University of Antwerp

Faculty of Science · Department of Computer Science

A Multi-Paradigm Modelling Foundation for Twinning
within the context of Systems Engineering

Een Multi-Paradigma Modelleerbasis voor Tweelingsystemen in de context van Systeemontwikkeling

Auteur:
Randy Paredis

Promotor:
prof. dr. Hans Vangheluwe

Proefschrift ingediend tot het behalen van de graad van Doctor in de Wetenschappen: Informatica

Contact
Randy Paredis

University of Antwerp
Faculty of Computer Science
AnSyMo - MSDL
Middelheimlaan 1, 2610 Antwerpen, België
M: randy.paredis@uantwerpen.be

Acknowledgements

I would like to thank a couple of people that played an important role during my PhD.

Firstly, a massive amount of appreciation goes towards my supervisor, prof. Hans Vangheluwe. Without his help, enthusiasm and support I would not be where I am now, both academically, and as a person. I truly believe I could not have wished for a better supervisor. You not only showed me the ropes of the academic world (and small sailboats), you also continuously encouraged my eagerness to do research and submerge myself in the wonderful world of Twinning. You gave me so many different opportunities that I am incredibly grateful for. Thank you for the hundreds of hours in (mostly) useful lectures, and for having a contagious passion for scientific research.

I would also like to thank my colleagues at MSDL, MICSS-lab, Cosys-Lab, AnSyMo, and the University as a whole. Thank you Joeri for sharing an office with me and putting up with me for four years. Thank you Rakshit, Lucas, Pamela, Hussein, Arkadiusz, Yon, Bert, and Sahar for the many research discussions and collaborations we had. Yon, in particular, thank you for sharing your way too fancy room with me in San Diego, even though you did not have to. Thank you Burak for the numerous breaks we took and talks we had. To the extended MSDL family (Cláudio, Bentley, and Istvan); thank you for providing feedback and pointers for papers and research, and for the numerous job offers. To all other people of the University (in the department, the faculty and the administration); thank you for your helpfulness and flexibility.

Every scientific work needs to be reviewed and accepted by the scientific community. This thesis is no different. I would like to thank the members of my PhD committee and jury: Moharram Challenger, Joachim Denil, Loek Cleophas, Sanja Lazarova-Molnar, and Giovanni Lugaresi. With all of you I've had interesting discussions that allowed me to rethink some aspects of my research, and gave me some much needed insights. Thank you for your diligent reading and detailed feedback, as it has tremendously improved my thesis.

Additionally, I would like to thank the Port of Antwerp-Bruges for their explanations of internal functionalities of the nautical chain. This research was also partially supported by Flanders Make, the strategic research center for the manufacturing industry.

Thank you mom and dad, without whom I would not have made it this far. Thanks for listening to my countless explanations of things you did not necessarily understand. Thank you for your never-ending love, support and feedback. Thank you for helping me find a path through these past few years. Thank you, Seth, Karo, and Indra for your unwavering support and curiosity.

Thanks to everyone in the Minelabs project. Thanks Maja and Bert for bringing me on board, for making sure all the workshops were well prepared, and for ensuring the entire project was a success. A special thanks to Dante, Ine, and Joey for always being there and putting up with my randomness and numerous rants. Thank you for being there when I needed it most. Thank you for this amazing friendship.

Thank you to the people from Sporta Oostmalle – my extended family, who gave me the necessary distractions outside of the university. In particular: thank you Jordi, Tibo, Ella, Laure, Floor, Jeff, Tobias, Jan, and Inês for being there for me and supporting me all the way. I don't know how I would have survived these years without you.

Thank you Ludo for coaching me in my acting endeavours, wherever they brought me. I am sure that your wisdom allowed me to grow as a public speaker and give better presentations as a result.

And last, but far from least, thank you Camille, Jeremy and Jimmy for all the years we've been friends. Our everlasting friendship that reappears each time we meet is truly special to me.

Randy Paredis
5 March 2025

Abstract

The purpose of Systems Engineering is to analyse, design, optimize, operate, and evolve complex Cyber-Physical Systems (CPSs). This often happens collaboratively and by following complicated workflows. In our current day and age, these systems become increasingly complex, up to the point that simple software does not suffice for their creation. In order to deal with this complexity, the domain of Modelling & Simulation (M&S) allows for system engineers to focus on their specific domain knowledge, without the need for knowing all aspects of the CPSs.

These engineers build models, *i.e.,* abstractions of the reality. Each model has a very specific usage context. For instance, a model for the movement of vessels through a marina is constructed entirely differently to a model that focuses on the safety of a roundabout. Furthermore, the languages in which these models are created all have their own advantages, disadvantages, and ideal use cases. Multi-Paradigm Modelling (MPM) advocates to use the most appropriate languages, frameworks, tools, . . . when explicitly modelling all aspects of a system.

Digital Twins (DTs) are simulation models running in parallel with a real-world system while being fed the same input stimuli as that system. They can be used to analyse, optimize and adapt CPSs. DTs have been identified as a major player in the current industrial revolution. Combined with Big Data, Artificial Intelligence (AI), Internet of Things (IoT), Green IT, and an increased focus on sustainability, they are predicted to cause a major economic boom. Unfortunately, there is still very little consensus on an exact definition for DTs (and their many variations). They have been used to mean anything from M&S techniques to smart IoT solutions. In this work, we use the catch-all *Twinning Paradigm* to refer to all concepts, techniques, architectures, . . . related to these systems. In this paradigm, virtual instances, known as Twin Objects (TOs) (*i.e.,* models), of an Actual Object (AO) (*i.e.,* a System under Study (SuS) in its environment) are continually updated with the SuS's state, health, performance, and maintenance status, over its entire life-cycle.

Throughout the (currently quite ad-hoc) creation of Twinning Systems, a plethora of choices impacts the functionality and performance of the realized system, comprised of the actual SuS and its model. This work identifies four stages at which this variability may appear:

- *Properties of Interest (PoIs) in the Problem Space* – An initial choice of the exact goal(s) and purpose(s) for the Twinning System highly impacts the end product. The literature has identified a vast number of these purpose(s), yet in this work, they are grouped and joined together into a large Feature Tree.
- *(Conceptual) Architecture and Design* – When a choice is made in terms of goal(s), it is important to identify the individual system components required to (functionally)

ensure the valid behaviour of the Twinning System. A *Twinning Experiment (TE)* appears as a first-class entity in this conceptual architecture. It identifies a Twinning System that is actively trying to solve a specific set of PoIs.

- *Modelling & Simulation* – The selection of the exact modelling languages and the creation of the models is heavily impacted by the end goal(s), whilst also a large point of variation in what the resulting Twinning System will look like. This may also be impacted by the models that are already available for the system engineers.
- *Deployment* – Individual tools and frameworks need to be selected, as well as the middleware, the communication protocols... Common choices for network communication such as the Data Distribution Service (DDS) and MQTT each have their own strengths and weaknesses which must be taken into account when trying to satisfy non-functional properties such as meeting real-time deadlines.

Choices made in one stage influence the solutions and subsequent possible choices (or configurations) in the subsequent stage.

As adoption of Twinning increases, the need to combine TEs arises. The question then arises how to combine, compose, and federate such systems. Multiple TEs may correspond to different requirements, goals and PoIs; may correspond to different components in an architecture; may represent a system at different levels of detail, abstraction, and fidelity; may be used at a type, or aggregate level or at instance level; ... These various reasons for combining TEs will be explored and some architectural solutions given. Such combinations can consist of merging in the case of white-box components, or orchestration in the case of black-box components.

Following MPM principles, we propose to explicitly model and use workflows of TEs, as well as their architectures. We apply the concepts of variability (also known as product family) modelling, in particular to Twinning workflows and architectures. This allows for the de-/re-construction of the different variants in a principled, reproducible and partially automatable manner.

The choices that are made have a high impact on the required investment when creating the Twinning System, most notably on the development time and deployment cost. This is especially true when multiple iterations are needed to find the most appropriate level(s) of abstraction and detail, architecture, technologies and tools. This work follows a Model-Based Systems Engineering (MBSE) methodology: before realizing a Twinning architecture, Discrete EVent system Specification (DEVS) models for deployed architecture alternatives can be constructed and simulated, to evaluate their suitability.

Two representative use-cases have been researched:

- Automated Guided Vehicle (AGV): Line Following Robot (LFR) – A simple LEGO robot has a task to follow a visual trajectory on the ground. Alternatively, to show the expansion to a more industrial setting, a TurtleBot has been used as well. The TurtleBot focusses more on the combination of the TEs, whereas the LEGO LFR shows the feasability of the presented approaches.
- Port of Antwerp-Bruges – A very simple model for the movement of vessels in the Port of Antwerp-Bruges has been constructed. Throughout the system execution,

an anomaly occurs and the simulation needs to identify why this has happened. Additionally, a 1D kinematic model of a vessel has been studied in much more detail, including a DEVS simulation of the deployment of this system.

Because both use-cases are in completely different domains, we hypothesise that the proposed methodology is applicable to all use-cases in similar domains, and everywhere in-between.

Nederlandstalige Samenvatting

Het doel van Systems Engineering is het analyseren, ontwerpen, optimaliseren, exploiteren en evolueren van complexe Cyber-Physical Systems (CPSs). Dit gebeurt vaak in samenwerking en volgens gecompliceerde workflows. Vandaag de dag worden deze systemen steeds complexer, tot op het punt dat eenvoudige software niet meer voldoende is om ze te maken. Om met deze complexiteit om te gaan, stelt het domein van Modelling & Simulation (M&S) systeemingenieurs in staat zich te richten op hun specifieke domeinkennis, zonder dat ze alle aspecten van de CPSs hoeven te kennen.

Deze ingenieurs bouwen modellen, dat wil zeggen, abstracties van de realiteit. Elk model heeft een zeer specifieke gebruikscontext. Bijvoorbeeld, een model voor de beweging van schepen door een jachthaven wordt volledig anders opgebouwd dan een model dat zich richt op de veiligheid van een rotonde. Bovendien hebben de talen waarin deze modellen zijn gemaakt allemaal hun eigen voordelen, nadelen en ideale toepassingsgebieden. Multi-Paradigm Modelling (MPM) pleit voor het gebruik van de meest geschikte talen, raamwerken, applicaties... bij het expliciet modelleren van alle aspecten van een systeem.

Digital Twins (DTs) zijn simulatiemodellen die in parallel lopen met een systeem in de echte wereld, terwijl ze dezelfde invoerstimuli ontvangen als dat systeem. Ze kunnen worden gebruikt om CPSs te analyseren, optimaliseren en adapteren. DTs zijn geïdentificeerd als een belangrijke speler in de huidige industriële revolutie. Gecombineerd met Big Data, Artificial Intelligence (AI), Internet of Things (IoT), Groene IT en een verhoogde focus op duurzaamheid, worden ze voorspeld een grote economische bloei te veroorzaken. Helaas is er nog steeds weinig consensus over een exacte definitie voor DTs (en hun vele alternatieven). Ze worden gebruikt voor alles van M&S-technieken tot slimme IoT-oplossingen. In dit werk gebruiken we *Twinning Paradigma* als overkoepelend begrip om te verwijzen naar alle concepten, technieken, architecturen... die gerelateerd zijn aan dit soort systemen. In dit paradigma worden virtuele instanties, bekend als Twin Objects (TOs) (*d.w.z.* modellen), van een Actual Object (AO) (*d.w.z.* een System under Study (SuS) in diens omgeving) voortdurend bijgewerkt met hun status, de gezondheid, prestatie- en onderhoudsinformatie van de SuS gedurende de hele levenscyclus.

Tijdens de (momenteel vrij ad-hoc) creatie van Twinning Systemen, beïnvloedt een overvloed aan keuzes de functionaliteit en prestaties van het gerealiseerde systeem, bestaande uit het feitelijke SuS en het model daarvan. Dit werk identificeert vier stadia waarin deze variabiliteit kan optreden:

- *Properties of Interest (PoIs) (Belangseigenschappen) in de Probleemruimte* – De initiële keuze van de exacte doel(en) en doelstelling(en) voor het Twinning Systeem heeft een grote invloed op het eindproduct. De literatuur heeft een groot aantal van deze doelstelling(en) geïdentificeerd, maar in dit werk zijn ze gegroepeerd en samengevoegd in een grote Feature Tree.

- *(Conceptuele) Architectuur en Ontwerp* – Wanneer een keuze is gemaakt in termen van doel(en), is het belangrijk om de individuele systeemcomponenten te identificeren die nodig zijn voor het correcte gedrag van het Twinning Systeem. Een Twinning Experiment (TE) verschijnt als een hoofdentiteit in deze conceptuele architectuur. Een TE identificeert een Twinning Systeem dat actief probeert om een bepaalde set van PoIs op te lossen.
- *Modelleren & Simulatie* – De keuze van de exacte modelleertalen en de creatie van de modellen wordt sterk beïnvloed door de einddoel(en) van het Twinning Systeem. Tegelijkertijd is dit ook een groot punt van variatie in hoe het resulterende Twinning Systeem eruit zal zien. Dit kan ook worden beïnvloed door de modellen die reeds beschikbaar zijn voor systeemingenieurs.
- *Realisatie* – Individuele applicaties en raamwerken moeten worden geselecteerd, evenals de tussenliggende software, communicatieprotocollen... Gebruikelijke keuzes voor netwerkcommunicatie, zoals de Data Distribution Service (DDS) en MQTT, hebben elk hun eigen sterke en zwakke punten, die in overweging moeten worden genomen bij het streven naar niet-functionele eigenschappen, zoals het voldoen aan realtime deadlines.

