Towards Ontological Service-Driven Engineering of Digital Twins

Bentley Oakes
Polytechnique Montréal
Montréal, Canada
bentley.oakes@polymtl.ca

Claudio Gomes
Giuseppe Abbiati
Aarhus University
Aarhus, Denmark
claudio.gomes@ece.au.dk
abbiati@cae.au.dk

Eduard Kamburjan
University of Oslo
Oslo, Norway
eduard@ifi.uio.no

Elif Ecem Bas
R&D Test Systems
Hinnerup, Denmark
eeb@rdas.dk

Sebastian Engelsgaard
LORC
Munkebo, Denmark
se@lorc.dk

ABSTRACT

The systematic engineering of Digital Twins (DTs) requires the establishment of clear methodologies supported by intelligent tooling. We propose an approach to guide the user in the creation and deployment of services for DTs utilizing ontologies and workflows. In our approach, the user selects a desired DT service from an array of options. This selection is then used to suggest a) enablers and models to place in the DT, and b) development and deployment workflows for the DT service. The aim is to provide DT engineering guidance to assist non-software engineering experts to develop DT services more rapidly with less effort. We describe our initial work on applying this approach to a derived version of an industrial wind turbine generator case study, utilizing openCAESAR for ontology definition and enacting the workflows with Jupyter notebooks.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; • **Software and its engineering**;

KEYWORDS

digital twins, ontologies, DT services, wind turbine testing, guided software engineering, recommendation, workflows

ACM Reference Format:
Bentley Oakes, Claudio Gomes, Giuseppe Abbiati, Eduard Kamburjan, Elif Ecem Bas, and Sebastian Engelsgaard. 2024. Towards Ontological Service-Driven Engineering of Digital Twins. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3652620.3688261

1 INTRODUCTION

The concept of Digital Twins (DTs) has evolved from its beginning in product design [10] to cover a multitude of definitions [23]. Here, we define DTs as *virtual representations of complex (cyber-)physical systems*, and refer to the system as the Physical Twin (PT). One of the key characteristics of DTs is that they provide *services* to their user and to the connected PT [7, 21]. To name just a few services, these range from visualization of the DT, to monitoring, optimization, and fault injection [3]. These services connect the DT to the PT by providing key *insights* and *actions* which modify the PT or produce actionable information.

Problem. While DT engineering is an evolving field, there is still a gap in mature systematic approaches and especially those that guide the user along the definition of the DT services. From a recent survey on open-source DT platforms [6], the majority of the guidance provided is in the form of documentation and examples, without explicit tool support. We thus argue that there is a research gap in defining and implementing guidance for the practitioner on how to build the DT services.

In particular, we take the position that a low-level approach which 'starts from the bottom' and focuses on modeling of the data and connections is not appropriate for a wide range of DT practitioners and stakeholders. Rather than software engineers, these DT stakeholders include management, local and national governments, engineers from other scientific domains, and potentially citizen developers. These stakeholders can benefit from DTs and their services (such as optimization), yet they may not have the technical skills to effectively use current DT platforms. In particular, we see a high barrier in the requirements for core modeling, simulation, and software engineering skills, including the creation and calibration of system models, and deployment to a running DT platform.

Approach. In this paper, we discuss our on-going work in engineering DTs "top-down". We propose to leverage the unique structure of DTs as being a *constellation* of services, enablers, models and data [7], as shown in Figure 1. We envision users composing DTs by selecting the DT services required, and being guided by tool support through the creation and deployment of the DT constellation.

Our approach relies on *ontologies* to explicitly capture and represent the heterogeneous domain knowledge required to build DT services: a) the selection of the tools/enablers, models, and data required, b) the workflow(s) for developing the DT service and the required models, and c) as structure for a *knowledge graph* [19] providing model management capabilities.

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688261

Contribution and Structure. Our contribution is a first definition and application of our *ontologically-based approach for engineering DT services*. We a) overview the approach, including a selection of the ontologies and workflows developed, and b) present prototype tooling for its application. The running example presented in Section 2 is the creation of a DT service (conformance monitoring) for wind turbine generator testing. Section 3 overviews our approach, including our current view on the broad steps for the engineering of DT services. Section 4 discusses the application of our approach to a derived version of an industrial case study. Section 5 presents related work while Section 6 concludes.

