Towards a Product Line Architecture for Digital Twins

Jérôme Pfeiffer*, Daniel Lehner†, Andreas Wortmann*, Manuel Wimmer†
* Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW)
University of Stuttgart
Seidenstraße 36, 70174 Stuttgart, Germany
{firstname.lastname}@isw.uni-stuttgart.de
† Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT)
Institute for Business Informatics - Software Engineering
Johannes Kepler University Linz, Science Park 3, 4040 Linz, Austria
{firstname.lastname}@jku.at

***Abstract*—Digital twins are a new kind of software systems for which corresponding architectures in different engineering domains have emerged for enabling the efficient interaction of software systems with physical systems to realize cyber-physical systems (CPS). To facilitate the development of digital twins, various software platforms emerged in recent years, which often come with a certain architecture for the developed systems together with a set of domain-specific languages (DSLs) that help domain experts to configure the platform and implement the digital twins. This results in a set of architectures and DSLs which are currently used to realize the various concerns of digital twins. Thus, creating a comprehensive digital twin for a given system requires the combination of several architectures and DSLs, which is challenging as (i) the components of the different architectures have to be combined on a technological level, and (ii) the concerns specified with the different DSLs are developed in isolation which potentially leads to inconsistencies, especially during the evolution of digital twins.**

To tackle these challenges, we outline our vision of a product line architecture that explicitly specifies the different concerns of digital twins and their alignment on both, the technological level considering the different architectural elements as well as on the language level considering the different language elements. As a result, glue code that is currently required to compose the individual features together into particular digital twin systems is automatically generated. We demonstrate the applicability of this approach by (i) specifying an example product line architecture for selected structural and behavioral concerns of digital twins, and (ii) configuring an existing digital twin based architecture for self-adaptive systems based on this product line architecture by (iii) applying the selected platforms realizing these concerns to a smart room use case. Finally, we discuss expected benefits of the presented approach, such as plug-&-play of digital twin modules, as well as sketch out future work to realize the presented vision.

***Index Terms*—Digital twins, domain-specific languages, product lines, software integration**

I. Introduction

Today Digital Twins (DTs) are utilized in many domains, such as manufacturing [1], injection molding [2], or even farming [3], leveraging different use cases, such as predictive maintenance, reactive planning [4], or self-adaptation [5]. As a result, the different usages of DTs have resulted in software architectures to satisfy these different purposes [6]–[8]. In these architectures, DTs are leveraged to, e.g., enable a bi-directional synchronization to a physical system (i.e., get/send data from/to the physical system) [1], or to perform experiments based on a formal description of the expected behavior of a physical system [9]. Various platforms emerged to provide tooling to support the development of different concerns of DTs such as providing access to the structure [10] or behavior [4]. As features of DTs usually have to be specified by domain experts which are not necessarily experts in software engineering, the emerging platforms are often model-driven, i.e., the software is configured by models conforming to a Domain-Specific Language (DSL) [10]. Thus, they imply a certain architecture on the developed systems.

For integrating these different concerns into an overarching software architecture, glue code needs to be written to glue the individual parts, i.e., platforms and realized components, together. This is particularly challenging because different platforms usually realize DTs with different architectures and DSLs. Thus, the glue code to integrate different concerns has to be rewritten for every platform combination on both levels: (i) language level to bridge the DSLs and (ii) technology level to bridge the different software components. Although there are already approaches to automate the integration of different DSLs, e.g., [11], or the integration of software components, e.g., [12], these solutions currently do not consider the coupling of both, DSLs and software at the same time.

To tackle this challenge, we present our vision of a product line architecture [13] that explicates and aligns different concerns of DTs as DT features. This product line can be used to integrate DT features into different software architectures based on the specific available platforms and their imposed architectures for particular features. The novelty of this approach lies in the integrated reuse of both DSLs and software components for building new architectures that leverage common features of DTs. We showcase our idea with an initial product line architecture of DTs in the context of self-adaptive systems. More precisely, this product line architecture describes (i) the system structure and data of a physical system using a monitor,