Keuzes die in één stadium worden gemaakt, beïnvloeden de oplossingen en de daaropvolgende mogelijke keuzes (of configuraties) in het volgende stadium.

Naarmate dat Twinning toeneemt, ontstaat de behoefte om TEs te combineren. De vraag is dan hoe dergelijke systemen gecombineerd, samengevoegd, of gefedereerd kunnen worden. Meerdere TEs kunnen overeenkomen met verschillende doelstellingen en PoIs; kunnen overeenkomen met verschillende componenten in een architectuur; kunnen een systeem vertegenwoordigen op verschillende niveaus van detail, of abstractie; kunnen worden gebruikt op een type- of aggregatieniveau of op individueel niveau; ... Deze verschillende redenen voor het combineren van TEs zullen worden verkend en enkele architecturale oplossingen worden gegeven. Dergelijke combinaties kunnen bestaan uit samenvoeging in het geval van white-box componenten, of orchestratie in het geval van black-box componenten.

Volgens de MPM-principes stellen we voor om workflows van TEs expliciet te modelleren en te gebruiken, evenals hun architecturen. We passen de concepten van variabiliteitsmodellering (ook wel productfamilies genoemd) toe, met name op Twinning-workflows en architecturen. Dit stelt ons in staat om de verschillende varianten op een principiële, reproduceerbare en deels automatiseerbare manier af te breken of herop te bouwen.

De keuzes die worden gemaakt hebben een grote impact op de benodigde investering bij het creëren van een Twinning Systeem, met name op de ontwikkelingstijd en de implementatiekosten. Dit is vooral het geval wanneer meerdere iteraties nodig zijn om het meest geschikte abstractie- en detailniveau, de architectuur, technologieën en applicaties te vinden. Dit werk volgt een Model-Based Systems Engineering (MBSE) methodologie: voordat een Twinning-architectuur wordt gerealiseerd, kunnen Discrete EVent system Specification (DEVS) modellen voor alternatieve geïmplementeerde architecturen worden gebouwd en gesimuleerd om hun geschiktheid te evalueren.

Twee representatieve toepassingen werden onderzocht:

- Automated Guided Vehicle (AGV): Line Following Robot (LFR) – Een eenvoudige LEGO-robot met de taak om een visuele route op de grond te volgen. Om de uitbreiding naar een meer industriële setting te tonen, is ook een TurtleBot gebruikt. De TurtleBot focust meer op het combineren van TEs, en de LEGO LFR op de geschiktheid van de voorgestelde technieken.
- Port of Antwerp-Bruges – Een zeer eenvoudig model voor de beweging van schepen in de Port of Antwerp-Bruges werd gebouwd. Tijdens de uitvoering van het systeem doet zich een anomalie voor en de simulatie moet identificeren waarom dit is gebeurd.
  Daarnaast is er een 1D-kinematisch model van een schip in veel meer detail bestudeerd, inclusief een DEVS-simulatie van de implementatie van dit systeem.

Omdat beide toepassingen in totaal verschillende domeinen zitten, veronderstellen we dat de voorgestelde methodologie toepasbaar is op alle toepassingen in soortgelijke domeinen, en overal daartussenin.

Publications

The following list of peer-reviewed papers (and a poster) that I co-authored formed a basis for this dissertation.

- Paredis, R., & Vangheluwe, H. (2021). Exploring a Digital Shadow Design Workflow by Means of a Line Following Robot Use-Case. *Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM)*, 1–12

  Hans and I came up with the idea. I created the Line Following Robot (LFR) and the Formalism Transformation Graph and Process Model (FTG+PM). I wrote the paper. Hans reviewed and modified the paper accordingly.

- Paredis, R., Gomes, C., & Vangheluwe, H. (2021). Towards a Family of Digital Model / Shadow / Twin Workflows and Architectures. *Proceedings of the 2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2021)*, 174–182

  Hans, Cláudio and I came up with the idea. Cláudio created the initial Feature Trees. Cláudio wrote about the incubator use-case and I wrote about the LFR use-case. I worked on the system architectures and the workflow (which was coordinated with Cláudio). Hans reviewed and modified the paper accordingly.

- Paredis, R., & Vangheluwe, H. (2022). Towards a Digital Z Framework Based on a Family of Architectures and a Virtual Knowledge Graph. *Companion Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS-C)*, 491–496

  Hans and I came up with the idea. I created the LFR and the FTG+PM. I wrote the paper. Hans reviewed and modified the paper accordingly.

- Paredis, R., Gomes, C., & Vangheluwe, H. (2023). A Family of Digital T Workflows and Architectures: Exploring Two Cases. In A. Smirnov, H. Panetto, & K. Madani (Eds.), *Innovative intelligent industrial production and logistics* (pp. 93–109). Springer Nature Switzerland

  This chapter was an extension of Paredis, Gomes, and Vangheluwe (2021). Hence, Hans, Cláudio and I came up with the idea. Cláudio wrote about the incubator case and I wrote about the LFR. I updated the Feature Trees. Cláudio and I wrote the chapter. Hans reviewed and modified the chapter accordingly.

- Marah, H., Paredis, R., Challenger, M., & Vangheluwe, H. (2023). A Multi-Robot Warehouse System: An Exemplar. *Proceedings of the 2023 ACM/IEEE International*

Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 530–538

Moharram came up with the idea. Hussein focused on the warehouse aspect, whereas I focused on the individual robots. Hussein and I wrote the paper. Hans and Moharram reviewed and modified the paper accordingly.

- Paredis, R., Vangheluwe, H., & Albertins, P. A. R. (2024). *COOCK project Smart Port 2025 D3.1: "To Twin Or Not To Twin"* (tech. rep.) (ArXiv preprint). University of Antwerp

  This paper was a deliverable for the Smart Port 2025 project with Port of Antwerp-Bruges. Port of Antwerp-Bruges came up with the idea (in collaboration with Hans). I analysed a lot of the literature to construct the data presented in the paper. Pamela focused on cost analysis. I wrote the paper and Hans reviewed and modified the paper accordingly.

- Paredis, R., & Vangheluwe, H. (2024a). Exploring Twinning Variability [Poster]. *Proceedings of the 4th Conference on Machines, Vehicles and Production Technology (CMVPT)*

  Hans and I came up with the idea. I focused on the architecture. I created the poster. Hans reviewed the poster accordingly.

- Paredis, R., & Vangheluwe, H. (2024b). Modelling and Simulation-Based Evaluation of Twinning Architectures and Their Deployment. *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, 170–182. https://doi.org/10.5220/0012865300003758

  Hans and I came up with the idea. I created the experimentation setup and came up with viable experiments. I wrote the paper. Hans reviewed and modified the paper accordingly.

The following list of peer-reviewed papers that I co-authored are not core contributions within this thesis, but are nonetheless an important basis to support the previous list.

- Paredis, R., Denil, J., & Vangheluwe, H. (2021). Specifying and Executing the Combination of Timed Finite State Automata and Causal-Block Diagrams by Mapping onto DEVS. *Proceedings of the 2021 Winter Simulation Conference (WSC)*

  Hans, Joachim and I came up with the idea. I created the CBD simulator and embedded it in DEVS. I wrote the paper. Hans reviewed and modified the paper accordingly.

- Paredis, R., Exelmans, J., & Vangheluwe, H. (2022). Multi-Paradigm Modelling for Model-Based Systems Engineering: Extending the FTG+PM. *Proceedings of the 2022 Annual Modeling and Simulation Conference (ANNSIM)*, 461–474

  Hans came up with the idea. Joeri and I discussed the ideas and wrote the paper. Hans reviewed and modified the paper accordingly.

The following list of peer-reviewed papers that I co-authored are not part of the research of this thesis.

- Paredis, R., Van Mierlo, S., & Vangheluwe, H. (2020). Translating Process Interaction World View Models to DEVS: GPSS to (Python(P))DEVS. In K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, & R. Thiesing (Eds.), *Proceedings of the 2020 winter simulation conference* (pp. 2221–2232). Institute of Electrical; Electronics Engineers, Inc.

  This is a paper that summarizes my work in translating GPSS to DEVS, as created for my Master thesis. Hans came up with the idea. I wrote the paper, heavily supported by Simon's insights and pointers. Hans reviewed and modified the paper accordingly.

- Van Tendeloo, Y., Paredis, R., & Vangheluwe, H. (2020). An Introduction To Modular Modeling And Simulation With PythonPDEVS And The Building-Block Library PythonPDEVS-BBL. *Proceedings of the 2020 Winter Simulation Conference (WSC)*, 1152–1166

  This tutorial is a rework of a previous tutorial by Hans and Yentl. I contributed a set of building blocks for DEVS, as described in my Master's thesis. Hans wrote the paper.

- Parezys, J., Paredis, R., & Vangheluwe, H. (2023). CLAVS/ODVS: Combining Class/Object Diagrams and DEVS. *Proceedings of the 2023 Winter Simulation Conference (WSC)*, 2591–2602

  Hans came up with the idea. Jordan researched this topic during his Master's thesis. I wrote the paper. Hans reviewed and modified the paper accordingly.

- Van Tendeloo, Y., Paredis, R., & Vangheluwe, H. (2023). An Introduction to Discrete-Event Modeling and Simulation with DEVS. *Proceedings of the 2023 Winter Simulation Conference (WSC)*, 1531–1545

  This tutorial is a rework of a previous tutorial by Hans, Yentl and I. I contributed the use-case description on the Port of Antwerp. Hans wrote the paper.

Finally, there are also some journal papers currently in preparation. Note that there is no guarantee that these will be accepted. Below, the working titles are listed.

- Denis, H., Paredis, R., Albertins, P., Vangheluwe, H., Farzadmehr, M., Carlan, V., Vanelslander, T., Luong, N.-Q., & Mercelis, S. (2025). Towards Smart Port of the Future: Harnessing Ai and Simulation Models for Nautical Chain Optimization. *Transportation Engineering*

  This paper contains the results for the COOCK SmartPort 2025 project and this was submitted to a Transformation Engineering special issue. The Port of Antwerp-Bruges came up with the original project scope, which was later reduced by imec, Department of Transport and Regional Economics (TPR) and Antwerp Systems and software Modelling (AnSyMo). I researched and wrote the (Discrete EVent system Specification (DEVS)) simulation, imec researched and wrote about the Artificial Intelligence (AI) models, and TPR researched and wrote about the KPIs. All authors reviewed the paper.

- Nezhad, S. N., Paredis, R., Van Acker, B., & Vangheluwe, H. (n.d.). Combining experiments and a Historian for building Twinning systems, applied to a TurtleBot use-case

This paper will be the journal version of the second half of Chapter 7, whilst also expanding on Chapter 6. The TurtleBot use-case will be discussed in detail, and it will be shown how multiple Twinning Experiments (TEs) can be combined when deploying this system. Additionally, the Historian's uses will be discussed in much more detail. Note that the authors are not fixed for this work.

Activities

This chapter gives a brief overview of the academic activities I have actively partaken in during my PhD. Note that this list is not exhaustive and mainly focuses on my academic attributions. These activities are to some extend related to my research.

Organization of Scientific Activities

- Organizing Committee for the 4th International Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) – 2022
- Organizing Committee for the 5th International Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) – 2023
- Organizing Committee for the 6th International Workshop on Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) – 2024

Reviewing Scientific Posters and Papers

I reviewed posters for:

- the 26th International Conference on Model-Driven Engineering Languages and Systems (MoDELS) – 2023
- the 27th International Conference on Model-Driven Engineering Languages and Systems (MoDELS) – 2024

I reviewed papers for:

- the Journal of Simulation Modelling Practice and Theory 2021 (TJSM'21)
- the Journal of Simulation Modelling Practice and Theory 2022 (TJSM'22)
- the Journal of Simulation Modelling Practice and Theory 2023 (TJSM'23)
- the 2022 Annual Modeling and Simulation Conference (ANNSIM'22)
- the 2023 Annual Modeling and Simulation Conference (ANNSIM'23)
- the 2024 Annual Modeling and Simulation Conference (ANNSIM'24)

- the 2025 Annual Modeling and Simulation Conference (ANNSIM'25)
- Simulation: Transactions of the Society for Modeling and Simulation International (2025)

Projects

- I worked on a detailed Causal-Block Diagram (CBD) simulator.
- I gained repository ownership for PythonPDEVS (Van Tendeloo & Vangheluwe, 2015)[1] and I actively maintained this project.
- I worked on the COOCK project "*Smart Port 2025: improving and accelerating the operational efficiency of a harbour eco-system through the application of intelligent technologies*", with a focus on nautical chain optimization of the tugboats and pilots behind the locks. This project was a collaboration between imec-UAntwerpen, TPR – University of Antwerp, AnSyMo – University of Antwerp, and the Port of Antwerp-Bruges.
- I collaborated on Minelabs[2], a scientific expansion of Minecraft. It is meant for high school students to experiment in a fun and interactive way with physics and chemistry. This project was funded by imec within the scope of "*Smart education @ schools*". It was a collaboration between UAntwerpen, KLA and PITO Stabroek. Minelabs Essentials was released as part of Minecraft Education on October 4th, 2024 (Segers, 2024).