2 WIND TURBINE TESTING EXAMPLE

The running example of this study is derived from the 16 MW Highly Accelerated Life-time Testing (HALT) testbench for wind turbine generators operated by the Lindo Offshore Research Center (LORC), Denmark[1]. A large rotor, composed of three blades connected to a hub, converts wind kinetic energy into torque feeding an electric generator, which converts the work produced by the spinning rotor into electric energy. A drivetrain is used to vary the rotational speed of the generator shaft w.r.t. the rotor shaft.

The HALT testbench exposes the drivetrain-generator system to an equivalent load as that experienced by a real wind turbine throughout its entire life-cycle (up to 20 years), but compressed in a few months of continuous testing. The HALT testbench utilizes a hexapod to exert bending moments, shear forces and axial load to the rotor shaft while an electric motor imposes torque as it was produced by the rotor. As a result, only the wind turbine generator nacelle is tested and the rotor (whose diameter could be up to 250 m) is numerically simulated. In the context of experimental testing, the drivetrain-generator system is a Device Under Test (DUT) while the hexapod is a Test Loading Unit (TLU).

DIGIT-BENCH DT. The model of the HALT testbench (termed the DIGIT-BENCH DT) with a parametrized DUT is being developed within the Digit Bench project[2]. For the sake of simplicity, the rotor dynamics are neglected and the model is constructed to represent only the energy conversion process from shaft motion to electric power generation. Specifically, the model represents the 1) electric motor of the HALT system, 2) drivetrain, and 3) electric generator.

The main DT service we present here is the *DUTMonitor* service, which outputs a measure of conformance of the PT and this model of the dynamics of the TLU coupled with the DUT (see Figure 2 for the context diagram). The purpose of this service is to detect discrepancies with the PT during an experimental campaign, which could be a symptom of a fault in the configuration of the TLU or the DUT, or a change in the coupling stiffness between the TLU and DUT. The service must therefore detect these discrepancies early, and warn the user to stop the experiment. This DT service is thus crucial to prevent structural damage and/or catastrophic failures.

To detail the DIGIT-BENCH DT, we present here some DT characteristics from the reporting framework of Gil *et al.* [7]. Figure 1 shows a) the *insights*, *actions*, and *data* between the PT and the DT, and b) the dependency relationships of services, enablers, and models/data inside the DT constellation.

[1]Please see https://www.lorc.dk/test-facilities for more on the HALT testbench.
[2]https://digit.au.dk/research-projects/digit-bench

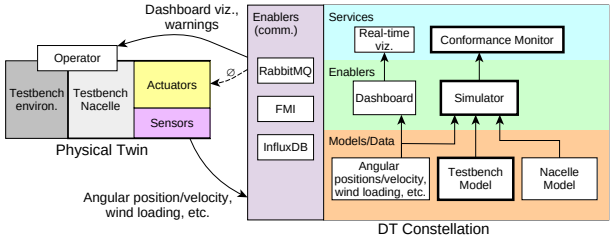

Figure 1: DIGIT-BENCH PT and DT *constellation*.

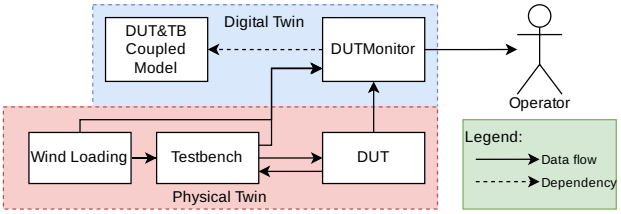

Figure 2: DIGIT-BENCH DT service context diagram.

DT Services, Insights/Actions. Other than the *DUTMonitor service*, the *real-time visualization service* displays the conformance data from the DUTMonitor, model predictions, and data coming from the PT sensors as *insights* for the user. There are (currently) no automatic *actions* taken by the DT on the PT.

Tooling and Enablers. InfluxDB[3] provides a framework for the creation of dashboards. The simulator has been exported as a *Functional Mockup Unit* (FMU[4]) from the OpenModelica tool [5].