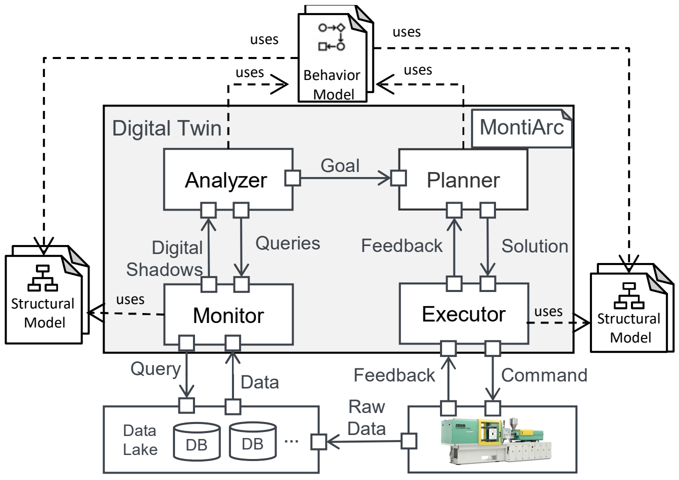
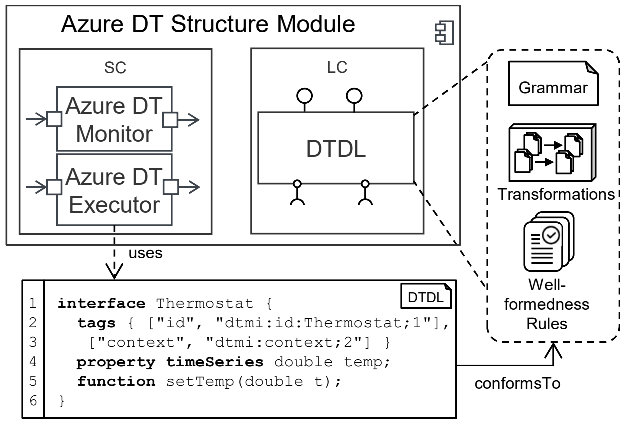


Fig. 1. An exemplary software architecture for digital twins realizing the MAPE-K loop based on [5].



```
interface Thermostat {
  tags { ["id", "dtmi:id:Thermostat;1"],
   ["context", "dtmi:context;2"] }
  property timeSeries double temp;
  function setTemp(double t);
}
```

Fig. 2. Implementation of a DT module comprising software and language components for Microsoft Azure Digital Twin Definition Language (DTDL).

executor, and a structural DSL based on platforms provided by Microsoft Azure[1], Amazon Web Services[2], and Eclipse[3], as well as (ii) the behavior of a physical system described in a state chart used by a planner. We demonstrate how a chosen structural and behavioral platform is configured for the example of a smart building use case.

II. Motivating Example

The approach presented in this paper is demonstrated in the exemplary context of DT architectures realizing self-adaptive systems using the MAPE-K pattern [14]. A reference software architecture realizing this pattern is depicted in Fig. 1 [5]. Such DT architectures consist of four components. A `Monitor` receives the data emitted from the cyber-physical system (CPS), and converts it into a digital shadow, an abstraction of the actual data. The `Analyzer` then receives these shadows and identifies potential anomalies in the data, and subsequently notifies the planner to react on this anomaly. The `Planner` creates a plan to correct the anomaly and sends the plan to the `Executor`, which then sends a control command that is processable by the connected CPS. Additionally, the `Planner` stores successfully executed plans in the knowledge base. In [5], this architecture is realized with MontiArc [15], a component & connector architecture description language. It uses component types with interfaces consisting of typed ports, that are connected by unidirectional connectors for transmitting data. Components can be decomposed into multiple connected subcomponents. Specific to this architecture is the usage of different modeling languages to configure each of the components. For instance, the `Monitor` and `Executor` use data and constraint models, e.g., class diagrams to interact with the CPS. Furthermore, the `Planner` has a behavior model, e.g., a statechart to react on erroneous behavior of the CPS recognizable through the data. Thus, there is (i) a relation between the software components of the architecture that send data to another, e.g., the monitor sends digital shadows to the analyzer, and (ii) a relation between the models that are used by these components, i.e., the statechart using the types defined in the data models, to react on values on the data. Thus, when exchanging the `Planner` using statecharts with a `Planner` using, e.g., PDDL [16] or other planning formalisms, not only the planner and its specification have to be replaced, but also the glue code that aligns the planning DSL and software with other parts of the architecture needs to be updated, i.e., (i) relations from statecharts to other models, (ii) relations from the statechart language to other DSLs, and (iii) the code in components that send data to the `Planner` (i.e., the `Analyzer` sends the goal information). This is also the case when replacing a data modeling language such as class diagrams with, e.g., the proprietary data modeling languages of cloud vendors that are emerging [10], [17]. This exchange is complex, and thus, cost- and time-intensive as the engineer has to understand the specifics of both, software and language implementations and their reuse to realize their combination.

