Author pre-print. Publication accepted for EDTconf 2025.

DTInsight: A Tool for Explicit, Interactive, and Continuous Digital Twin Reporting

Kérian Fiter
Dept. of Computer and Software Eng.
Polytechnique Montréal
Montréal, Canada
kerian.fiter@polymtl.ca

Louis Malassigné-Onfroy
École d'Ingénieurs du Conservatoire National des Arts et Métiers (EICNAM)
Paris, France
louis.malassigne@gmail.com

Bentley Oakes
Dept. of Computer and Software Eng.
Polytechnique Montréal
Montréal, Canada
bentley.oakes@polymtl.ca

***Abstract*—With Digital Twin (DT) construction and evolution occurring over time, stakeholders require tools to understand the current characteristics and conceptual architecture of the system at any time. We introduce DTInsight, a systematic and automated tool and methodology for producing continuous reporting for DTs. DTInsight offers three key features: (a) an interactive conceptual architecture visualization of DTs; (b) generation of summaries of DT characteristics based on ontological data; and (c) integration of these outputs into a reporting page within a continuous integration and continuous deployment (CI/CD) pipeline. Given a modeled description of the DT aligning to our DT Description Framework (DTDF), DTInsight enables up-to-date and detailed reports for enhanced stakeholder understanding.**

***Index Terms*—digital twins, software visualization, software documentation, decision-making, ontologies, OML, monitoring**

I. Introduction

Digital Twins (DTs) are digital representations of physical systems, enabling monitoring, insight generation, and control of their physical counterparts [1]. DT engineering is becoming increasingly systematized through model-based approaches [2]. However, previous work has shown that the *reporting* of DT capabilities often leaves out important details [3], [4]. We argue this lack of systematic reporting hinders research into DT capabilities and engineering process.

In particular, the motivation behind DTInsight is to apply TwinOps [5] to DT engineering. That is, the key problem to address is: *as the DT changes, how can reporting be kept up-to-date such that stakeholders (including non-technical personnel such as management) are always aware of the DT's current capabilities, architecture, and state*. We aim to improve users' understanding of their DT and enhance their decision-making capabilities regarding the DT's evolution.

Another aspect of this tool is to *assist practitioners in reporting their DT* [3]. This approach works towards the vision of DTs having *characteristic cards* unifying their digital and physical element reporting, similar to machine learning model cards [6], [7] or a *Digital Twin Bill of Materials* (DT BOM).

Our research treats DTs as a complex software system, therefore we bring in concepts from fields related to software engineering and adopt principles of *observability* [8], to have a deeper understanding of the DT.

This means working towards giving the user the ability to investigate and understand the DT's current structural and behavioral state. For example, our architectural view promotes understanding of the data flow between DT components. We argue that this approach enables practitioners to gain deeper insights into service-driven DTs, to better support their construction, evolution, and reporting.

In this article, we *present the DTInsight tool and methodology, for continuously reporting a DT's conceptual architecture, its capabilities, and a selection of its behavior*. Our tool's source code is available at [9], with [10] containing the reporting framework ontology and our example modeled DT.

Our DTInsight approach has three pillars, as in Figure 1:

Pillar 1) *Explicit reporting*, of the DT in the DTDF (Section II): Explicit DT reporting is achieved by modeling the DT using the 21 characteristics from the DT Description Framework (DTDF) [11], in the Ontology Modeling Language (OML). We use openCAESAR [12], an ontology development framework from NASA Jet Propulsion Laboratory (JPL). These DTDF characteristics are formalized in an OML *vocabulary*, providing concepts for DT *description models*.

Pillar 2) *Interactive reporting*, through a game engine and system monitoring (Section III): The interactive reporting uses description models conforming with the DTDF vocabulary to generate an interactive conceptual architecture of our DT, in a visualization called a *DT constellation* [3]. Using the Godot Engine [13], we can represent the flow of data from the system's sensors with the appropriate communication libraries.

Pillar 3) *Continuous reporting*, through integration into CI/CD (Section IV): Finally, this visualization is integrated in a GitHub Actions CI/CD pipeline to create an interactive and up-to-date reporting page of the DT. This reporting page uses the description model of the DT to generate a conceptual architecture diagram, a summary table of the reporting characteristics, and an embedding of the interactive visualization.