The communication between the DT services is achieved using RabbitMQ[5]. The DUTMonitor service is implemented in Python and uses the Maestro2 orchestrator [11] to simulate the coupled model as a co-simulation [9]. During the development and decoupling of the model (described later), we have also used the FMPy library[6].

DT Data and Models. For confidentiality reasons, we present only the relevant data exchanged between the PT and the DT, consisting of the angular position, velocity, and wind loading. This provides the inputs required to run the simulation of the energy generated by the PT as well as its dynamics.

The model used in the DUTMonitor service is a simplified version of the real world model. It is used to predict the energy generated by the nacelle. This model represents a coupling of two sub-models: 1) the TLU model, and 2) the DUT model, with wind loading conditions and the energy regeneration control as inputs.

These models are shown coupled together in Figure 3, in the Open Modelica Connection Editor[7]. The TLU model on the left of Figure 3 is a rotational inertia driven by a torque generated by a speed PI controller. The speed settings come from the wind loading conditions. The coupling with the DUT model is achieved by a rigid

[3]https://www.influxdata.com/
[4]https://fmi-standard.org/
[5]https://www.rabbitmq.com/
[6]https://github.com/CATIA-Systems/FMPy
[7]https://www.openmodelica.org/

rotational spring/damper. The DUT model is similar except there is a breaking torque to represent the electrical energy generation.

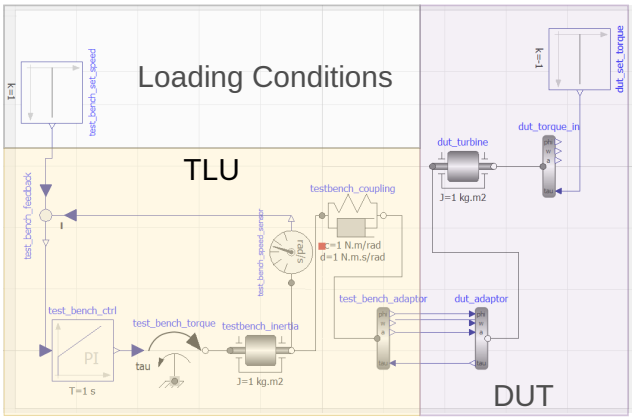

Figure 3: Coupled model overview.

In the following, we discuss the construction and deployment of the *conformance monitor service* and its supporting *testbench model*, highlighted in bold in Figure 1. In particular, we describe how our approach guides the user in a) decoupling this coupled model, and b) deploying the conformance monitor as a service on the DT.

3 DT SERVICE ENGINEERING APPROACH

This section discusses our proposed approach for engineering DT services, as diagrammed in Figure 4. The essence of the approach presented in the following is that 1) the user selects the desired DT service from a "menu". This selection informs the ontologies and knowledge graphs to query, providing: 2) guidance for the types of tools and enablers to used within the DT service, and 3) the service and model development and deployment workflows. These workflows are enacted in 4) tooling to guide the user along the modeling, simulation, model management, and deployment steps necessary to engineer the new DT service.

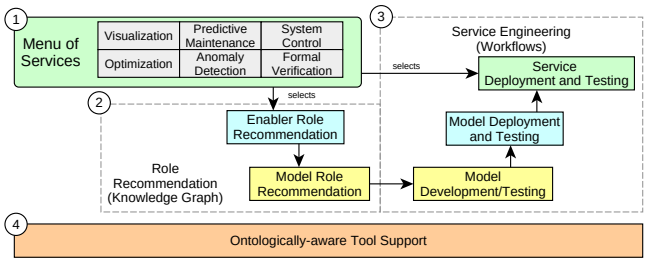

Figure 4: Overview of our DT service engineering approach.

3.1 Step 1 - Service Selection

In our approach, we assume that the user has a well-defined problem or question for which the DT will provide an answer [19]. This corresponds to a "business case" for the DT, developed through detailed requirements gathering and ideation. The user must then connect their business case with the DT service to be implemented which provides the necessary value. For example, a need to visualize the PT for training purposes is addressed by a training service and a visualization service. In our industrial project, the DT must be able to sense when the PT behavior does not match the predicted behavior to prevent structural damage. Therefore, the conformance monitor service provides the necessary value.