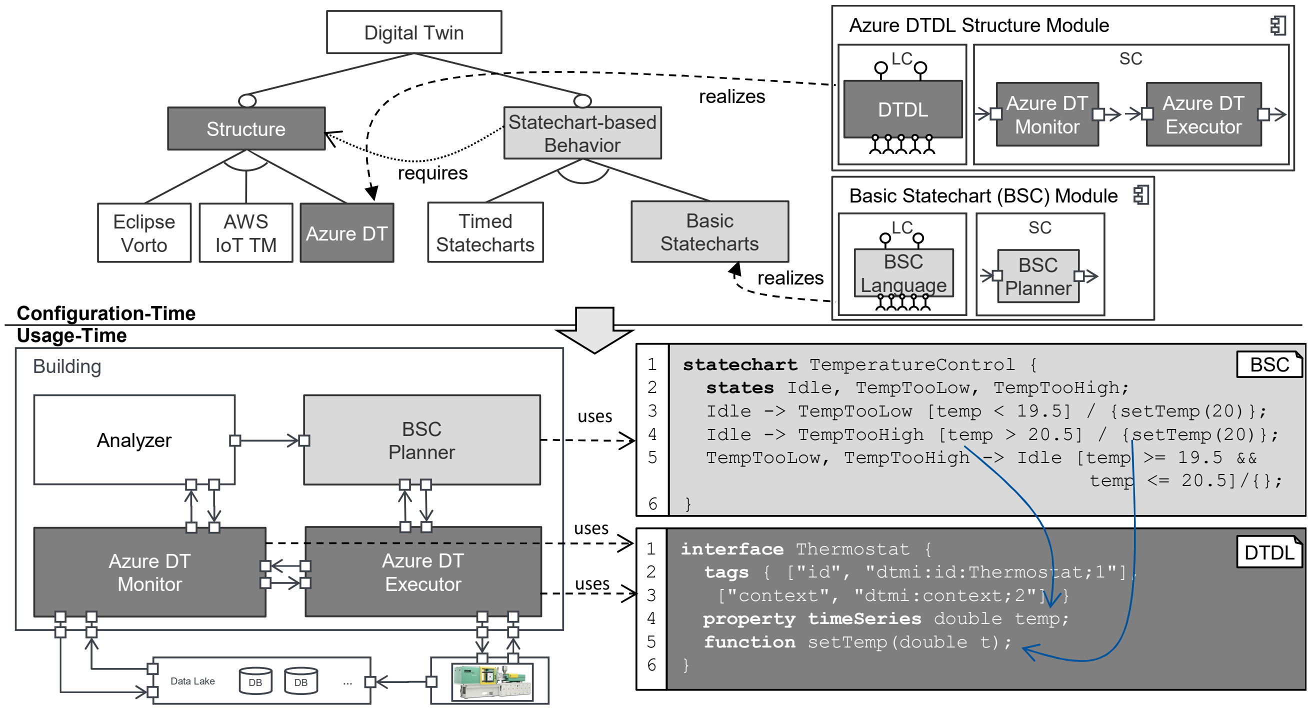
III. Vision

This section describes our envisioned approach towards a product line architecture for DTs. This includes a novel approach for modules for DT architecture implementations comprising both software components (SC) as well as language components (LC) in one module of reuse. These modules are then generalized into DT features and aligned into a product line to derive a specific software architecture for DTs, potentially tailored to a specific use case, and derive particular implementations based on the selected module implementations.

A. Digital Twin Modules

The main entity of our approach are the DT modules. Each module consists of (i) language components [11], and (ii) software components that use or are configured by models conforming to the language component.

[1]https://azure.microsoft.com/products/digital-twins/
[2]https://aws.amazon.com/iot-twinmaker/
[3]https://www.eclipse.org/vorto/



```
statechart TemperatureControl {
  states Idle, TempTooLow, TempTooHigh;
  Idle -> TempTooLow [temp < 19.5] / {setTemp(20)};
  Idle -> TempTooHigh [temp > 20.5] / {setTemp(20)};
  TempTooLow, TempTooHigh -> Idle [temp >= 19.5 &&
                                   temp <= 20.5]/{};
}
```

```
interface Thermostat {
  tags { ["id", "dtmi:id:Thermostat;1"],
   ["context", "dtmi:context;2"] }
  property timeSeries double temp;
  function setTemp(double t);
}
```

Fig. 3. Overview of the envisioned solution comprising a product line. Components are realized through DT modules. By the configuration of the product line, a specific DT architecture and language is generated.