II. Explicit DT Description Framework Ontology

Previous work has found that DTs experience reports often lack crucial information for comparing and understanding the capabilities of said DT [3], [14]. Therefore, we advocate for DTs' *explicit reporting* through a systematic and consistent reporting framework composed of 21 characteristics

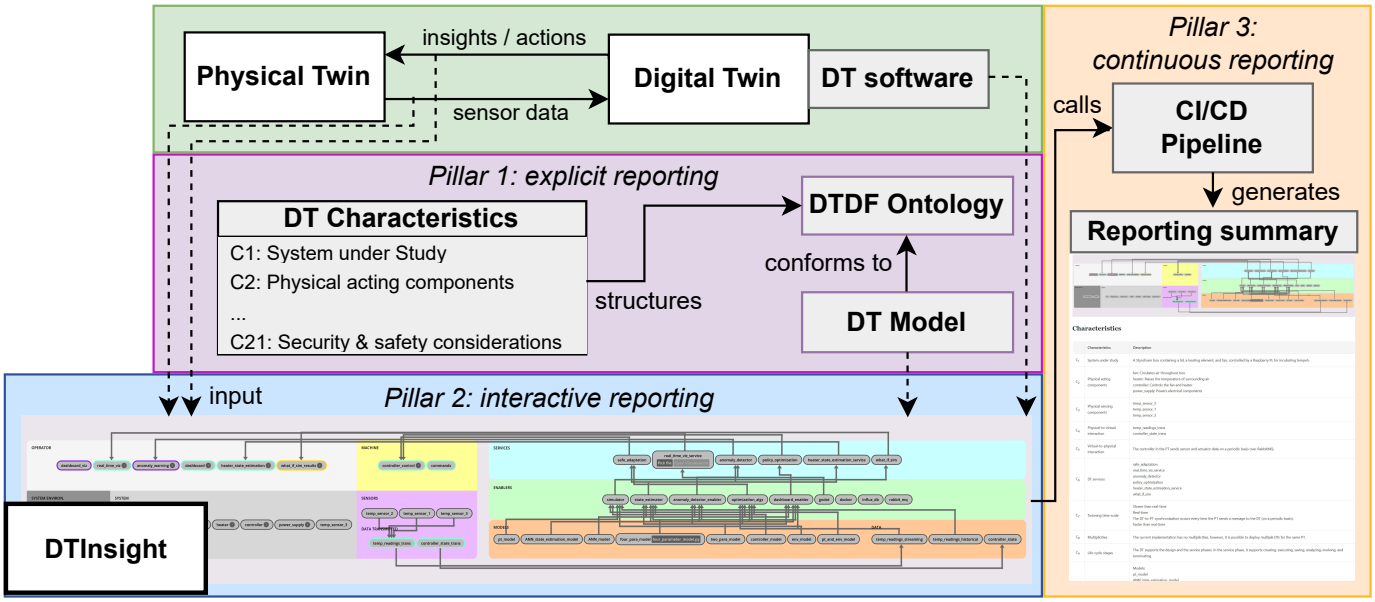

Fig. 1. DTInsight monitors the communication between Physical and Digital Twin, reads the modeled DT, and generates a report.

to describe DTs [11]. These reporting characteristics range from the twinning time-scale (C7), to fidelity and validity considerations (C14), technical implementation (C15), and security and safety considerations (C21).

We formalize this *DT Description Framework* (DTDF) as an ontology, available at [10]. An ontology is "an explicit specification of a conceptualization" [15]. It defines classes, relations, functions, and axioms that make shared meaning machine-readable. For increased agility and rigor, we model the DTDF using OML in the OML Rosetta editor [12]. OML acts as a Domain Specific Language (DSL) layer above the well-known Web Ontology Language (OWL) to simplify ontology creation. It represents ontologies as *vocabularies* (containing concepts, similar to a meta-model) and *descriptions* (containing instances of those concepts). The DTDF vocabulary thus formalizes the 21 DT reporting characteristics and their ontological relationships.

Then, in an OML *description model*, we model the incubator DT example [11], [16]. The incubator's purpose is to maintain the temperature in its enclosure through the control of a heater and the monitoring of temperature sensors. The incubator DT can then visualize the incubator's behavior, optimize the control policy, etc. The incubator DT is proposed as a case for DT engineering as it has complex behavior in multiple domains (electrical, thermal, mechanical) yet is simple enough for pedagogical purposes [17].