In our approach, we propose that the practitioner selects their desired DT service from a "service menu", as demonstrated with a selection of possible high-level services in the top-left box of Figure 4. This service menu approach is inspired by others: the *DT purposes* from Figure 6 in Dalibor *et al.* [3], and the Digital Twin Capabilities Periodic Table (CPT) from the Digital Twin Consortium[8].

These inspirations present high-level categories for DT services. However, we foresee that a finer-grained selection is required. In our vision, the DT service selection menu should provide a level of detail similar to the *services layer* represented in Figure 1. That is, it should offer options such as 'visualization', 'optimization', etc. As discussed below, each service corresponds to a defined workflow, where the user enters in relevant details, such as which models and their values are considered for conformance monitoring.

Ontological Approach. We propose utilizing multi-layered ontologies and knowledge graphs to capture domain knowledge for engineering particular DT services. This includes: a) based on the service, recommending types of enablers and models to be placed in the DT constellation, and b) providing workflows such that the user can engineer each selected DT service and the necessary models.

We are motivated to use an ontological approach in this work for two main reasons. First, the use of the openCAESAR platform allows for agile and rigorous ontology definition [4]. Over a meta-modelling approach, this offers improved consistency checking, inferencing, and explicit semantics. Second, we believe that a multi-layered ontology approach better captures the multi-disciplinary nature of DTs (software, mechanical, electrical, etc.), including the heterogeneous information required for DT engineering (structural, behavioral, requirement, technological, etc.).

Currently, we have three layers of ontologies for recommendations and workflow definition which have been iteratively created to support the functionality required in the project tooling and to represent the DIGIT-BENCH DT components. As openCAESAR supports robust ontology layering and federation [4], we do not expect major difficulties in extending and aligning these ontologies as our project proceeds. Thus over time, we will provide additional services, workflows, and recommendations for our users to follow.

3.2 Step 2 - Role Suggestion Ontology

Our approach aims to provide guidance to the user on how to select the enablers and models to place in their DT constellation. We represent the types of enablers and models that support a service [18] using the concept of *roles*, which we store in a knowledge graph.

A selection of the OML *description* model (corresponding to ABox statements) for the DIGIT-BENCH DT from Figure 1 is shown in Lst. 1. It defines a `conformanceService` which is typed by the concept `ServiceRole`. This `conformanceService` is `enabledBy` the

[8] https://www.digitaltwinconsortium.org/initiatives/capabilities-periodic-table/

`simulatorEnable`, which is typed by an `EnablerRole`. In turn, the `simulatorEnable` `requires` three instances of type `modelRole`.

```
1  instance conformanceService : dtdeploy_system:ServiceRole [
2     dtdeploy_system:enabledBy simulatorEnable
3     dtdeploy_system:deployedUsing conformServiceProcess ]
4  instance visualizationService : dtdeploy_system:ServiceRole [
5     dtdeploy_system:enabledBy dashboardEnable ]
6  instance dashboardEnable : dtdeploy_system:EnablerRole[
7     dtdeploy_system:requires dataRole ]
8  instance simulatorEnable : dtdeploy_system:EnablerRole[
9     dtdeploy_system:requires dataRole
10    dtdeploy_system:requires nacelleRole
11    dtdeploy_system:requires testbenchRole ]
12 instance nacelleRole : dtdeploy_system:ModelRole
13 instance testbenchRole : dtdeploy_system:ModelRole [
14    dtdeploy_system:deployedUsing TestBenchProcess ]
15 instance dataRole : dtdeploy_system:ModelRole
```

Listing 1: A portion of our recommendation knowledge graph, linking services, enablers, models, and processes.

We also trace the deployment processes in the knowledge graph, made explicit as workflows, for service roles and model roles. This is demonstrated by lines 3 and 14 in Lst. 1. Our approach thus provides traceability by linking the DT services to its enablers, the models, and the workflows required and enacted. The knowledge graph can be queried by our tooling to find the models which have yet to be developed to support a particular service, and the workflow to develop them. This is demonstrated in Lst. 2 which finds the `ModelRoles` and their workflows (the `process`) for the `conformanceService`, where the `ModelRoles` are not yet filled.

```
1 SELECT ?modelRole ?process {
2   ?serviceRole enabledBy ?enablerRole.
3   ?enablerRole requires ?modelRole.
4   ?modelRole deployedUsing ?process.
5   FILTER NOT EXISTS { ?serviceRole playedBy ?element }
6   FILTER NOT EXISTS { ?enablerRole playedBy ?element }
7   FILTER NOT EXISTS { ?modelRole playedBy ?element }
8   VALUES ?serviceRole { conformanceService }
9 }
```

Listing 2: A query to find not-yet-developed models and their workflows for the conformance monitor service.

