paradigms, International Workshop UPP, Revised Selected and Invited Papers*, volume 3566 of *LNCS*, pages 326–341. Springer.

Czarnecki, K., Østerbye, K., and Völter, M. (2002). Generative programming. In Núñez, J. H. and Moreira, A. M. D., editors, *Object-Oriented Technology, ECOOP 2002 Workshops and Posters*, volume 2548 of *LNCS*, pages 15–29. Springer.

de Weck, O. L., Roos, D., Magee, C. L., and Vest, C. M. (2011). *Life-Cycle Properties of Engineering Systems: The Ilities*, pages 65–96. MIT Press.

Feng, H., Gomes, C. a., Thule, C., Lausdahl, K., Iosifidis, A., and Larsen, P. G. (2021). Introduction To Digital Twin Engineering. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*.

Garnier, J.-L., Bachatene, H., and Nowodzienski, P. (2020). Architecture Patterns for Digital Twins in Space Applications. Presentation at the AFNeT Standardization Days: https://download.afnet.fr/ASD2020/ASD2020-13a-DigitalTwin-JeanLucGarnier.pdf, accessed: 2021-05-28.

Grieves, M. and Vickers, J. (2017). Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In *Transdisciplinary Perspectives on Complex Systems*, pages 85–113. Springer.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45.

Kang, K. and Lee, H. (2013). Variability Modeling. In *Systems Software and Variability Management*, pages 25–42.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University.

Kritzinger, W., Karner, M., Traar, G., Henjes, J., and Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022.

Kutin, A. A., Bushuev, V. V., and Molodtsov, V. V. (2019). Digital twins of mechatronic machine tools for modern manufacturing. *IOP Conference Series: Materials Science and Engineering*, 568:012070.

Lin, W. D. and Low, M. Y. H. (2019). 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)*, pages 1455–1459.

Lu, Y., Liu, C., Wang, K. I.-K., Huang, H., and Xu, X. (2020). Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and Computer-Integrated Manufacturing*, 61:101837.

Madni, A. M., Madni, C. C., and Lucero, S. D. (2019). Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems*, 7(1):7.

Negri, E., Fumagalli, L., and Macchi, M. (2017). A Review of the Roles of Digital Twin in CPS-based Production Systems. *Procedia Manufacturing*, 11:939–948.

Paredis, R. and Vangheluwe, H. (2021). Exploring A Digital Shadow Design Workflow By Means Of A Line Following Robot Use-Case. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*.

Park, H., Easwaran, A., and Andalam, S. (2019). Challenges in Digital Twin Development for Cyber-Physical Production Systems. In Chamberlain, R., Taha, W., and Törngren, M., editors, *Cyber Physical Systems. Model-Based Design*, volume 11615, pages 28–48. Springer International Publishing, Cham.

Rosen, R., von Wichert, G., Lo, G., and Bettenhausen, K. D. (2015). About The Importance of Autonomy and Digital Twins for the Future of Manufacturing. *IFAC-PapersOnLine*, 48(3):567–572.

Tao, F., Zhang, H., Liu, A., and Nee, A. Y. C. (2019). Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics*, 15(4):2405–2415.

Tekinerdogan, B. and Verdouw, C. (2020). Systems Architecture Design Pattern Catalog for Developing Digital Twins. *Sensors*, 8(18):5103.

Van Acker, B., Mertens, J., De Meulenaere, J., and Denil, J. (2021). Validity Frame Supported Digital Twin Design of Complex Cyber-Physical Systems. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*.

Zhang, H., Ma, L., Sun, J., Lin, H., and Thürer, M. (2019). Digital Twin in Services and Industrial Product Service Systems:. *Procedia CIRP*, 83:57–60.