The DTDF vocabulary (excerpted in Listing 1) adopts a three-layered, service-oriented conceptual architecture for describing DTs. *Models* and *Data* are inputs to computational components called *Enablers*, which process the models and data to enable *Services* providing the DT's actions or insights.

The DTDF description imports the DTDFVocab to *instantiate* its defined concepts. This allows each DT use case to reuse the vocabulary by creating instances tailored to its specific conceptual architecture. Listing 2 illustrates this approach through an example describing the incubator using the DTDF.

```
// C6: Services
concept Service < DTComponent, TimeScaleThing
relation entity Provides [from Service to
    ProvidedThing forward provides reverse
    providedBy]
// C11: Enablers
concept Enabler < DTComponent
relation entity Enables [from Enabler to Service
    forward enables reverse enabledBy]
scalar property IsCommEnabler [domain Enabler range
    xsd:boolean functional]
// C10: Models/Data
aspect Input
concept Model < DTComponent, Input
concept Data < DTComponent, Input
relation entity InputTo [from Input to Enabler
    forward inputTo reverse hasInput]
relation entity DataInput [from DataTransmitted to
    Data forward asData reverse fromData]
// C20: Standardization
concept Standardization < base:DescribedThing
```

Listing 1. DTDF ontology vocabulary model (DTDFVocab) excerpt in OML.

III. Interactive DTInsight Tool

This section describes how DTInsight loads the user's DT as described in the DTDF, and generates an interactive visualization. The objective is to unify the structural and behavioral descriptions of the DT and extend the user's reporting view to incorporate both reporting *and* behavioral insights.

a) Technical Details: DTInsight is built in the Godot Engine [13], an open-source game engine with active development and community. It is well-suited to our needs as it a) is MIT-licensed, b) supports both 2D and 3D, with advanced

```
// SERVICE EXAMPLE (C6)
instance what_if_sim : DTDFVocab:Service
    [DTDFVocab:provides what_if_sim_results
    DTDFVocab:atStage baseDesc:operation]
// ENABLER EXAMPLE (C11)
instance simulator : DTDFVocab:Enabler
    [DTDFVocab:enables what_if_sim]
// MODEL EXAMPLE (C10)
instance controller_model : DTDFVocab:Model
    [DTDFVocab:inputTo simulator DTDFVocab:inputTo
    state_estimator DTDFVocab:inputTo
    optimization_algs]
// DESCRIBED CHARACTERISTIC EXAMPLE (C20)
instance standardization :
    DTDFVocab:Standardization [base:desc
    "Communication is carried out using AMQP
    standard via RabbitMQ. Behavioral models have
    been produced following the FMI standard
    version 2."]
```

Listing 2. DTDF ontology incubator description model excerpt in OML.

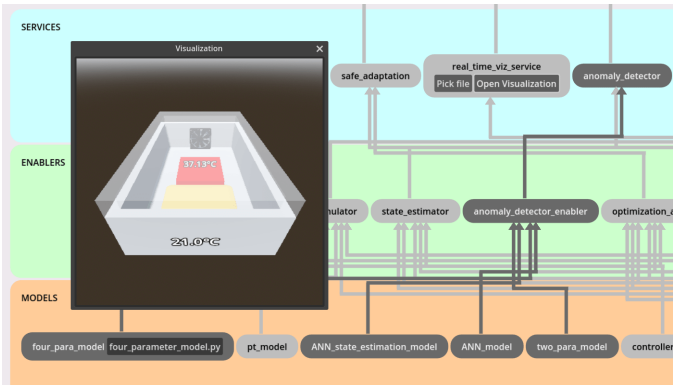

Fig. 2. DT constellation interactivity: highlight DT component dependencies and display an up-to-date 3D visualization of the system

UI creation capabilities, c) exports to web and all desktop platforms, and d) focuses on ease-of-use. By using the .NET version of Godot, we can also subscribe to RabbitMQ [18] message broker queues through available C# libraries, enabling us to capture data flowing from the real or simulated incubator system and display it in the visualization.