The to-be-created, existing, and already-deployed models in the DT can play these `ModelRoles`. We foresee that this role information could be augmented to better select the required model from a repository (see [15]), or to improve the development workflow. An example would be the addition of validity information for the model [16, 22] to capture the expected validity envelope for the model, or by detailing further fidelity requirements [18].

3.3 Step 3 - Model and Service Workflows

In our approach, we propose the use of enactable workflows to develop and deploy DT services, and the models required to support those services. We select the workflow formalism to explicitly represent and allow customization of the process steps.

Workflow Ontology. Note that while there are numerous approaches for workflow modeling [12] and an equally rich body of work on ontologies in DTs [13], we here introduce a specialized workflow ontology to target our exact needs.

In particular, we express a) the workflow itself, with `Steps` and `Decisions` (decision points), and also b) the workflow's relation between artifacts, such as model instances and data, and the tasks that they produce and consume. For example, each step can either describe a purely mechanical computation (e.g., loading a model instance, simulation, plotting) or a task that requires human involvement (e.g., comparing data). Lst. 3 presents a selection of the workflow ontology, with concepts for these various steps, the artifacts they consume and produce, and the relationships between these. Not shown in Lst. 3 are concepts for traceability, such as recording provenance and usage of data and parameters for steps.

```
1  concept Process,Section,Step,GenericArtifact
2  concept GenDiagram,GenResult,GenModel < GenericArtifact
3  concept Decision,LoadStep,SimulateStep,DeployStep,CompareStep < Step
4  relation entity SectionContent [ from Section to Step
5     forward sectionOf reverse inSection ]
6  relation entity HasStep [ from Process to Step
7     forward hasStep reverse stepOf ]
8  relation entity Production [ from Step to GenericArtifact
9     forward produces reverse producedBy ]
10 relation entity Consumption [ from Step to GenericArtifact
11    forward consumes reverse consumedBy ]
```

Listing 3: Concepts from our OML workflow vocabulary, defining steps, artifacts, and their relationships.

Service Workflow Structure. From our application of this approach to our project DT, we have the intuition that there are common workflow stages to develop and deploy DT services. This is shown in the right-hand side in Figure 4 as involving three stages: 1) first the model is *developed and tested* to satisfy the user's requirements, 2) the model is *deployed* to the running DT platform and tested in the DT environment by connection to incoming data, and 3) the service itself is created through the *connection and deployment* of all the models and enablers.

While these stages are somewhat coarse, our insight is that defining smaller development stages is too rigid for our users. In particular, it is hard to predict exactly how the user's model will evolve over time, as there is expected back-and-forth experimentation when developing the models and the service. In our project, we have iterated closely with the practitioners to define the workflows at the right level of detail, as presented in Section 4.2.

3.4 Tooling for Enactment and Deployment

Our approach requires tooling to enact the workflow(s) and guides the user along it step-by-step to provide model configuration and management assistance, as well as visualization functionality as appropriate. We also foresee extensive modeling guidance [2] as essential to assist our non-software engineering experts. Currently, we are developing our approach tooling (Section 4.1) to query and modify the knowledge graphs to handle model loading and simulation, record model parameters and experiments, and be able to deploy the models and service to the DT deployment platform.

4 APPROACH APPLICATION

This section describes a preliminary application of our approach from Section 3 to the DIGIT-BENCH DT from Section 2, where the user has selected the 'conformance monitor' service from the service menu. In particular, we describe our prototype tooling and

the intention behind it. We also present below workflows for: a) decomposing the model in Figure 3 into sub-models that can be individually wrapped into FMUs for deployment to the DT, and b) developing and deploying the DUTMonitor service (recall Figure 2).