Teaching

- Teaching Assistant for the "Model-Driven Engineering" course at the University of Antwerp (2021-2022)
- Teaching Assistant for the "Modelling of Software-Intensive Systems" course at the University of Antwerp (2020-2024)

Participation in Scientific Activities

2020

- The 23th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2020), Virtual Conference. 16 - 23 October 2020.
- The Winter Simulation Conference 2020 (WSC'20), Virtual Conference. 14 - 18 December, 2020.

[1]https://msdl.uantwerpen.be/git/yentl/PythonPDEVS
[2]https://minelabs.be/

2021

- The Flanders MAKE Scientific Conference[3], Virtual Conference. 4th of February 2021.
- The 2021 Annual Modeling and Simulation Conference (ANNSIM 2021), Virtual Conference. 19 - 22 July, 2021.
- The NEMO Summer School 2021, Virtual Conference. 19 - 30 July, 2021.
- The 24th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2021), Virtual Conference. 10 - 15 October, 2021.
- The 2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2021), Virtual Conference. 25 - 27 October, 2021.
- The Winter Simulation Conference 2021 (WSC'21), Virtual Conference. 14 - 16 December, 2021.

2022

- The 2nd Conference on Machines, Vehicles and Production Technology (CMVPT), Flanders Expo, Ghent, Belgium. 22th of March 2022.
- Workshop on Fidelity in Systems Engineering, Lisbon, Portugal. 30 - 31 May 2022.
- The micro MPM Workshop, University of Antwerp, Antwerp, Belgium. 14 June 2022.
- The ADEPT Workshop 2022, Het Pand, Ghent, Belgium. 17th of June 2022.
- The 2022 Annual Modeling and Simulation Conference (ANNSIM 2022), San Diego State University, San Diego, CA, USA. 18 - 20 July 2022.
- The 2022 ESI Symposium, Theatre De Schalm, Veldhoven, Meiveld 3, the Netherlands. 27th of September 2022.
- The 25th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2022), Montreal, Canada. 23 - 28 October 2022.

2023

- The 18th Computer Automated Multi-Paradigm Modelling (CAMPaM) workshop with a focus on Model Based Systems Engineering for and by Digital Twins, Institute of Scientific Studies of Cargèse, Corsica, France. 20 - 24 March 2023.
- MSDL Summer Research Day, University of Antwerp, Antwerp, Belgium. 1st of September 2023.
- The 26th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2023), Västerås, Sweden. 1 - 6 October 2023.

[3]Later renamed to "Conference on Machines, Vehicles and Production Technology (CMVPT)".

2024

- The 4th Conference on Machines, Vehicles and Production Technology (CMVPT), Kortrijk, Belgium. 26th of March 2024.
- The 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2024), Dijon, France. 10 - 12 July 2024.

Contents

List of Figures

"All the secrets of the world are contained in books. Read at your own risk."

– Lemony Snicket

Chapter 1 Introduction

From the human body to photosynthesis, from hurricanes to human interactions, from gravity to factories... the world we know and love consists of countless interacting processes (or *systems*) that span any number of domains. Over time, each of these domains have embraced software as a core component, yielding *software-intensive systems* (Giese, 2006). Cyber-Physical Systems (CPSs) are defined as systems that consist of *cyber* components (as computerized implementations) and *physical* components (Carreira et al., 2020). The main focus here is on the interaction between these components. The cyber part should be able to cause the physical component to change state, and such a change will feed-back into a state change on the cyber component.

The main goal of Systems Engineering is to design, integrate, and manage systems over their entire lifecycle (Kassiakoff et al., 2010). Commonly, this is done by separating a system in its subcomponents, each of them individual parts of the system, with their own functionalities. Ackoff (1971) defines a system as *"a set of interrelated elements"*. It can be considered *simple* when there is a limited number of subcomponents and the output(s) of each subcomponent can easily be derived from their input(s). It is called *complicated* when the total number of subcomponents is too large to get a good understanding by just looking at it. As soon as randomness and human interaction are introduced (such that the output(s) of a subcomponent cannot easily be determined), a system can be considered *complex* (Grieves & Vickers, 2017; Mitchell, 2009).

Creating, deploying, modifying and maintaining a very large, complex CPS is a difficult undertaking. There is a lot that can go wrong on every level in this process. Modelling & Simulation (M&S) techniques can help solve the plethora of issues that may arise in this context.

A model of a system is an abstraction of that system in a specific *validity frame*. It is a specific, contextualized *view* on this system. Constructing models allows us to better grasp their correlated systems and thus the world we live in. A modelling formalism defines the *syntax* (*i.e.,* the set of allowed constructs, including their textual or visual representation) and the *semantics* (*i.e.,* the unambiguous evaluation of the models). Model-Based Systems Engineering (MBSE) combines Systems Engineering and M&S such that complex CPSs can be created, abstracted, analysed, modified, ... from a model.

Multi-Paradigm Modelling (MPM) advocates modelling every part of a system explicitly, using their most appropriate modelling language(s), the most appropriate tool(s), ... at the most appropriate level(s) of abstraction and detail (Mosterman & Vangheluwe, 2004). This also encourages using multiple models for the same system (Batty, 2021). Digital Twins (DTs) are already commonly used by system engineers to create CPSs. MPM provides guidance in the choice of modelling formalisms and workflows. For DTs, it is emphasised that there should be *"different kinds of twins"* (Arcaute et al., 2021), thus encouraging the use of MPM.

1.1 Motivation

As outlined in the Reference Architecture Model for Industry 4.0 (RAMI 4.0) (Hankel & Rexroth, 2015), we are currently at the frontier of the fourth major industrial revolution. This revolution is also commonly called *Industry 4.0*, *Industry 5.0*, or (more accurately) *Industry 4.0S* (Raja Santhi & Muthuswamy, 2023), where the "S" stands for "sustainability". The increased rise in power and usability of Artificial Intelligence (AI) techniques, increasing complexity of CPSs, and ever-increasing volumes of big data push our world towards digitization. As a result, DTs have started to emerge as a major backbone for Industry 4.0S.

Figure 1.1 shows the worldwide Google Trends for the topic *Digital Twin*. Note that the number on the plot represents *"search interest relative to the highest point on the chart"*. It is clear that the introduction of DT in 2014 (Grieves, 2014) and its exploration in 2017 (Grieves & Vickers, 2017) caused a large surge in interest.

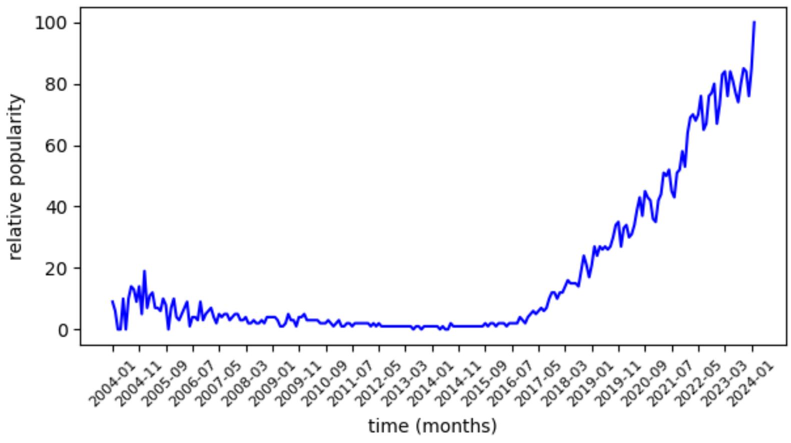


Figure 1.1: Worldwide Google Trends for the *Digital Twin* topic, as obtained on February 7th 2024.

Similarly, a search on Scopus using the terms `"digital twin*" OR "digital shadow*"` yielded a total of 85,759 documents between 1996 and 2024; with 32,870 documents in 2024 alone. This is visualized in Figure 1.2. 26,921 of these documents have been published by China; 9,954 by the US; and 7,587 by Germany.

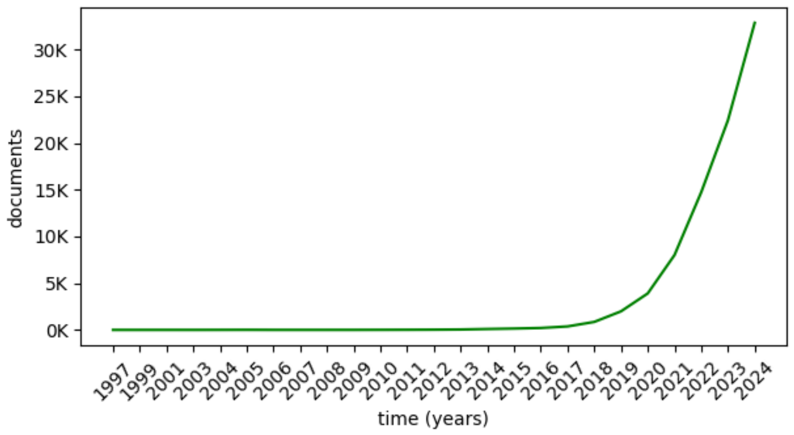
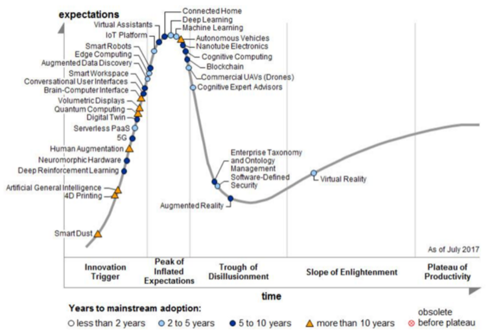
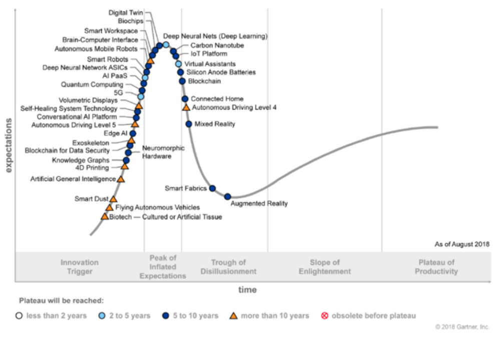


Figure 1.2: Scopus document count between 1996 and 2024 for `"digital twin*" OR "digital shadow*"`, as obtained on February 22nd 2025.

In the 2017 Gartner Research Hype Cycle for Emerging Technologies (see Figure 1.3a)[1], we can find DT at the *Innovation Trigger* stage. In the 2018 edition (see Figure 1.3b)[2], it had already moved to the *Peak of Inflated Expectations*. Surprisingly, later reports do no longer mention DTs.



(a) 2017 Gartner Research Hype Cycle



(b) 2018 Gartner Research Hype Cycle

Figure 1.3: Gartner Research Hype Cycles for Emerging Technologies

DTs (and their various forms) appear to be omnipresent in any number of domains (Dalibor et al., 2022; Paredis et al., 2024), albeit often used as a vacuous marketing instrument. While there is a vast body of research on theoretical frameworks, the practical implementations and deployment are still often omitted (Hassan & Aggarwal, 2023).

There is still a lot of research to be done, but they already solve real problems and use a number of mature techniques and technologies from M&S. Hence, we hypothesize that DT has reached the *Slope of Enlightenment* on the hype cycle. In 2017, Market Research Future predicted that the DT market would reach 15 billion USD worldwide by 2023

[1]https://www.gartner.com/en/documents/3768572
[2]https://www.gartner.com/en/documents/3885468

(El Saddik, 2018; Market Research Future, 2017). In reality, 16.8 billion USD was reached. Horizon Grand View Research (2024) predicts a growth of 37.5%, totalling for USD 155 billion, by the end of 2030.

Despite its growing popularity, a vast number of different DT definitions exist. Rumpe (2021) mentions 112 different definitions[3], found during the largest DT literature study to date (Dalibor et al., 2022). While there is some consensus on these definitions, some are too domain-specific (Steinmetz et al., 2018), whereas others are incredibly vague (Yacob et al., 2019) or even contradictory (Autiosalo et al., 2019). As a result, related terminology (like *Digital Shadow*, *Digital Model*, *Digital Avatar*, *Digital Passport*, *Digital Thread*,. . .) also suffers from this ambiguity.

In fact, the term *Digital Twin* has been used to mean anything between M&S and Internet of Things (IoT). For instance, Damköhler (2022) uses it to refer to a simulation, whereas Ferguson (2020), Mahmoud et al. (2023), and Z. Zhang et al. (2022) apply it in the IoT domain. This is where we introduce the *Twinning Paradigm* as the entire spectrum between those research areas. It is the conceptual notion of the interaction between a System under Study (SuS) in its environment and its corresponding model, and how both are connected throughout the entire life-cycle of the system. A main reasoning behind this paradigm is that a solution to any problem in this spectrum, will likely be applicable to other problems in the spectrum. This thesis will contribute to the Twinning Paradigm by using DTs (and its many variations) as a main point of reference.