The DTDF ontology is served by the Apache Jena Fuseki server in Rosetta [12]. We use *queries* to retrieve the details of the DT instance described in DTDF via SPARQL, which is a standard query language for retrieving and manipulating data stored in Resource Description Framework (RDF) format. Users interact with the Fuseki server by sending HTTP GET requests to its endpoint, allowing them to retrieve and explore DTDF ontological data represented in RDF. This functionality facilitates the querying of complex datasets and the extraction of specific relationships and entities, enabling efficient data retrieval within semantic web applications.

From these query results, DTInsight creates a DT visualization termed a *constellation*, following the approach suggested in [3]. A *DT constellation* is a conceptual architecture that represents the flow of data between Physical Twin (PT), DT, and within the DT. The DT is thus viewed "as an agglomeration of all related models, data, enablers, and [services] that are used in the DT activities" [3].

Note that while other ontology visualizations exist [19], these often use conventional class- or graph-based visualization methods which are not suited for DTs and their rich semantic structure. Instead, we see that this constellation view can improve stakeholder collaboration in DT engineering [20].

We make the DT constellation *interactive*, to unlock the filtering and presenting of relevant and detailed information to the user [21], and *behavioral*, by reflecting the incoming data flow of sensor information. These properties motivate the choice of Godot to build a dynamic visualization.

b) User Interaction with DTInsight: DTInsight offers users various features to gain insights into the system. Figure 1 (DTInsight) illustrates the DT system's composition, structured into two primary sections. The left side represents the five PT components: *Operator*, *Machine*, *System Environment*, *System*, and *Sensors/Data Transmission*. The right side (from bottom-to-top) illustrates the three capability categories of the DT: *Models/Data*, *Enablers*, and *Services*. One-directional arrows connect the components, indicating their inter-dependencies and data flow relationships.

Figure 2 shows DT constellation *interactivity*: hovering the mouse or clicking over DT components highlights connected components and their data flows. This provides the user with a *structural* understanding of the DT by allowing them to easily explore its components as they are formally described in ontologies. It can help them find components that depend on each other, and navigate backward or forward in the layered and service-based DT conceptual architecture.

Additional Interaction: If the user has connected the software folder containing the DT code, they can also click on a component to see its associated script. If they have connected RabbitMQ, they can click on a sensor component to see a graph of its real-time data.

And finally, the user can pick an external visualization file (a Godot resource pack) to open a 3D visualization of the DT in a pop-up that reflects incoming data from sensors. In the example incubator visualization presented in Figure 2, data labels are updated to reflect the temperature in real-time and the heater color turns red when heating. This strengthens the structural and behavioral descriptions, while keeping the 3D visualization decoupled from the conceptual architecture by being packed into an external file.

IV. Continuous Report Generation

With DTInsight, we are targeting both DT experts and non-technical personnel who benefit from *continuous reporting* to stay updated on the evolution of the DT as it is being developed. From a CI/CD pipeline, currently implemented using GitHub Actions, we can automatically generate a characteristics table and visualizations[1] from every commit to the repo containing the DT code and description model.

[1]An example page for the incubator DT is found here: https://oakeslabmtl.github.io/DTDF/

Thus, the reporting page deployment takes three steps:

1) Modeling the DT in the DTDF (in OML/OWL/RDF)
2) Setting up the deployment of the static website
3) Running the CI/CD workflow to create the report page

The first step is that the user updates the description model of their DT or system using an ontology editing tool such as openCAESAR Rosetta, which is then committed to a Git repository. Upon committing, the reporting pipeline loads the ontology in a Fuseki server and runs the DTInsight tool in a Linux xvfb display server. We trigger its query in Godot via an HTTP request. It outputs the HTML characteristics table, the screenshot of the conceptual architecture, and the YAML file describing it. The web-export of DTInsight is embedded as an iframe into the reporting page, which is then deployed on the web as a static website using Hugo [22].

Thus, as the underlying ontology of the DT evolves, the reporting page reflecting the DT characteristics and conceptual architecture is automatically updated to mirror these changes.