4.1 Prototype Tooling

Our prototype tooling is built on Jupyter notebooks[9] running Python. This allows us to serve these notebooks on a web server running JupyterLab on the project premises, and to allow for non-interactive modes to utilize CI/CD pipelines. We are thus targeting users who have a basic understanding of command line tools, scripting in Python, and the use of Jupyter notebooks. Our goal is to automate tasks so that the users, who understand the PT at a deep level, can focus on the core engineering steps of modeling, simulation, and identifying discrepancies in model and service results.

In particular, if a step depends on models, simulation results, configurations, etc., from any previous steps, then inconsistencies must be detected by the tool and reported to the user. For example, a user shall not be able to run a step with such dependencies if the previous steps have not been run before, or if their outputs have become out of date. Our tools do not automatically detect discrepancies in the results between different steps as this is domain-dependent. It must also be possible for the user to inspect these results and implement automatic detection of discrepancies.

Our ongoing work is on generating the Jupyter notebooks with a detailed skeleton of each step in the workflow. The user is expected to modify these with the actual code for running the experiments, as this is application-specific. Our tooling provides generated code in the notebooks that communicates with the knowledge graph, such that the knowledge graph can be checked for consistency before/after executing each experiment.

4.2 Development Workflows

Workflow 1: Testbed Model Development Workflow. This workflow develops the testbed model for the conformance monitoring service, corresponding to the *model development/deployment/testing* stages on the right-hand side of Figure 4. The model is a monolithic simulation model in OpenModelica, which must be partitioned into a co-simulation, with FMUs to be deployed for all sub-models. During partitioning, each sub-model must be individually verified, along with their coupled form in a co-simulation. Each step in this workflow is designed to quantify one source of error, as errors may arise from partitioning or the FMU coming from different suppliers. Each step also includes an implicit plotting task for visual inspection.

- **Step 1** Load/build the monolithic model into a model instance, simulate a test scenario, and store the simulation results.
- **Step 2** Load a model that partitions the monolith with an explicit coupling mechanism between the parts of the turbine testing bench. Simulate it, and compare it with the original model to assess whether the loss of accuracy is acceptable. This step ensures that the simulation error introduced by the coupling can be quantified.
- **Step 3** Load a model that refines the model from step 2 into a hierarchical model where the parts of the turbine testing bench are decoupled. Simulate it, and compare it with the original model to assess whether the decoupling has introduced any loss of precision.
- **Step 4** Export the decoupled models as FMU's, co-simulate them, and compare the results with the previous step. This step quantifies the error introduced by the co-simulation orchestration algorithm and its configuration.
- **Step 5** The FMUs can now be swapped by FMUs exported from multiple modeling and simulation tools. This step quantifies any discrepancy introduced by producing the FMUs in different tools.

Service Deployment Workflow. This workflow involves the development and deployment of the conformance monitor service, corresponding to the *service deployment and testing* stage on the right-hand side of Figure 4.

- **Step 1** Run a co-simulation using the DUTMonitor as an FMU connected to the simulations of the rest of the system as FMUS. The co-simulation scenario is exactly like Figure 2 except all boxes (except DUT&TB Coupled Model) are FMUs. The monitor uses the models produced in the previous steps and provides the results as an FMU via, e.g., hierarchical co-simulation [8]. A fault is introduced in the testbench or DUT model to test the monitor, such as a change in the testbed coupling component stiffness (seen in the lower-middle of Figure 3). The monitor output should clearly highlight this fault. This step tests the functionality of the monitor.
- **Step 2** The DUTMonitor is packaged as a DT service, using a library to communicate with other services via RabbitMQ. This is the monitor interface for production. The simulation results of the previous step are streamed to the service, and the resulting stream of events is stored and compared with the results of the previous step. This step tests the service interface with the rest of the system.
- **Step 3** This step is similar to the previous step except the stream of data now comes in real time at the same rate as the sensor data coming from the PT. This step focuses on testing the real-time capabilities of the service.
- **Step 4** In this step the service is deployed in the production system but its inputs are connected to a PT service which is just a real time simulation of the PT. This step focuses on testing the service integration with the rest of the system. The results are compared with the previous steps.
- **Step 5 (Optional for Digital Shadow services)** This step is similar to the previous step except now the inputs to the service are from the real PT. The outputs remain connected to the simulated PT. This step focuses on testing the service integration with the real PT and assessing whether its outputs match the results from the previous steps.
- **Step 6** The service is now connected to the real PT. This step deploys the service in the production system.