Fig. 2 shows such a DT module for the structural concern, i.e., structure of the data exchanged between CPS and DT. The module comprises a language component for the Microsoft Azure Digital Twin, that can be used to describe structural aspects of a CPS [10]. A language component [11] contains the syntax, in terms of a grammar and semantic definition including well-formedness rules and code generators, of a DSL. Language components can be reused and extended by other language components via their interface. We distinguish between a provided interface, i.e., language concepts that are exposed to be reused, and a required interface that exposes concepts to be realized by provided extensions of other language components and enables language evolution, e.g., new concepts of the Digital Twin Definition Language (DTDL)[4]. When composing two language components, a new language component emerges, where both languages are either embedded in each other [11], enabling modelers to build one model that comprises concepts of both languages, or aggregated [18] where one language can reference concepts of another language. In this paper, we only showcase aggregation due to space limitations. Besides, the module comprises a software component, in our example, a monitor component that parses data of the CPS using a DTDL model. For the software component to be compatible with the existing architecture for MAPE-K DTs, the software component has to realize the ports imposed by the concern for which it is intended to be inserted, e.g., the DTDL monitor has to provide the same interface regarding input and output ports, and their types as the monitor of the architecture in Fig. 1.

[4]https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md

B. A Product Line Architecture for Digital Twin Engineering

In our envisioned approach, various DT modules as described above are arranged into a product line architecture for DTs. From this product line architecture, product owners can select one variant for a specific DT kind, in our example an architecture realizing the MAPE-K pattern for DTs as proposed in [5]. Selecting a feature of the product line architecture requires to select one implementation of the DT module available for this feature, and thus, effects the reuse of both language and software components at the same time. After the selection at configuration-time, the software components are available in the particular implementation of the architecture, and the corresponding languages are integrated with each other via the mechanism of composing language components.

Fig. 3 depicts an excerpt of our envisioned product line architecture for the creation of MAPE-K DTs. In the upper part, the product line architecture with its features is depicted. For the structural description of the underlying system, the product owner can decide between Eclipse Ditto, AWS IoT TwinMaker, or Azure DTDL realization. The feature of Azure DTDL is realized by a module (cf. Fig. 2) comprising a language component containing the language definition of DTDL, and two software components for monitor and executor that use models of this language. Besides, the product line architecture has a feature behavior where either timed-statecharts [19], with transitions based on timing constraints, or basic statecharts (BSC) with states, transitions, guard conditions, and actions, for the definition of behavior can be selected. In our example, the product owner selects the features Azure DTDL and basic statecharts at configuration-time. Afterwards, a DT software architecture (cf. [5]) is generated that comprises the software components of the modules of the selected features of our product line. In addition, these components use models conforming to the languages defined in the language components of the modules of the selected features. Furthermore, like the software components that are able to interact with each other via their interface, the language components are composed, which has the effect that the models of the basic statechart language can now reference DTDL models.

In our example, the DT of a building is created, where the statechart defines when to increase or decrease the room temperature, and the DTDL model provides the information about the datatypes and methods available to do so. Thus, our approach facilitates the reuse of existing DT software and language implementations, by using DT modules, e.g., the Azure DTDL structure module and basic statechart module. Their alignment in a product line architecture enables product owners to configure their desired DT features at coconfiguration-time to generate a DT architecture that comprises the software and models to realize the selected features for a given physical system. Hence, the product owner does not need to take care of how to compose the software components or the modeling languages that are used by these components.

IV. Discussion & Conclusion

In this paper, we presented our envisioned approach to realize a product line architecture for DT software architectures. The product line architecture consists of features for concerns of the DT. These features are realized by DT modules containing language and software components. We demonstrated our vision with a product line architecture for structural and behavioral concerns of a DT architecture realizing the MAPE-K pattern. This enables developing new digital twin architectures based on existing software and language components going beyond the borders of single digital twin platforms. However, it is up to future work to evaluate our concept with other DT architectures and their modules.