V. Related Work

a) DT Architecture: Dalibor *et al.* employ DSLs to generate interactive DT cockpits that integrate both the internal DT infrastructure and a monitoring frontend [23]. A similar low-code approach is taken by De Sanctis *et al.* in the context of smart cities [24]. However, they do not adopt the ontological and service-oriented approach resulting in the DT constellation view [25], nor do they report infrastructure evolution within a continuous CI/CD pipeline. Carrion *et al.* recently proposed a conceptual Entity-Relationship Digital Twin (ERDT) model to represent the PT, DT, and their interconnections [26]. Information retrieval within this model is facilitated through DSL-defined queries [27], [28]. In contrast, our approach leverages an ontological framework that enables formal semantics, logical inference, and automated reasoning, using SPARQL for queries. Our work resembles Software Architecture Reconstruction (SAR) explored by Ducasse *et al.* [29], but applied to DTs. Unlike traditional *reverse architecting* from source code, we perform *manual SAR* by leveraging user-defined ontologies to construct a *conceptual and architectural view*, utilizing the DT constellation's *reporting style*. Ozkaya *et al.* proposed a modeling language for DT architecture using C4 (Context, Containers, Components, Code) [30]. In contrast, our work focuses on reporting DT architecture components in the DTDF, using ontologies as ground truth.

b) DT Reporting: The lack of adequate documentation in DT systems and the frequent divergence between actual and original design is highlighted by Gunasekaran *et al.* [4]. To address this, their work leverages logging to analyze the behavior of evolving DT systems. In contrast, our approach focuses on simplifying the generation of continuous software documentation through DTDF reporting driven by expert knowledge. Yahouni *et al.* propose a reporting system for decision-making actors during manufacturing. It uses a Multi-Agent System (MAS) to gather user needs, extract data, calculate Key Performance Indicators (KPIs), and generate reports [31]. In contrast, our approach is centered on generating *conceptual* DT reports, focusing on their architecture and characteristics. Uhlenkamp *et al.* propose a DT maturity assessment tool[2], allowing the user to assign values to their assessment categories through a web-based form and obtain a maturity score [32]. By comparison, we focus on the conceptual DT architecture, describing DTs using free-form text for most characteristics.

c) Software Visualization: Langelier *et al.* used visualization to analyze large scale-systems [33]. Similarly, Cerny *et al.* explored microservice architecture visualization techniques for static and dynamic SAR [34], [35]. Meanwhile, Antoniazzi *et al.* focused on RDF graph visualization [36]. We bring those works into SAR for (micro)service-based DTs using the DTDF ontology, structuring its RDF graph as a DT constellation visualization and making it behavioral by integrating data flow.

VI. Conclusion

DTInsight's primary benefit is enhancing communication with stakeholders, which is achieved through a threefold contribution to DT reporting: making it *explicit* through the DTDF ontology, *interactive* via the structural and behavioral conceptual architecture, and *continuous* through reporting page generation. This provides a domain-specific view on DTs, building on prior work on a DT reporting framework [3], [11]. Furthermore, the integration of scripts directly into the DT constellation and the inclusion of real-time data in graphs and 3D visualizations promote a practice of interactive monitoring.

Limitations: However, there are some limitations to consider. The system presents a high-level conceptual visualization of the DT architecture, but it requires modeling by a DT expert that has an overarching understanding of the DT. Moreover, the visualization itself could be made more readable and offer a denser representation of information. Going deeper into examining each component of the DT would also promote observability [8]. On the technical side, the system currently supports only one message broker (RabbitMQ), which limits its flexibility. Additionally, the reliance on manually written ontologies is a limitation, though this could be improved by using techniques like Large Language Models (LLMs) for automatic ontology generation, or by incorporating visual interfaces to offer a creation GUI. Another limitation is that the current tool only supports the representation of a single DT at a time, but this could be expanded to represent systems-of-systems [37]. Furthermore, the system does not support real-time changes to represent self-reconfiguring DTs [38]. This aligns with research on human/DT interaction with intelligent and adaptive interfaces capable of evolving in real time and adapting to both the operator's context and needs [39].

Future work: We are currently pursuing editing the description model directly from a visual interface, assisted by integrated LLMs. We are also looking at exposing more information about each component in the DT, such as its current lifecycle stage, operational state, and visualizing simulation results over time.

[2]https://dt-maturity.eu/

References

[1] A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging digital twin technology in model-based systems engineering," *Systems*, vol. 7, no. 1, p. 7, 2019.