5 RELATED WORK

The recent paper of Carrión and Pastor surveys DT creation methodologies [1]. We note that these methodologies often start at the data or interface level, instead of the service-oriented approach we propose here. For example, Kirchhof *et al.* focus on modeling components and interfaces and then generating code for them [14].

[9] https://jupyter.org/

They correctly identify the challenge and importance of generating and maintaining correct software interfaces for the DT services to communicate. Our intention is to provide a methodology that can, starting from the DT service itself, guide the user to develop or find the necessary enablers and/or models for that service. In [17], the authors propose the use of UML and OCL for the generation of digital twins. This work includes the notion of DT services as components in the DT architecture. However, it does not propose following customized workflows or recommending specific enablers and models depending on each service as we do.

For ontologies and workflows, Mittal *et al.* [16] propose explicit model management workflows combined with ontologies to capture the processes and boundaries of *model validity* in simulation [22]. Earlier work [20] summarizes the development workflow for two DT case studies, and together with [16] serves as inspiration for our own work. However, these works do not discuss the engineering of DT *services* using ontologies. Taking this focus allows us to consider customized workflows for each particular service, integrating multi-domain knowledge and allowing for consistency checking.

6 CONCLUSION

This paper has presented our on-going work on developing an ontologically-based approach to engineering DT services. The approach starts from the user selecting a service. Then, based on ontologies and knowledge graphs, the user is guided through the development and deployment of the service to the DT. The expected impact of our approach is that a domain expert will be able to use tools conforming to this approach to more efficiently build DT services, as measured by time, effort, and usability metrics.

Our approach has been very welcomed by the industrial practitioners, who see value in this service-driven, workflow-based approach. They appreciate that the workflows and Jupyter notebooks promote best practices for reproducibility and automation in the experiments. However, there are challenges in balancing rigid workflow steps with the flexibility required by the practitioners.

Our future work is to further develop our approach, the underlying ontologies, and our prototype tooling. In particular, we are interested in collecting multiple DT exemplars reported in the DT description framework of Gil *et al.* [7]. This information, along with the insights from our existing DT case studies, will be used to enrich the enabler/model recommendations and development workflows stored in our knowledge graph. Another open question is the connection of the lifecycles of the different ontologies and our framework. Using it requires to harmonize the ontologies it is using with each other, as well as integrate this harmonization into the lifecycle of the tooling.

ACKNOWLEDGMENTS

Partly funded by the EU project SM4RTENANCE (grant no. 101123423) and by the EUDP DIGIT-BENCH project (grant no. 640222-497272).

REFERENCES

[1] Emilio Carrión and Óscar Pastor. 2023. A systematic review of methodologies for developing Digital Twins: Insights and recommendations for effective implementation. (2023).

[2] Shalini Chakraborty and Grischa Liebel. 2024. Modelling guidance in software engineering: a systematic literature review. *Software and Systems Modeling* 23, 1 (2024), 249–265.

[3] Manuela Dalibor, Nico Jansen, Bernhard Rumpe, David Schmalzing, Louis Wachtmeister, Manuel Wimmer, and Andreas Wortmann. 2022. A cross-domain systematic mapping study on software engineering for digital twins. *Journal of Systems and Software* 193 (2022), 111361.

[4] Maged Elaasar, Nicolas Rouquette, David Wagner, Bentley Oakes, Abdelwahab Hamou-Lhadj, and Mohammad Hamdaqa. 2023. openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering. *Inter. Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (2023).

[5] Peter Fritzson, Peter Aronsson, Adrian Pop, Hakan Lundvall, Kaj Nystrom, Levon Saldamli, David Broman, and Anders Sandholm. 2006. OpenModelica-A free open-source environment for system modeling, simulation, and teaching. In *IEEE International Conference on Computer-Aided Design*. IEEE, 1588–1595.

[6] Santiago Gil, Peter H Mikkelsen, Cláudio Gomes, and Peter G Larsen. 2024. Survey on open-source digital twin frameworks–A case study approach. *Software: Practice and Experience* (2024).