Twins are often built for already realized systems, forcing system engineers to drastically modify their architecture. This is usually done in an ad-hoc manner, where very little attention is given to the choices made throughout the construction and deployment of these Twinning Systems. In reality, there is a large amount of variability that can seep in. These differences can be as little as using different communication protocols, but also as big as the purpose for which the twin is used.

Thus, we can define the following research questions:

- **RQ1:** What are the most common reasons and definitions for (creating) DTs?
- **RQ2:** Given the large number of existing DTs in the literature, can we unify?
- **RQ3:** What is the relationship between specific DT requirements, the system architecture, the used models, and the eventual deployment?
- **RQ4:** How to quantitatively support deployment choices?
- **RQ5:** How can we combine multiple DTs into a larger system?

1.2 Contributions and Gaps

In Paredis, Gomes, and Vangheluwe (2021), we hinted at the "multiple definitions problem" and tried to introduce *Digital X* as a solution (where the *X* identified any DT related term). In Paredis and Vangheluwe (2022), this became *Digital Z*, where the *Z* referred to

[3]All are listed on https://awortmann.github.io/research/digital_twin_definitions/.

the demographic generation naming convention (Generation X, Generation Y, Generation Z, Generation Alpha,...). Later, we referred to this as *Digital T* (Paredis et al., 2023), before we actually started focusing on the *Twinning Paradigm* (Paredis et al., 2024).

A Twinning System consists of an Actual Object (AO) (*i.e.,* a SuS in its environment) that is *continually* kept *in-sync* with a Twin Object (TO) (*i.e.,* a model of that SuS). Notice the usage of "continual" as opposed to "continuous". Whereas the latter implies "*an uninterrupted execution*"[4], "continual" can be defined as "*in a constantly repeated manner: over and over*"[5]. As such, normal M&S is not necessarily continual, as there simply needs to exist an original (Stachowaik, 1973). As soon as this continuality is introduced, the Twinning Paradigm arises.

Important to denote here is that the AO does not have to exist in the real, physical world (for instance, it can be a set of data traces, or even a set of algorithms), and neither does the TO need to be digital. In the case that the TO is digital, the Twinning System can be considered a DT.

We will consider a Twinning Experiment (TE) a Twinning System that only focuses on a single Property of Interest (PoI). For instance, when the goal is *anomaly detection*, the corresponding TE will execute an *anomaly detection experiment*. When the goal is *fault diagnosis*, there will be a *fault diagnosis experiment*. Note that we define an experiment as follows: "*an experiment is an intentional set of (possibly hierarchically composed) activities, carried out on a specific SuS in order to accomplish a specific set of goals.*" A Twinning System can consist of one or more TEs. Hence, we will use TEs as first-class entities in the realization of a Twinning architecture. Because each TE conceptually focuses on a single goal, the TE will encapsulate each AO-TO combination.

This thesis focuses on the creation of TEs and their usage. Mainly, on *why* and *how* they are used, including the many different possibilities of this usage. We have identified four main stages where Twinning variability may occur. They are shown in Figure 1.4. This workflow of stages will be applicable for systems engineering, with a specific focus on Twinning. Important to denote is that each of these stages has their own (set of) workflow(s), which is part of an overall set of workflows that can be used to construct a TE. Note the relationship to the IIRA Architecture Framework (Industry IoT Consortium, 2022) and The ISO/IEC 12207 standard (R. Singh, 1996).

Stage Ⓐ (*Properties of Interest in the Problem Space*)
Twins are always constructed with very specific goals or purposes in mind – sometimes known as *digital twin capabilities* (Ali et al., 2025). The architecture of a Twinning System that focuses on safety-monitoring will be different to a Twin for predictive maintenance. Based on existing studies in the literature (Dalibor et al., 2022; Jones et al., 2020; Minerva et al., 2020; Van der Valk et al., 2020; Wanasinghe et al., 2020), a non-exhaustive list of requirements was created in the form of a Feature Tree (Kang & Lee, 2013; Paredis et al., 2024). It classifies the most common goals for constructing TEs, and focuses on a set of preliminary questions that system engineers should consider.

Stage Ⓑ (*Design (Conceptual) Architectures*)
This stage helps in identifying which goals lead to which components, and which resulting conceptual (or functional) system architecture will be the result. Each goal leads to

[4]https://www.merriam-webster.com/dictionary/continuous
[5]https://www.merriam-webster.com/dictionary/continually

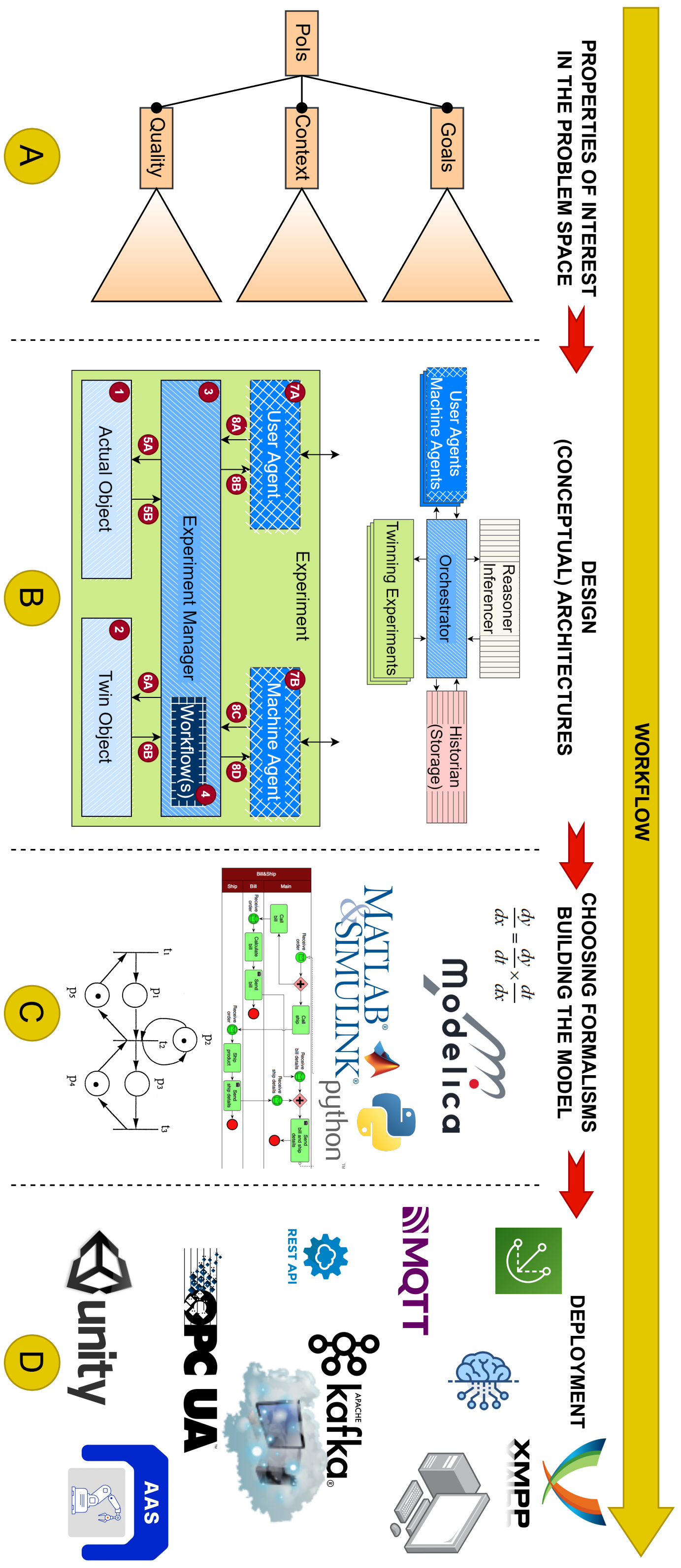


Figure 1.4: Twinning variability driven by workflows.

a single experiment, so a subset of goals will lead to a collection of experiments. These experiments are conceptually separated, but may need to collaborate with each other. At a high level, an orchestration unit may manage these experiments, do some reasoning, store the data, *etc.*

Of course, this is still only at the conceptual level, and many experiments might need to be combined. If all components are a so-called *white box*, introspection is possible, and the combination of experiments can be done easily. When they are *black box*, more complicated scenarios similar to co-simulation orchestration are needed.

Stage Ⓒ (*Choosing Formalisms / Building the Model(s)*)
The previous stage yielded a set of components and parts, including a description of how they must be combined in an architecture. Using the Feature Trees, and a set of additional requirements, it is possible to select which formalisms (or languages) are the most appropriate to implement the given requirements. Notice how the MPM methodology shines through in this phase.

Once all the formalisms were selected, MBSE techniques can be used to create these models. An Formalism Transformation Graph (FTG) can show how these are then combined.

Stage Ⓓ (*Deployment*)
Deployment is the concrete realization of a system on a hardware and software level. The components are deployed on specific hardware, using predefined protocols. Networked connections between components are constructed, such that the architectures from Ⓑ reappear here, albeit more specialized. There exist a plethora of possible technologies to choose from, and system engineers can be informed by what is available to them and, if possible, assisted in their choice.

MBSE often uses simulation to prototype a system, before it is deployed. This reduces cost, workload, and manpower. We can use a similar approach here, where the deployment is done in simulation, and a deployment space exploration is performed. This was done in Paredis and Vangheluwe (2024b) to verify which of the available technologies would be the best fit for a system.

Eventually, the system can be realized.

1.3 Roadmap

Chapter 2 provides a short background to the techniques used for this thesis, mainly focusing on the modelling tools and frameworks that were used. It also discusses how DTs came to be and some of the current-day research on it.

The main idea behind the next chapters is visualized in Figure 1.4. Stages Ⓐ, Ⓑ, and Ⓓ are discussed respectively in Chapter 3, Chapter 4, Chapter 5. Here, Chapter 3 aims to answer ***RQ1***, and Chapter 4 answers ***RQ2***. Stage Ⓒ focuses on MBSE and MPM techniques for creating (or reusing) models and their languages. This thesis will not delve deeper into this stage, however some scientific contributions were also made on this front, as discussed in Chapter 2. Chapter 3, Chapter 4, and Chapter 5 outline the details for Figure 1.4 by focusing on a single goal and a single PoI. Chapter 6 on the other hand focuses on multiple goals, and how to combine the previously discussed techniques. The main focus here will be on expanding Chapter 4, yet all four stages will be discussed. It therefore focuses around ***RQ5***.

To illustrate the validity of this research and to experiment with, a simple Line Following Robot (LFR) was constructed in Paredis and Vangheluwe (2021). Marah et al. (2023) expands on this idea by placing it in the context of a shop-floor or warehouse. Chapter 7 delves deeper in the LFR models that were researched within the context of differential-drive robots. It is applied to a simple LEGO robot, eventually resulting in a Digital Shadow (as per Kritzinger et al. (2018)'s definition). Another application focuses on a more industrial domain, as it creates a Twinning System with multiple experiments for a TurtleBot use-case.

Additionally, Antwerp Systems and software Modelling (AnSyMo) was part of the COOCK project "*Smart Port 2025: improving and accelerating the operational efficiency of a harbour ecosystem through the application of intelligent technologies*". Together with imec and Department of Transport and Regional Economics (TPR) (at the University of Antwerp), the Port of Antwerp-Bruges tasked us to construct an optimization process for their nautical chain. Our contributions in this project mainly focus on simulation and animation. As will be discussed in Chapter 8, this can be considered a twin as well. Information provided by the domain experts at Port of Antwerp-Bruges and execution traces from 2022 allowed us to create a posthumous Twinning System.

Similar to the LFR, the Port of Antwerp-Bruges use-case will focus on two cases: (1) the movement of vessels throughout the port, and (2) the behaviour of a 1D ship. This latter case is simplified such that the deployment phase can be discussed in more detail, including a simulation of the deployment (thus verifying its correctness). This provides a proof-of-concept answer to ***RQ4***.

The LFR use-case is vastly different from the Port of Antwerp-Bruges situation. Hence, if we can apply the same techniques to the LFR as to the Port of Antwerp-Bruges, we hypothesise that these techniques can also be applied anywhere in-between. This was not verified. Similarly, the use-cases that lie outside of this range are left as an exercise to the reader.

Chapter 7 and Chapter 8 not only show a proof-of-concept for each of the stages, they also aim to answer ***RQ4*** by following the presented workflow in detail. They contain specific use-cases, but they can be generalized to other use-cases in other domains.

Finally, some concluding remarks and identifications for future work is presented in Chapter 9.