[2] J. Michael, L. Cleophas, S. Zschaler, T. Clark, B. Combemale, T. Godfrey, D. E. Khelladi, V. Kulkarni, D. Lehner, B. Rumpe, M. Wimmer, A. Wortmann, S. Ali, B. Barn, I. Barosan, N. Bencomo, F. Bordeleau, G. Grossmann, G. Karsai, O. Kopp, B. Mitschang, P. Muñoz Ariza, A. Pierantonio, F. A. C. Polack, M. Riebisch, H. Schlingloff, M. Stumptner, A. Vallecillo, M. van den Brand, and H. Vangheluwe, "Model-driven engineering for digital twins: Opportunities and challenges," *Systems Engineering*, p. sys.21815, Apr. 2025.

[3] B. Oakes, A. Parsai, S. V. Mierlo, S. Demeyer, J. Denil, P. D. Meulenaere, and H. Vangheluwe, "Improving digital twin experience reports," in *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2021, pp. 179–190.

[4] R. Gunasekaran, B. Haverkort, and L. Kruger, "Behavioral analysis of a digital twin using logging and model learning," *Journal of Object Technology*, vol. 24, no. 2, pp. 2:1–14, May 2025, the 21st European Conference on Modelling Foundations and Applications (ECMFA 2025). [Online]. Available: http://www.jot.fm/contents/issue_2025_02/a7.html

[5] J. Hugues, A. Hristosov, J. J. Hudak, and J. Yankel, "TwinOps - DevOps meets model-based engineering and digital twins for the engineering of CPS," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Virtual Event Canada: ACM, Oct. 2020, pp. 1–5.

[6] E. Ozoani, M. Gerchick, and M. Mitchell, "Model card guidebook," 2022. [Online]. Available: https://huggingface.co/docs/hub/en/model-card-guidebook

[7] T. R. Toma, B. Grewal, and C.-P. Bezemer, "Answering user questions about machine learning models through standardized model cards," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1488–1500.

[8] C. Majors, L. Fong-Jones, and G. Miranda, *Observability Engineering: Achieving Production Excellence*. Sebastopol, CA: O'Reilly Media, 2022.

[9] K. Fiter, L. Malassigné-Onfroy, and B. Oakes, "DTInsight," https://github.com/oakeslabmtl/DTInsight, 2025.

[10] ——, "DTDF ontology," https://github.com/oakeslabmtl/DTDF, 2025.

[11] S. Gil, B. Oakes, C. Gomes, M. Frasheri, and P. G. Larsen, "Toward a systematic reporting framework for Digital Twins: A cooperative robotics case study," *SIMULATION*, vol. 101, no. 3, pp. 313–339, Aug. 2024.

[12] M. Elaasar, N. Rouquette, D. Wagner, B. Oakes, A. Hamou-Lhadj, and M. Hamdaqa, "openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Västerås, Sweden: IEEE, Oct. 2023, pp. 221–230.

[13] Godot Engine contributors, "Godot engine," https://godotengine.org/, 2025, version 4.4.1.

[14] B. Oakes, A. Parsai, B. Meyers, I. David, S. Van Mierlo, S. Demeyer, J. Denil, P. De Meulenaere, and H. Vangheluwe, "A digital twin description framework and its mapping to Asset Administration Shell," in *MODELSWARD, Communications in Computer and Information Science*, vol. 1708. Springer, Aug. 2023, pp. 1–24.

[15] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[16] H. Feng, C. Gomes, C. Thule, K. Lausdahl, M. Sandberg, and P. G. Larsen, "The incubator case study for digital twin engineering," *arXiv preprint arXiv:2102.10390*, 2021.

[17] C. Gomes, M. H. Kristensen, M. S. Andersen, P. Talasila, H. Feng, T. Wright, and P. G. Larsen, "Digital twin tutorial: The incubator case study," in *Engineering Trustworthy Software Systems: 6th International School, SETSS 2024, Chongqing, China, April 14–21, 2024, Tutorial Lectures*. Springer, 2025, pp. 68–101, code for the incubator DT is found at https://github.com/INTO-CPS-Association/example_digital-twin_incubator.

[18] RabbitMQ, "RabbitMQ: Open source message broker," 2025, accessed: 2025-06-01. [Online]. Available: https://www.rabbitmq.com/

[19] M. Dudáš, S. Lohmann, V. Svátek, and D. Pavlov, "Ontology visualization methods and tools: a survey of the state of the art," *The Knowledge Engineering Review*, vol. 33, p. e10, 2018.