In the future, we aim to identify abstract commonalities between the realizations of these concerns, e.g., structural aspects of DTs are similarly described [10], and derive generic software components and languages to further extend them with use case specifics for enabling portability. Moreover, we aim to clarify the constituents of the interface of DT modules to finally connect and compose these modules as well as to extend our product line architecture with more features for DTs [20]–[22]. Finally, our approach may be generalized beyond the application of DTs.

Acknowledgement

This work has been supported by the German Federal Ministry of Economic Affairs and Climate Action (BMWK, SofDCar – 19S21002L), as well as the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG).

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] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, and A. Wortmann, "Model-driven development of a digital twin for injection molding," in *CAiSE*. Springer, 2020, pp. 85–100.

[3] R. G. Alves, G. Souza, R. F. Maia, A. L. H. Tran, C. Kamienski, J. Soininen, P. T. Aquino, and F. Lima, "A digital twin for smart farming," in *GHTC*. IEEE, 2019, pp. 1–4.

[4] M. Eisenberg, D. Lehner, R. Sindelár, and M. Wimmer, "Towards reactive planning with digital twins and model-driven optimization," in *ISoLA*. Springer, 2022, pp. 54–70.

[5] T. Bolender, G. Bürvenich, M. Dalibor, B. Rumpe, and A. Wortmann, "Self-adaptive manufacturing with digital twins," in *SEAMS*. IEEE, 2021, pp. 156–166.

[6] E. Ferko, A. Bucaioni, and M. Behnam, "Architecting digital twins," *IEEE Access*, vol. 10, pp. 50 335–50 350, 2022.

[7] M. Caporuscio, F. Edrisi, M. Hallberg, A. Johannesson, C. Kopf, and D. Perez-Palacin, "Architectural concerns for digital twin of the organization," in *ECSA*. Springer, 2020, pp. 265–280.

[8] M. M. Bersani, C. Braghin, V. Cortellessa, A. Gargantini, V. Grassi, F. L. Presti, R. Mirandola, A. Pierantonio, E. Riccobene, and P. Scandurra, "Towards trust-preserving continuous co-evolution of digital twins," in *ICSA-C*. IEEE, 2022, pp. 96–99.

[9] M. Schluse, M. Priggemeyer, L. Atorf, and J. Rossmann, "Experimentable digital twins - streamlining simulation-based systems engineering for industry 4.0," *IEEE TII*, vol. 14, no. 4, pp. 1722–1731, 2018.

[10] J. Pfeiffer, D. Lehner, A. Wortmann, and M. Wimmer, "Modeling capabilities of digital twin platforms-old wine in new bottles?" in *ECMFA*, 2022.

[11] A. Butting, J. Pfeiffer, B. Rumpe, and A. Wortmann, "A compositional framework for systematic modeling language reuse," in *MODELS*, 2020, pp. 35–46.

[12] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. C. Cavalcanti, and S. R. de Lemos Meira, "Twenty-eight years of component-based software engineering," *J. Syst. Softw.*, vol. 111, pp. 128–148, 2016.

[13] D. Batory, "Product-line architectures," in *Smalltalk and Java Conference*, 1998.

[14] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[15] A. Haber, J. O. Ringert, and B. Rumpe, "Montiarc-architectural modeling of interactive distributed and cyber-physical systems," *arXiv preprint arXiv:1409.6578*, 2014.

[16] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "PDDL-The Planning Domain Definition Language," *Tech. Rep.*, 1998.

[17] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, and M. Wimmer, "Digital twin platforms: requirements, capabilities, and future prospects," *IEEE Software*, vol. 39, no. 2, pp. 53–61, 2021.

[18] J. Pfeiffer and A. Wortmann, "Towards the black-box aggregation of language components," in *MODELS-C*. IEEE, 2021, pp. 576–585.

[19] Y. Kesten and A. Pnueli, "Timed and hybrid statecharts and their textual representation," in *FTRTFT*. Springer, 1992, pp. 591–620.

[20] 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," *J. Syst. Softw.*, p. 111361, 2022.

[21] S. Koch, E. Hamann, R. Heinrich, and R. H. Reussner, "Feature-based investigation of simulation structure and behaviour," in *ECSA*. Springer, 2022, pp. 178–185.

[22] I. Ruchkin, S. Samuel, B. Schmerl, A. Rico, and D. Garlan, "Challenges in physical modeling for adaptation of cyber-physical systems," in *WF-IoT*. IEEE, 2016, pp. 210–215.