"History does not repeat itself, but it does rhyme."
– Mark Twain

Chapter 2 Background

This chapter provides a background for this thesis. It presents a number of concepts and techniques that will be used to create a modelling foundation for the Twinning Paradigm. Section 2.1 will outline the current state-of-the-art for DTs and Twinning in general. More related work will be highlighted in future chapters, but this section provides a basic understanding of what DTs are and how to use them. Next, Section 2.2 will introduce the MPM methodology that was briefly hinted at in the introduction. MPM is a core concept in this thesis, as it allows us to focus on formalism-independent solutions to Twinning problems. In Section 2.3, an overview of the general formalisms that were used in this research will be provided. Using multiple formalisms for their individual purposes shows the practical application of the MPM ideology. Finally, in Section 2.4, the most common communication protocols for Twinning will be discussed.

Additionally, in Chapter 10, some additional concepts are also introduced, as well as where some ideas of this work came from. This appendix contains some not too scientific background information and thus is not an important part of this work, but is included for completeness.

2.1 Digital Twins: State-of-the-Art

In 2001, the concept of Digital Twins (DTs) was born. Figure 2.1 was shown during a presentation that focused on Product Lifecycle Management (PLM) in the aerospace industry (Grieves & Vickers, 2017). PLM focuses on the trajectory of a company's products throughout their lifetime (Stark, 2022), which is massively helped with DTs.

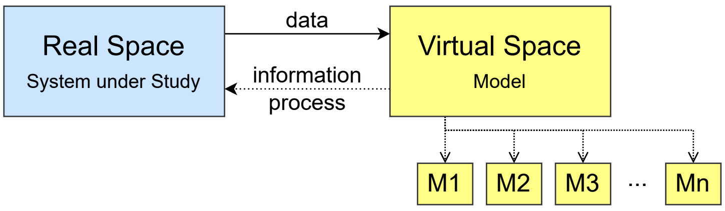


Figure 2.1: The "*conceptual model for PLM*", adapted from Grieves and Vickers (2017).

In essence, it combines a SuS with a virtual copy (*i.e.,* a model) thereof, such that additional analysis, optimization, and adaptation becomes available. The figure illustrates this by showing a *Real Space* that transfers some data (commonly sensor and environmental data) to a *Virtual Space*. This *Virtual Space* can then control and reply to the *Real Space* by forwarding analysis information. Multiple instances of the *Virtual Space* can exist, each of them allowing different kinds of analysis. The idea behind DTs is also known as *Symbiotic Simulation Systems* (Aydt et al., 2008) *Virtual Digital Fleet Leader* (Glaessgen & Stargel, 2012), *Information Mirroring Model* (Grieves, 2008), and *Mirror Worlds* (Gelernter, 1993), but *Digital Twin* is the common consensus since 2010 (M. Singh et al., 2021).

The *Real Space* is also referred to as *Physical Shop-Floor* (Tao & Zhang, 2017), *Physical Counterpart* (Dalibor et al., 2022), *Real Twin* (El Saddik, 2018), *Physical Object* (Becker et al., 2021; Kritzinger et al., 2018; Paredis & Vangheluwe, 2021), *Physical Entity* (Jones et al., 2020; Negrin et al., 2021), *Physical Twin* (David et al., 2023), *Twin of Interest* (Diakité & Traoré, 2023), or *Actual System* (Eramo et al., 2021; Tao et al., 2022), to name a few.

Whereas "Physical" often recurs here, this is not always the case. For instance, Heithoff et al. (2023) aims to use DTs for sustainable software systems. And Rambow-Hoeschele et al. (2018) builds a DT for business processes. Hence, a better name would be AO, which combines the more general "Actual" with the "Object" term from Kritzinger et al. (2018).

Similarly, the *Virtual Space* is also considered a *Virtual Shop-Floor* (Tao & Zhang, 2017), a *Digital Object* (Becker et al., 2021; Kritzinger et al., 2018), a *Logical Object* (Minerva et al., 2020), a *Virtual Entity* (Jones et al., 2020), or a *Digital Twin* (Bibow et al., 2020; Bolender et al., 2021; Kibira et al., 2021; Madni et al., 2019; Tekinerdogan & Verdouw, 2020).

"Virtual" actually means "abstracted" (*i.e.,* a model), but nowadays, it is more often used as a synonym for "digital". This is also how most of the above terminologies are usually interpreted, but this is not necessarily the case: the model does not necessarily exist in the digital space. The same can be said for this *Virtual Space*: there is no requirement that the model exists only digitally. Tero et al. (2010) and Topuzoglu et al. (2019) discuss how biological slime molds can be used to map out subway networks. Ferguson (2020) discusses how an electromechanical system was used as a *Virtual Entity*[1]. Even so-called Sand Tables (see Chapter 10) can technically be considered a *Virtual Entity*, even though they are fully analog. Hence, the TO is introduced to also capture those particularities. The word "twin" was chosen to avoid the virtual/digital confusion, and "object" for the same reasoning as before. Whenever the TO is not in the digital domain, we can consider the twin an Analog Twin (AT), as opposed to a DT.

2.1.1 Digital Twin Architectures

A common categorization of DT systems was done by Kritzinger et al. (2018), where most notably the connection between the AO and the TO was analysed. This is visualized in Figure 2.2. Whenever this connection is mainly manual, the authors suggest calling such a system a *Digital Model*. Whenever the connection is automated from the AO to the TO, allowing an automated transfer of sensor-data and environmental information, the authors suggest using *Digital Shadow*. Finally, a *Digital Twin* appears if this connection

[1]Note that this might have had digital components, but did not really exist in a digital space.

is automated in both directions. Note that this connection will never be *fully* automatic. For instance, a smart lightning system still needs a manual light bulb replacement.

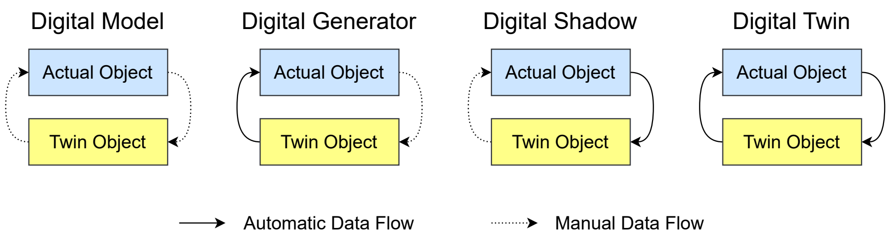


Figure 2.2: Digital Model vs Digital Generator vs Digital Shadow vs Digital Twin (Kritzinger et al., 2018; Tekinerdogan & Verdouw, 2020).

Tekinerdogan and Verdouw (2020) extends this categorization with a *Digital Generator* (*i.e.,* the dual of the *Digital Shadow* – also pictured in Figure 2.2). David and Bork (2024) delve deeper into this notion of automation and identify a *Human-Actuated Digital Twin* (*i.e.,* a *Digital Generator* that requires the human to take the actions) and a *Human-Supervised Digital Twin* (*i.e.,* a DT that needs the human's approval when steering the physical object).

While there are a plethora of different DT architectures (each of which focusing on different aspects of the system), Tao and Zhang (2017)'s 5D architecture reappears the most often. It is pictured in Figure 2.3. As the name describes, the architecture contains five main components:

1. the *Physical Shop-Floor* (*i.e.,* the AO);
2. the *Virtual Shop-Floor* (*i.e.,* the TO);
3. the *Shop-Floor Service System* (*i.e.,* the set of services or processes that compute, visualize, yield, and store information about both AO and TO);
4. the *Shop-Floor Digital Twin Data* (*i.e.,* an active database to maintain the current execution of the overall system); and
5. all connections between these components.

In essence, this architecture highlights and separates the key components in a Twinning system.

Other architectures mainly use the same separation of concerns. Madni et al. (2019) also separates the architecture in a *Digital Twin* (in this context, the TO is meant), a *Physical Twin* (*i.e.,* the AO), some knowledge base as sole source of truth, and a set of so-called services that are executed on the Twinning system. A similar separation is done by Llopis et al. (2023), where a *data lake* is located between the AO and the TO. The set of services is separated by the authors in *service components* (*i.e.,* components that add functionality) and *analysis components* (*i.e.,* components that do some analysis). Lutters and Damgrave (2019) identify the *Digital System Reference*, showing how DTs enable the creation of smart products.

Other sources focus on the details of the TO itself. For instance, Bibow et al. (2020), Bolender et al. (2021), and Tekinerdogan and Verdouw (2020) show the behaviour of

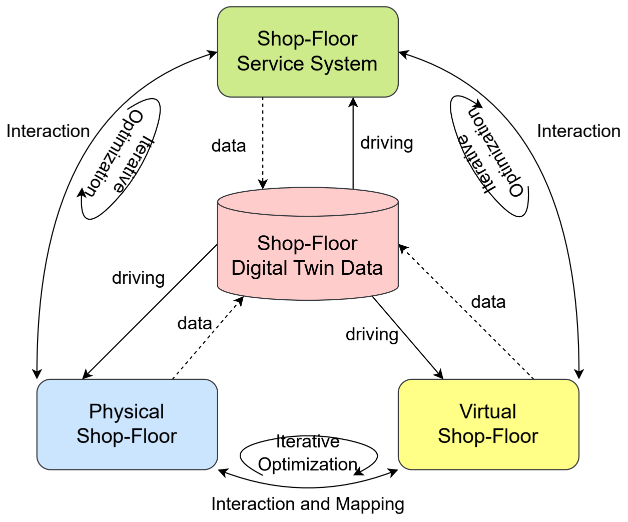


Figure 2.3: 5D architecture for Twinning (Tao & Zhang, 2017).

the TO as a MAPE-k loop. In Kibira et al. (2021), a separation of the entities inside the TO is done, yielding a set of services or processes that need to be active for a valid TO execution.

Notice that the architecture outline coincides with the identification of what exactly a *Digital Twin* is. Bibow et al. (2020), Bolender et al. (2021), and Kibira et al. (2021) have identified the *Digital Twin* as a synonym for the TO, as opposed to the whole system (which is the definition used for this work). In fact, at RWTH Aachen, a different DT definition is used. For them, the DT is the set of models of the system, a set of *Digital Shadows, and* the set of services. They define a *Digital Shadow* as a "*set of contextual data traces and their aggregation and abstraction collected for a specific purpose with respect to an original system*" (Rumpe, 2021). In short, they identify a DT as being the full set of models, their execution traces, and what happens with the models. A part of this relates to our usage for TO. Notice as well the relationship with *cognitive twins* (Lu et al., 2020) (*i.e.,* twins where all extraneous information about the system is also stored).

2.1.2 Asset Administration Shell

The RAMI 4.0 proposed the Asset Administration Shell (AAS) as a concept (Hankel & Rexroth, 2015). Nowadays, the AAS is widely recognized as a foundation for interoperability (Ferko et al., 2024; Quadrini et al., 2023). In order to support Industry 4.0, the AAS was designed as a standardized representation of an asset (Boss et al., 2020). It provides an interoperable solution for capturing the essential information of assets (Ye et al., 2021).

The AAS is based on the notion that the data, models and meta-information encloses an asset like a shell. It consists of a header and a body. The header describes the identification of the asset, and the body consists of a number of submodels. Each submodel represents a different aspect of the concerned system (*i.e.,* a different view). These can be CAD models, functional descriptions, algorithms. . . A complete description of the AAS metamodel can be found in Bader et al. (2022).

Note that this idea of the shell is highly similar to the cognitive twin, as well as the *Digital Shadow* as defined by Rumpe (2021). The set of submodels in the body highly resonates with the idea behind MPM.

In Oakes et al. (2021), a mapping is provided for DTs onto the AAS. Similarly, Ferko et al. (2024) shows how DTs can benefit from the AAS. Cavalieri and Salafia (2020) discusses how OPC UA fits with the concept of the AAS to provide an interoperable solution for Industry 4.0, a sentiment confirmed by Quadrini et al. (2023). Kannoth et al. (2021) shows how the AAS can be used practically through the Eclipse BaSyx middleware[2].

2.1.3 Existing Tools and Frameworks

Due to its popularity, it stands to reason that there are a lot of tools and frameworks to enable the creation of DTs. In fact, Hassan and Aggarwal (2023) indicates that most of the literature is focused around creating these frameworks, often omitting the practical implementation.

The Eclipse ecosystem provides multiple tools (*i.e.*, Eclipse Hono[3], Eclipse Ditto[4], Eclipse Vorto[5], Eclipse BaSyx. . .) that work in conjunction when used for DTs. Siemens also provides such a set of tools (*e.g.*, Siemens Xcelerator[6], Siemens Insights Hub[7]. . .). Alternatively, XMPro[8], FlexSim[9], and many other tools can be used to achieve the same.

Steindl et al. (2020) analysed the combination of AI techniques and DTs. Other papers have focused on exact DT characteristics (El Saddik, 2018; Oakes et al., 2021), such that tools can be implemented better.

This work will not propose a new tool. However, it will present a framework for Twinning, based on the literature. It will make use of the identified characteristics, purposes, architectures and deployment strategies.