[7] Santiago Gil, Bentley Oakes, Claudio Gomes, Mirgita Frasheri, and Peter G. Larsen. 2024. Towards a Systematic Reporting Framework for Digital Twins: A Cooperative Robotics Case Study. *SIMULATION* (2024), 1–27. https://doi.org/10.1177/00375497241261406

[8] Cláudio Gomes, Bart Meyers, Joachim Denil, Casper Thule, Kenneth Lausdahl, Hans Vangheluwe, and Paul De Meulenaere. 2018. Semantic adaptation for FMI co-simulation with hierarchical simulators. *SIMULATION* 95, 3 (April 2018), 241–269. https://doi.org/10.1177/0037549718759775

[9] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. Co-Simulation: A Survey. *Comput. Surveys* 51, 3 (May 2018), 1–33. https://doi.org/10.1145/3179993

[10] Michael Grieves and John Vickers. 2017. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. *Transdisciplinary perspectives on complex systems: New findings and approaches* (2017), 85–113.

[11] Simon Thrane Hansen, Casper Thule, Cláudio Gomes, Kenneth Guldbrandt Lausdahl, Frederik Palludan Madsen, Giuseppe Abbiati, and Peter Gorm Larsen. 2024. Co-simulation at different levels of expertise with Maestro2. *Journal of Systems and Software* 209 (March 2024), 111905.

[12] Andreas Harth, Tobias Käfer, Anisa Rula, Jean-Paul Calbimonte, Eduard Kamburjan, and Martin Giese. 2024. Towards Representing Processes and Reasoning With Process Descriptions on the Web. *Transactions on Graph Data and Knowledge* 2 (2024). Issue 1.

[13] Erkan Karabulut, Salvatore F Pileggi, Paul Groth, and Victoria Degeler. 2023. Ontologies in Digital Twins: A Systematic Literature Review. *Future Generation Computer Systems* 153 (2023).

[14] Jörg Christian Kirchhof, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. 2020. Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In *Inter. Conference on Model-driven Engineering Languages and Systems*. 90–101.

[15] Daniel Lehner, Sabine Wolny, Alexandra Mazak-Huemer, and Manuel Wimmer. 2020. Towards a reference architecture for leveraging model repositories for digital twins. In *2020 25th IEEE international conference on emerging technologies and factory automation (ETFA)*, Vol. 1. IEEE, 1077–1080.

[16] Rakshit Mittal, Raheleh Eslampanah, Lucas Lima, Hans Vangheluwe, and Dominique Blouin. 2023. Towards an Ontological Framework for Validity Frames. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 801–805.

[17] Paula Muñoz, Javier Troya, and Antonio Vallecillo. 2021. Using UML and OCL models to realize high-level digital twins. In *Inter. Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 212–220.

[18] Bentley Oakes, Claudio Gomes, Joachim Denil, Julien Deantoni, Joao Cambeiro, John Fitzgerald, and Peter Gorm Larsen. 2023. Examining Model Qualities and Their Impact on Digital Twins. In *Annual Modeling and Simulation Conference (ANNSIM)*. IEEE, 220–232.

[19] Bentley Oakes, Bart Meyers, Dennis Janssens, and Hans Vangheluwe. 2021. Structuring and Accessing Knowledge for Historical and Streaming Digital Twins. In *First Workshop on Ontology-Driven Conceptual Modeling of Digital Twins*. 1–13.

[20] Randy Paredis, Cláudio Gomes, and Hans Vangheluwe. 2021. Towards a Family of Digital Model/Shadow/Twin Workflows and Architectures.. In *IN4PL*. 174–182.

[21] Fei Tao, Meng Zhang, and AYC Nee. 2019. Five-dimension digital twin modeling and its key technologies. *Digital Twin Driven Smart Manufacturing* (2019), 63–81.

[22] Simon Van Mierlo, Bentley James Oakes, Bert Van Acker, Raheleh Eslampanah, Joachim Denil, and Hans Vangheluwe. 2020. Exploring Validity Frames in Practice. In *Proceedings of the First International Conference, ICSMM 2020, Bergen, Norway, June 25–26, 2020*. Springer, Cham, 131–148.

[23] Andreas Wortmann. 2024. Digital Twin Definitions. https://awortmann.github.io/research/digital_twin_definitions/. Accessed on March 3, 2024.