[20] P.-E. Goffi, R. Tremblay, and B. Oakes, "Engineering a digital twin for the monitoring and control of beer fermentation sampling," in *Proceedings of the 28th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2025.

[21] A. Štěpánek, D. Kuťák, B. Kozlíková, and J. Byška, "Interactive diagrams for software documentation," in *2024 IEEE Working Conference on Software Visualization (VISSOFT)*, Oct. 2024, pp. 12–23.

[22] Hugo Authors, "Hugo: The world's fastest framework for building websites," 2025. [Online]. Available: https://gohugo.io/

[23] M. Dalibor, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Towards a model-driven architecture for interactive digital twin cockpits," in *Conceptual Modeling*, G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, and H. C. Mayr, Eds. Cham: Springer International Publishing, 2020, vol. 12400, pp. 377–387.

[24] M. De Sanctis, L. Iovino, M. T. Rossi, and M. Wimmer, "A low-code assessment platform for urban digital twins," *Information and Software Technology*, vol. 183, p. 107726, Jul. 2025.

[25] B. Oakes, C. Gomes, E. Kamburjan, G. Abbiati, E. Ecem Bas, and S. Engelsgaard, "Towards ontological service-driven engineering of digital twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. Linz Austria: ACM, Sep. 2024, pp. 464–469.

[26] E. Carrión, Ó. Pastor, and P. Valderas, "Conceptual modelling method for digital twins," in *Conceptual Modeling*, W. Maass, H. Han, H. Yasar, and N. Multari, Eds. Cham: Springer Nature Switzerland, 2025, pp. 417–435.

[27] E. Carrión and P. Valderas, "Implementing digital twin query views," in *Research Challenges in Information Science*, J. Grabis, T. E. J. Vos, M. J. Escalona, and O. Pastor, Eds. Cham: Springer Nature Switzerland, 2025, pp. 279–294.

[28] E. Carrión, P. Valderas, and Ó. Pastor, "Querying Digital Twin Models:," in *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering*. Portugal: SCITEPRESS - Science and Technology Publications, 2025, pp. 398–405.

[29] S. Ducasse and D. Pollet, "Software Architecture Reconstruction: A Process-Oriented Taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, Jul. 2009.

[30] M. Ozkaya, "Towards an architecture modeling language for specifying digital twin architectures using c4," in *New Trends in Intelligent Software Methodologies, Tools and Techniques*. IOS Press, 2024, pp. 151–164.

[31] Z. Yahouni, A. Ladj, F. Belkadi, O. Meski, and M. Ritou, "A smart reporting framework as an application of multi-agent system in machining industry," *International Journal of Computer Integrated Manufacturing*, vol. 34, no. 5, pp. 470–486, May 2021.

[32] J.-F. Uhlenkamp, J. B. Hauge, E. Broda, M. Lütjen, M. Freitag, and K.-D. Thoben, "Digital twins: A maturity model for their classification and evaluation," *IEEE Access*, vol. 10, pp. 69 605–69 635, 2022.

[33] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 214–223.

[34] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice architecture reconstruction and visualization techniques: A review," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 39–48.

[35] T. Cerny, A. S. Abdelfattah, J. Yero, and D. Taibi, "From static code analysis to visual models of microservice architecture," *Cluster Computing*, vol. 27, no. 4, pp. 4145–4170, Jul. 2024.

[36] F. Antoniazzi and F. Viola, "RDF graph visualization tools: a survey," in *2018 23rd Conference of Open Innovations Association (FRUCT)*, 2018, pp. 25–36.

[37] F. Adesanya, K. C. Silva, V. V. G. Neto, and I. David, "Systems of twinned systems: A systematic literature review," *arXiv preprint arXiv:2505.19916*, 2025.

[38] E. Kamburjan, V. N. Klungre, R. Schlatte, S. L. T. Tarifa, D. Cameron, and E. B. Johnsen, "Digital twin reconfiguration using asset models," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2022, pp. 71–88.

[39] C. Palmer, Y. M. Goh, E.-M. Hubbard, R. Grant, and R. Houghton, "The need for a symbiotic interface for a digital twin," in *Advances in Transdisciplinary Engineering*, P. Koomsap, A. Cooper, and J. Stjepandić, Eds. IOS Press, Nov. 2023.