2.2 Multi-Paradigm Modelling

In our current day and age, the complexity of software-intensive systems (Giese, 2006) keeps increasing, up to the level of CPSs. CPSs emerge from the networking of multi-physical (mechanical, electrical, hydraulic, bio-chemical, ...) and computational (control, signal processing, logical inference, planning, ...) processes, often interacting with a highly uncertain environment, including human actors, in a socio-economic context (Carreira et al., 2020). It is the heterogeneity in views, components, abstractions and

[2]https://eclipse.dev/basyx/
[3]https://eclipse.dev/hono/
[4]https://eclipse.dev/ditto/
[5]https://eclipse.dev/vorto/
[6]https://xcelerator.siemens.com/global/en.html
[7]https://plm.sw.siemens.com/en-US/insights-hub/
[8]https://xmpro.com/
[9]https://www.flexsim.com/

their many inter-relationships, in combination with the many stakeholders from different domains, collaboratively designing such systems, that contribute to their complexity. For instance, planes run on approximately 6 million lines of code, whereas modern cars have over 100 million lines of code (Charette, 2009)[10]. This code steers the overall execution of the system, receives and handles sensor information, and sends control signals to engines and other actuators, whilst at the same time maintaining the legal (and physical) safety requirements and playing the music you selected.

All this code is too complex and too vast to maintain. M&S is an important enabler in the construction of such complex systems. MBSE focuses on modelling within the scope of Systems Engineering, where models are seen as first-class concepts in the development process. Starting from an initial set of goals, Systems Engineering carries out a number of *activities* (manual or automated) to achieve these goals. The combination of these activities is called the *workflow* or *life-cycle*. It can be explicitly modelled in a Process Model (PM), in an appropriate modelling language such as UML Activity Diagrams (Object Management Group, 2017). Russell et al. (2016) discuss a set of workflow patterns which are useful when designing a PM.

While each model corresponds to a formalism, there is no "*one formalism to rule them all*". In essence, each formalism has its own advantages and caveats, and a selection of the most appropriate ones will be advantageous for any project. This is the idea behind MPM: model every aspect and part of a system explicitly using the most appropriate formalisms, most appropriate views, most appropriate tools. . . (Mosterman & Vangheluwe, 2004).

To help realize MPM, Mustafiz et al. (2012) introduced the Formalism Transformation Graph and Process Model (FTG+PM) – a framework for MBSE in which a PM is combined with a FTG, *i.e.*, a "map" of all artifact types (also known as meta-models) and activity types (in the form of contracts) and how they are related. Lúcio et al. (2012) show how the FTG+PM can be used to reason about all models and modelling languages required for the valid execution of a power window.

In its 2035 vision for Systems Engineering, the International Council on Systems Engineering (INCOSE 2021) describes the critical role MBSE plays in tackling increasing system complexity, mainly when supported by toolchains (Ma et al., 2022). It also states the importance of integrated analysis in a broad set of system domains. We believe that MPM can help realize INCOSE's 2035 vision. DTs are already used by system engineers to create complex systems, but they can benefit from guidance in the choice of modelling formalisms and workflows, as provided by MPM.

2.2.1 Language Combinations

Given that we now know to use multiple formalisms to model a certain system, the main question that remains is how these can be combined and executed such that their *semantics* remains the same. Multiple solutions for this problem exist, but this section will focus on the three that were used during this research. For instance, Denil et al. (2012) shows how numerous formalisms can be used and combined for the valid execution of a power window in a car.

[10]These numbers from 2009 might be dwarfed by the current systems.

2.2.1.1 Formalism Transformations and Translations

The most common approach in combining languages is via a model transformation onto the same base language (Czarnecki, Helsen, et al., 2003). A few common base languages to translate to are Discrete EVent system Specification (DEVS), SysML, Timed Finite State Automata (TFSA), Petri-Nets, Ordinary Differential Equations (ODEs), but others are used as well.

Borland and Vangheluwe (2003) and Shaikh and Vangheluwe (2011) have attempted to translate Statecharts onto DEVS. In Paredis et al. (2020), we have shown a similar approach for General Purpose Simulation System (GPSS). Sanz et al. (2007) provided a mapping for SIMAN/Arena blocks by encoding the logic in Modelica. In order to provide discrete event simulation for SysML blocks, a translation onto DEVS was introduced in Kapos et al. (2014). Commonly, safety analysis is done via translating onto Petri-Nets (Choppy et al., 2011; Meyers et al., 2014).

A lot of the discussed translations focus on a mapping onto the DEVS formalism. In Vangheluwe (2000), Figure 2.4 was introduced to show that DEVS can be used as a "common denominator" and assembly language to which many other discrete-event modelling languages can be mapped. This figure is considered a FTG, but more accurately a *Formalism Relationship Graph*. It represents which discrete-event translations (hypothetically) exist. As can be seen, DEVS is the basis language that can be used for translations.

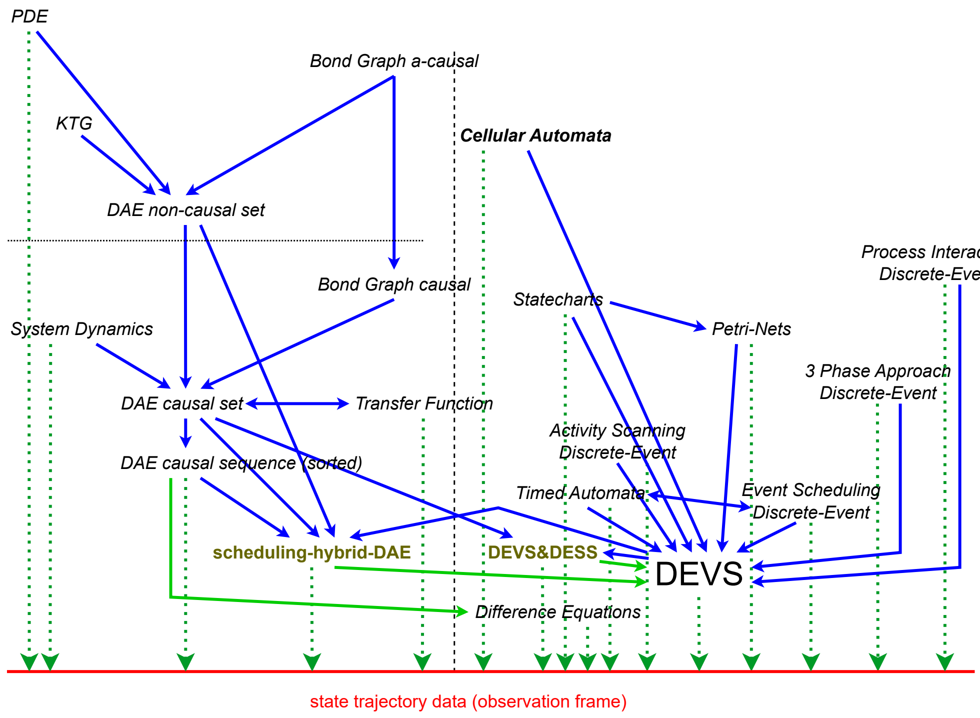


Figure 2.4: The original FTG, adapted from Vangheluwe (2000).

2.2.1.2 Embedding

Another way of combining multiple languages is via *embedding*, or another *hybrid* combination of the simulators. Bergero and Kofman (2011) shows how hybrid system modelling can be done in DEVS and D'Abreu and Wainer (2005) introduces M/CD++ for modelling and simulating continuous and hybrid systems, based on Modelica and DEVS. We have embedded CBD in TFSA (and DEVS) in Paredis, Denil, and Vangheluwe (2021).

2.2.1.3 Co-Simulation

Instead of trying to force a formalism onto another, or trying to merge the semantics, the idea behind co-simulation is that all formalisms are standalone and will have their own simulator. For each simulator, the interface is known to a common *Orchestrator*. This Orchestrator will get the task of instructing the simulators in the right order and at the right times, whilst also needing to transfer all inputs and outputs correctly.

Camus et al. (2018) developed MECSYCO, a co-simulation middleware that also provides a mapping onto DEVS (using co-simulation, instead of a hybrid or translational approach). The most common tool for co-simulation is using Functional Mock-Up Units (FMUs) (see also Section 2.3.3).

2.2.2 FTG+PM

The Formalism Transformation Graph and Process Model (FTG+PM) consists of an Formalism Transformation Graph (FTG) and a Process Model (PM) (Mustafiz et al., 2012). The FTG is a hypergraph that links languages by defining the relations and transformations between them. The ways in which models are related and combined through both human and computer-based activities is made explicit using PMs (or workflow models). The PM is a UML Activity Diagram which describes the ordering of activities (control flow) as well as their input and output artifacts (data flow). Individual activities are typed by transformations in the FTG.

An example PM is shown in Figure 2.5. Each activity is represented with a rountangle and it describes the activity name and its type. Additionally, there can be inputs and outputs for control flow (as indicated with the blue triangles), or data flow (green triangles). Bold blue arrows are be used to indicate the direction of the control flow, and slim green arrows for the data flow. Green rectangles denote the artifacts that are consumed and produced by activities.

An activity may be hierarchically refined, which will be denoted with a hierarchy symbol (🖧) in the top right, as can be seen in the `decomp: SystemDecomposition` activity. When they're automated (*i.e.*, computer-based), we will give the activity a yellow background and we will add a gear symbol (⚙) in the top left.

In Paredis et al. (2022), an expansion to the FTG+PM was presented. For each model used in the FTG, a corresponding meta-model (*i.e.*, a linguistic type model) should also be present. These meta-models also determine how to Create, Read, Update, or Delete

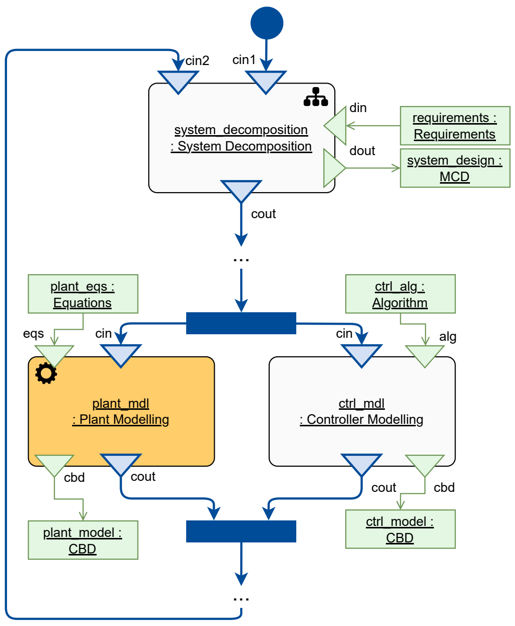


Figure 2.5: An example PM.

instances and artifacts. On the other hand, a Process Trace should be added to the FTG+PM. The Process Trace shows a single execution of the PM, including all timing information and parallelization specifics. This Process Trace contains all information about past executed activities and (versions of) generated artifacts. A Process Trace can be traced back to the PM it is an enactment of. Furthermore, a Process Trace model is append-only: added elements become immutable and as such provide an archival record that can be analysed or mined. It will therefore always give the same analysis results, even if the PM evolves (also in an append-only fashion).

Another extension to the FTG+PM framework is the ability to point to stored files for meta-models, versioned artifacts, service versions for executed activities, and physical locations for real-world artifacts (such as a built robot). This will be referred to as (physical) Storage/Service/Real-World Artifacts. Explicitly storing this information next to the Process Trace allows a clear, permanent introspection of all required components, such that the decisions made can be revisited and recalled later.

As a whole, this FTG+PM expansion allows for a lot of querying possibilities and introspection in the overall system creation or maintenance. On a high-level, it is possible to see relationships between models, but on a lower level, fine-grained traceability can be executed to answer questions such as *"where does this column in my table come from?"*

2.3 Modelling Formalisms

To show the applicability of this research, multiple modelling formalisms will be used. These are not new innovations for this research, but rather a selection based upon the

MPM methodology, as applied to this work. Note that other formalisms can also be used to solve the presented problems in later chapters. As will be discussed in Chapter 5, the choice of formalism is independent of the developed foundation and is, in fact, another point of variability. This section focuses on Feature Trees, Causal-Block Diagrams (CBDs), FMUs and DEVS as the basic formalisms in this thesis.

2.3.1 Feature Trees

It is common for multiple *variants* of a product to exist. These variants share some *common* 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*.

Feature Modelling (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 to the right[11], 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, custom Domain-Specific Languages may be used (Czarnecki, 2004).

In Figure 2.6, a small, example Feature Tree is shown in order to explain the syntax used in this thesis. The Feature Tree illustrates 4 different ways of visualizing information: *2D Animation*, *Live Plots*, *3D Animation*, and *Augmented Reality (AR)*/*Virtual Reality (VR)*/*Mixed Reality (XR)*.

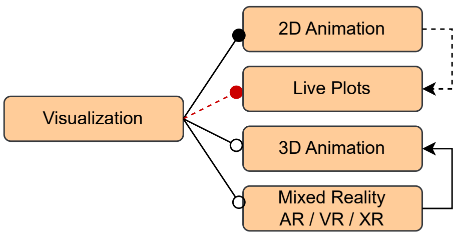


Figure 2.6: Example Feature Tree.

In this example, we assume *2D Animation* is a mandatory aspect if we want to visualize. This is denoted with the filled circle. Both *3D Animation* and *AR*/*VR*/*XR* are optional visualizations, as denoted with the open circle. However, *AR*/*VR*/*XR* implies the need for *3D Animation*, which is denoted with the full arrow. Throughout this thesis, the decision on whether or not an aspect is mandatory or optional is the result of a personal

[11]Or downwards, if the tree is oriented differently.

(and thus biased) analysis of the literature. These therefore can be changed, and are highly prone to errors.

The red dotted line with a filled circle is a custom notation for this work and indicates that, ideally, an aspect (like *Live Plots*) is mandatory, but most of the literature ignores their importance. In other words: based on the literature, these should be marked optional, but when thinking critically, these should be mandatory. The striped arrow identifies that *2D Animation* might rely on *Live Plots* in some instances.

2.3.2 Causal-Block Diagrams

One of the simplest ways to represent a system of multiple interacting components is to use boxes and arrows. They can be used for many purposes; from schematic system overview to electrical circuit modelling (Åström et al., 1998). To tackle complexity, blocks can be nested in hierarchical structures. One specialization of such structures are Causal-Block Diagrams (CBDs) (Gomes et al., 2020).

The arrows denote signals and the boxes represent mathematical operations over the input signals, producing an output signal. The denotational semantics of a CBD with continuous time base ($\mathbb{R}$) is the corresponding set of Algebraic, Ordinary Differential, or Differential Algebraic Equations (and ultimately, the signals, continuous functions of time, that satisfy the constraints imposed by these equations). As such, ODEs and CBDs can be transformed into each other without loss of information.

In their simplest form, Algebraic CBDs consist of blocks that represent algebraic computations. There is no immediate notion of time and simple systems of algebraic equations can be represented.

By introducing a discrete notion of time, Discrete-Time CBDs can represent time-varying dynamics. Their simulation computes, at each discrete time, the value of the output of each block, based on the values of its inputs. The order in which blocks are traversed is determined on their dependencies. All blocks' outputs are computed at the same (simulated) time as their inputs, with the exception of the `Delay Block`, which produces on its output the value of its input at the previous discrete time.

The semantics of Continuous-Time CBD models is given by ODEs. Therefore, they are widely used to model the behaviour of physical systems. Continuous-Time CBDs are solved numerically through time-discretization which yields an approximation in the form of a Discrete-Time CBD. In the limit, for the discretization time-step Δt tending to 0, the approximation converges to the exact solution of the ODEs.

2.3.2.1 CBD Simulation

To simulate a discretized CBD model, two nested loops are used. In the *outer loop*, a stepping variable k is started at 0 and increased for every iteration of the loop. In-between k and $k+1$, a time-delay $h_k \in \mathbb{R}$ is used to advance the simulated time. The simulation finishes when some termination condition (a function of current simulation time $\sum_{i=0}^{k} k_i$ and the state variables) becomes true. When h_k is independent of k, the simulation is said to have a *fixed step size*. The smaller h_k, the better the numerical simulation results will approximate the continuous solution. *Adaptive step size* algorithms vary h_k throughout the simulation, to keep the stepwize approximation error within given bounds. A commonly used adaptive step size discretization is Runge-Kutta-Fehlberg of 4th and 5th order (RKF45). The difference between the fourth and fifth order approximations is used as an estimate for the stepwize error, such that h_k can be varied.

The *inner loop* focuses on the computation of an output signal for a given iteration k. Here, a schedule is defined, determining the order in which the blocks of the model will be traversed. This schedule is typically based on the *topological order* of the *dependency graph* of the model, making sure a block is only visited once all its dependencies have been computed. Each block has their own definition of what this "computation" may entail. For instance, the adder block will output the sum of all its inputs.

An *algebraic loop* occurs when there is a cycle in the dependency graph. The *strong components*, *i.e.*, all blocks belonging the algebraic loop, should therefore be computed as a whole. It's possible to distinguish two methods for solving a strong component:

1. The blocks are converted to a system of equations, such that a (non-)linear solver is able to find a solution for the system. For instance, when these equations are all linear, the *Gauss-Jordan Elimination* algorithm can be used.
2. *Tearing* the algebraic loop. Some connections in the strong component are broken and replaced by initial guesses for the values that are supposed to be signaled over the connection. When this torn loop has found a solution, the initial guesses are replaced by this solution and the loop is computed again. This small algorithm is executed until convergence.

In Figure 2.7, a simple CBD model is shown, representing the algebraic equation $x = 7x-3$. The inner loop will compute 7, 3 and the negation of 3 ($= -3$), before arriving at the algebraic loop containing the Product block and the Adder block. As the equation is linear, Gauss-Jordan Elimination can be used to find $x = 0.5$.

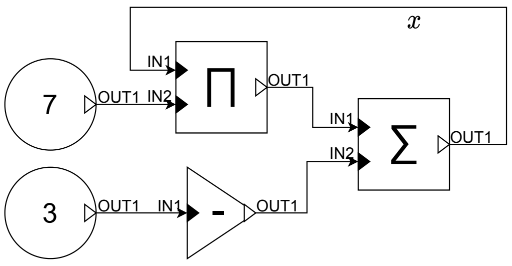


Figure 2.7: Simple CBD model with an algebraic loop.

2.3.3 Functional Mock-Up Units

Multiple formalisms may need to be combined or simulated at the same time. One way of doing this is by using co-simulation, in which an orchestrator manages interaction between multiple formalisms. The industrial standard for co-simulation is the Functional Mock-Up Interface (FMI). A single instance is called a Functional Mock-Up Unit (FMU).

An FMU is a simple ZIP archive, which consists of the following files and folders:

modelDescription.xml An XML document that describes all the exposed components of the FMU, as well as the default simulation setup.

sources A folder with an optional set of source files, written in C. They narrowly follow the FMI specification to ensure a valid execution. If desired, they can be omitted from the FMU, if there are binaries present instead.

binaries A folder with the set of executables of the FMU. Can be bundled for any operating system, so sharing FMUs can happen easily.

Other files or folders may be present, but are not of core importance in an FMU. The interface allows for *co-simulation* (used for the coupling of simulation tools, and the coupling of subsystem models), *model exchange* (exposes an ODE to an external solver of an importer; the integration algorithm is responsible for advancing time, setting states, handling events), or *scheduled execution*[12] (allows coupling several FMUs with one, external scheduler).

2.3.4 DEVS

Discrete EVent system Specification (DEVS) (Zeigler et al., 2018) is a modular discrete-event formalism introduced by Bernard Zeigler in the '70s. It consists of basic components, called *atomic DEVS*, which have the following structure: $\langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta\rangle$. Here, the *input set* X denotes the set of admissible input events of the model and *output set* Y denotes the same for the output events. When properly structured, X and Y describe a collection of *ports* through which a model sends or receives events. S is the *sequential state set* and indicates the set of sequential states of the model. The *internal transition function* $\delta_{int} : S \rightarrow S$ specifies the state the system transitions to after ta time, unless interrupted before. The *time advance function* $ta : S \rightarrow \mathbb{R}^{+}_{0,+\infty}$ specifies how long the system remains in a state, before δ_{int} takes it to the next sequential state. Prior to the application of δ_{int}, the *output function* $\lambda : S \rightarrow Y$ is called, specifying the output event that is to be produced. The *external transition function* $\delta_{ext} : Q \times X \rightarrow S$, where $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is called when an *external input* (*i.e.*, an interrupt) arrives.

Multiple DEVS models can be combined into a network structure. Such a *coupled DEVS* is fully characterized by $\Delta = \langle X_\Delta, Y_\Delta\, D, M_i, I_i, Z_{i,j}, select\rangle$. Similar to atomic DEVS, X_Δ and Y_Δ denote the in- and output sets of the network. The *set of component references* is denoted by D, where M_i identifies the DEVS model component $i, \forall i \in D$. I_i is the *set of influencees* of component $i, \forall i \in D \cup \{\Delta\}$ (with $i \notin I_i$). I fully specifies the connection topology of a coupled model. The *transfer function* $Z_{i,j}$ is defined $\forall i \in D \cup \{\Delta\}, j \in I_i$ as $Z_{\Delta,j}$: $X_\Delta \rightarrow X_j$; $Z_{i,\Delta}$: $Y_i \rightarrow Y_\Delta$; and $Z_{i,j}$: $Y_i \rightarrow X_j$. It serves to translate events as they travel along network connections. Finally, the *select function* ($select : 2^D \rightarrow D$) allows the selection of a single component $i \in D$ when multiple models in the network are

[12]New in FMI version 3.

simultaneously *imminent* (*i.e.,* they are due to transition at the same time). As DEVS is closed under coupling, coupled DEVS models may be nested to any arbitrary depth.

An abstract simulator for DEVS is described in Muzy and Nutaro (2005). Tools such as PythonPDEVS (Van Tendeloo & Vangheluwe, 2015) and adevs (Nutaro, 2015) implement these semantics. A simulation step in the algorithm for the classic version of DEVS, as implemented by these tools, can be summarized as follows:

1. compute the set of atomic DEVS models whose internal transitions are scheduled to fire (imminent components);
2. select one imminent component with the *select* tie-breaking function;
3. execute the imminent component's output function, generating an output event;
4. route events from sending components to receiving components;
5. determine the type of transition to execute for each atomic DEVS model, depending on it being imminent or receiving input;
6. execute, in parallel, all enabled internal and external transition functions;
7. compute, for each atomic DEVS model, the time of its next internal transition.

Autodesk imagined DesignDEVS (Goldstein et al., 2016), which would allow for modelling in DEVS for non-experts in simulation formalisms. The presented Graphical User Interface and approaches ensure that programmers can focus on the behaviour, instead of the formalism specifics (Maleki et al., 2015). Unfortunately, this tool was never actually realized.

The applicability of DEVS, the vast number of possibilities, the presented tooling, and the already existing set of transformations of other modelling languages onto DEVS, makes it stand out as a useful "assemby" language for other formalisms. Vangheluwe (2000) even showed that DEVS can be used as a common denominator for both continuous-time and discrete-event languages.

2.3.5 Deployment Diagrams

Deployment Diagrams model the physical architecture of a system. They illustrate how software is deployed on hardware components, capturing the relationships between software artifacts, such as applications, Operating System, or components, and the physical resources they run on, such as servers, nodes, or devices. Additionally, the communication paths between them are also considered.

They are commonly represented with hierarchical boxes, layered on top of each other. The connections are represented with solid lines and striped arrows. The solid lines identify which components are connected together (either wired, wireless, or due to being on the same device). The arrows indicate which components communicate to each other.

2.4 Communication Protocols

An important aspect in highly complex CPSs is the inter-communication between components. For DTs, this will also be very important. This section will describe the most prominent communication protocols used within the context of DTs.

2.4.1 MQTT

MQTT[13] is an OASIS standard messaging protocol, specifically for IoT systems. It has a publish-subscribe architecture that allows communication using minimal network bandwidth.

The network consists of a series of MQTT clients that are connected through a message broker. When a client publishes a message to a specific topic, the broker ensures that all clients that are listening to that topic receive the message. The receiving clients are considered *subscribers* of that topic. Each client has its own unique name and must be connected to a broker to publish or receive data. Whenever a publishing client disconnects due to a network outage, a *last will message* will be used to notify all subscribers.

Eclipse Mosquitto[14] or EMQX[15] will be used as a (localhost) message broker throughout this research.

2.4.2 DDS

The Data Distribution Service (DDS)[16] is a real-time middleware for high-performance machine-to-machine communication, using a publish-subscribe pattern. It is used in a lot of CPSs, robotics, and IoT systems.

In a network, there are nodes that publish *samples* over custom *topics*. Other nodes are subscribes to these topics, and they receive these samples. This happens decentralized: *i.e.*, peer-to-peer, without the need for a message broker.

Whereas both MQTT and DDS are publish-subscribe standards, they are not the same. MQTT is optimized for data centrality and DDS is optimized for distributed processing.

2.4.3 OPC UA

In the manufacturing domain, OPC UA[17] is a commonly used protocol that allows for cross-platform data exchange. It is structured like a service-oriented architecture. Note that OPC UA is a standard, and not a middleware (like DDS or MQTT), nor a protocol

[13]https://mqtt.org/
[14]https://mosquitto.org/
[15]https://www.emqx.com/en
[16]https://www.dds-foundation.org/
[17]https://opcfoundation.org/about/opc-technologies/opc-ua/

(like TCP/IP). This means that OPC UA can function on top of TCP/IP *and* MQTT (among others).

At the core, there is the client-server model. The server manages so-called *objects* that are hierarchically created in a tree-like structure. When the object represents a variable, a value is associated to the object. Clients can read or write a value to specific objects.

Besides this client-server model, there is also a publish-subscribe alternative. This has been optimized for many-to-many communications and may be inefficient for peer-to-peer communication.

The RAMI 4.0 (Hankel & Rexroth, 2015) recommends to use OPC UA as a communication layer. As a consequence, products are only considered Industry 4.0-enabled if they are OPC UA capable (Ho, 2023).

2.4.4 ROS 2

The Robot Operating System (ROS)[18] is more than just a communication protocol. It is a set of (open-source) software libraries and tools that allow for a good communication between different robotic machines, ensuring a seemless coordination between sensors, actuators and software. The current ROS 2 distro is *Iron Irwini*, and the latest long-term support *Jazzy Jalisco*[19]. Yet, for this research the *Humble Hawksbill* or even the *Foxy Fitzroy* distro were used.

Behind the scenes, ROS 2 uses DDS as its middleware. Hence, all communication happens via a decentralized publish-susbcribe protocol. The messages that are communicated can be recorded using ROS bag files or logs for easy testing, training and quality assurance.

While ROS has a lot to offer, within this work, we will mainly be using the communication capabilities. As Open Robotics describes in their introduction video for ROS: "*With ROS, you can work with a simulated robot, instead of the real thing.*" This clearly paves the way towards DTs.

[18]https://www.ros.org/
[19]This is the case at the time of writing – January 2025.

"To be or not to be – that's the question."
– William Shakespeare, Hamlet

Chapter 3

Goals: To Twin or Not To Twin

Given the overwhelming amount of literature on DTs, researchers have started to study and chart the purposes and methodologies used in the engineering of Twins. Dalibor et al. (2022) presents a large systematic mapping study and in Van der Valk et al. (2020), a taxonomy of DTs is constructed, based on 8 different dimensions. Jones et al. (2020) and Kritzinger et al. (2018) mainly focus on the specific kinds of architectures they have encountered. Wanasinghe et al. (2020) made a set of business-specific classifications; and Lim et al. (2020) and Minerva et al. (2020) mainly discuss the kinds of technology and deployment.

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 3.1.

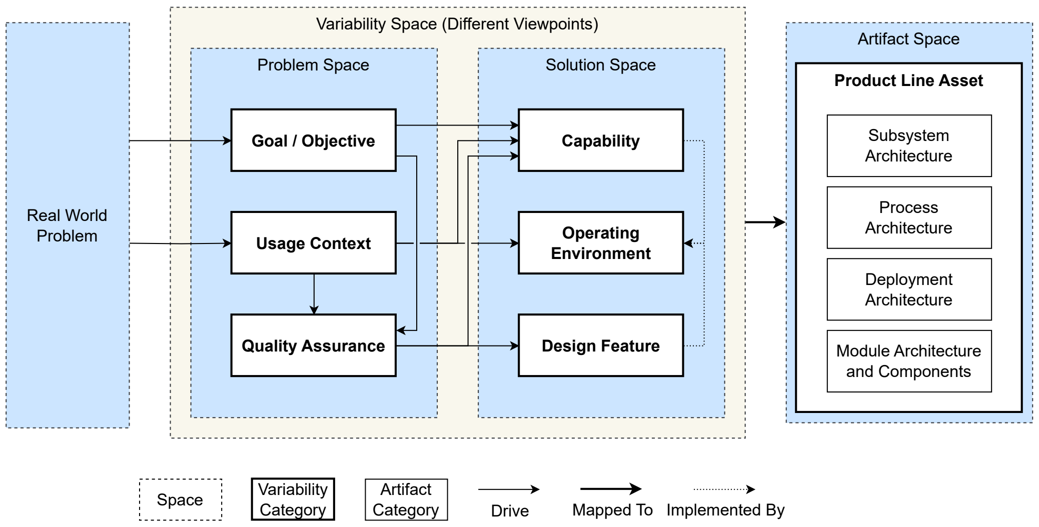


Figure 3.1: Variability Modelling Space, adapted from Kang and Lee (2013).

The variability in the Problem Space is broken down into variability of *Goals and Objectives*, the *Usage Context* and the *Quality Assurance* of the products. Variability in the *Solution Space* breaks down into variability in the *Capabilities*, the *Operating Environment* and the *Design Features* 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).

Using Dalibor et al. (2022)'s categorization as a starting point, also incorporating insights from other related research (including our own), a new, non-exhaustive, detailed set of aspects was constructed that relate to any number of Twinning systems. This set is categorized according to Kang and Lee (2013)'s *Problem Space* separation and presented throughout this chapter using Feature Trees.

3.1 Properties of Interest

Before delving into the exact categorization, it is important to remark the importance of specific PoIs that exist in parallel to the detailed set of aspects presented later. These PoIs are attributes (or descriptors) of an artifact that are either logical (*i.e.,* a car has wheels) or numerical (*i.e.,* a bicycle has 2 wheels). They can be computed (or derived) from other artifacts, or specified (*i.e.,* defined by a user) (Qamar & Paredis, 2012). These PoIs describe the specifics of the goals that will solve the problem.

For instance, is the focus on the energy efficiency, or on accuracy? Is the focus on the power consumption, or the overall operating cost? This will also have an impact on how the Twin should be constructed.

3.2 Goals, Purposes, and Objectives

First, let us discuss the goals, objectives and purposes w.r.t. (a set of) PoIs. These are generally goals that focus on the SuS, also known as *digital twin capabilities* (Ali et al., 2025). In essence, the SuS is the essential, central component that is to be analysed. Figure 3.2 shows a non-exhaustive feature tree for this section.

3.2.1 Design

Many literature studies and taxonomies highlight that some twins are created *before* the actual SuS. This is either used for *Virtual Prototyping* (Poppe et al., 2019) to quickly analyse and verify a prototype version, or as a more iterative approach. This is called *Variation*

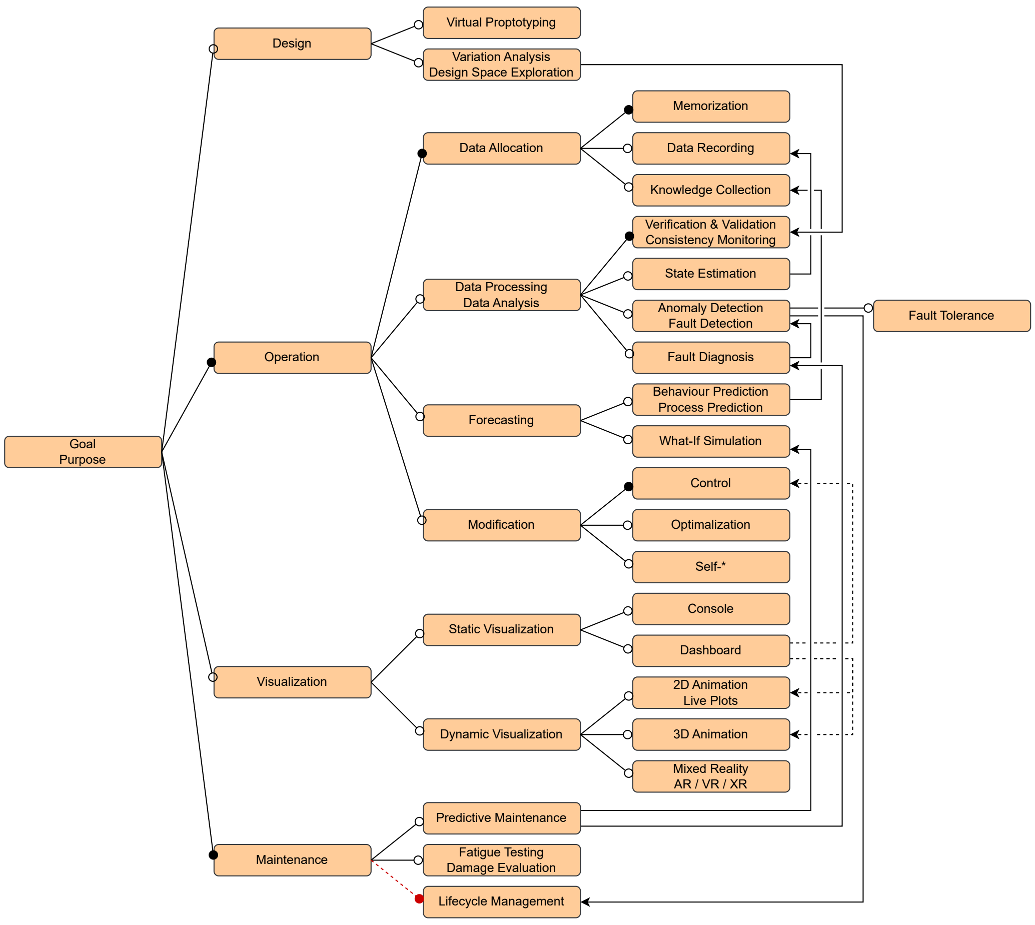


Figure 3.2: Feature Tree for the *Goals* of a Twin

Analysis (Wang et al., 2018) or *Design-Space Exploration* and it allows engineers to explore any number of variations to check their accuracy w.r.t. some PoI. While these techniques may help and enable twinning, in essence, they are normal MBSE techniques that apply to any SuS (and not just Twins).

3.2.2 Operation

Most purposes for creating a Twin have to do with the operation (*i.e.*, the at-runtime execution) of the overall system. We can consider a few sub-categories.

Data Allocation / Persistence In order for a Twin to work, data needs to be allocated from the AO during its lifetime. Some twins only use *memorization* (*e.g.*, only keeping track of the current runtime information) (Santillán Martínez, 2019; A. Sharma et al., 2022). Memorization should be present for any Twin, otherwise it cannot be kept equivalent to the SuS. The next step is to actually *record data* for a longer period of time (Brockhoff et al., 2021). This maintains historical data and is able to make decisions based on what happened in the past. According to Padovano et al. (2018), a Twin becomes "intelligent" as soon as we also maintain the context in which the data was obtained. This *knowledge collection* enables inferencing and reasoning on past decisions in order to optimize future decisions.

Data Processing / Analysis The data that was obtained from the AO can be processed and analysed in order to gain more insights into the system state (for both the AO and the TO). The most simplistic analysis is *validation*. Here, it is verified that the AO and the TO are accurate and represent the same kind of system (Grinshpun et al., 2016; P. Sharma et al., 2018). All models should be validly calibrated before their use, so in essence, validation should already be present in any twin. Based on historical data, we might also do *state estimation* in order to try and predict future situations (González et al., 2020).

Some twins monitor the data in order to identify things that may have gone wrong. The most famously used is *anomaly detection* (Verriet, 2019; Yacob et al., 2019), where we just identify a drift between the AO and the TO. Based on this drift, *fault diagnosis* can be started to identify the reason thereof (Gonzalez et al., 2019; Jain et al., 2020; Verriet, 2019). Here, it can be detected that the AO's *health* might have decreased (González et al., 2019), or that some *process parameters* are throwing a spanner in the works (Brockhoff et al., 2021).

Forecasting We can go to the next level by also doing *forecasting* (Gockel et al., 2012; Kosicka et al., 2017). A common approach here is *behaviour* or *process prediction* (Brockhoff et al., 2021). Mainly using knowledge about the models and current state, we will predict a future for the system. *What-If Simulation* is also a common technique that is used to both *explain* (Rivera et al., 2022) and *predict* (Flammini, 2021) situations in the system.

Modification According to Kritzinger et al. (2018), we can only call DTs "Twins" if there is a control signal from the TO to the AO. In essence, this is where the power of DTs comes from: having an external process update, *optimize* or *control* the AO. The modification could be used to solve *self-adaptation* (Bolender et al., 2021; Oreizy et al., 1999; Weyns et al., 2010), *self-control* (Graessler & Poehler, 2018), *self-healing* (X. Feng et al., 2023), *self-reconfiguration* (Müller et al., 2021), *self-learning* (Moya et al., 2020) . . .

3.2.3 Visualization

Most twins have a visual representation, such that a user is able to easily discern the current state of the system and make decisions based on this. While this visualization may be just that (*i.e., read-only*), there could also be a set of controls for a user to make decisions (*i.e., read-and-write*).

Console / Dashboard A console is a text-based tool that either logs the current state, or allows you to browse this state (Lutters & Damgrave, 2019). A dashboard (also known as a *Digital Cockpit* (Brockhoff et al., 2021)) is commonly a Graphical User Interface that provides the same information, but in a more user-friendly manner. Dashboards may include plots, figures, tables and potentially a runtime debugger of the TO.

Animation Many systems also show a graphical animation of the current state of the system. This may be in two dimensions (2D), using a plot (H. Feng et al., 2021), or a map (Paredis & Vangheluwe, 2021); or in three dimensions (3D) in a virtual world (Höpfner et al., 2021). 3D animations are commonly combined with AR, VR or XR approaches (El Saddik, 2018; Utzig et al., 2019).

3.2.4 Maintenance

A Twin *evolves* over time. Similar to real-world CPSs, there may be wear and tear, changing requirements, system updates and changing technologies. Hence, it is important for a Twin to be maintained throughout its lifetime. Additionally, a TO may be able to deduce all these aspects of the AO and act upon them.

Predictive Maintenance Based on vendor specifications, past experiments and future predictions, it is possible to predict when a system will fail. Verriet (2019) describes a smart lighting system for a building in which the twin can predict when the lights will break. When close to their end-of-life, it will automatically order a new light bulb, such that it can easily be maintained. Note that this example also shows that this maintenance process may not be fully automated.

Fatigue Testing / Damage Evaluation Instead of testing the sturdiness or wear and tear of a physical system, *fatigue testing* (Gomez-Escalonilla et al., 2020) or *damage evaluation* (Utzig et al., 2019) translate this analysis (and its required preventive measures) to the virtual world.

Lifecycle Management As is the case for any system, the AO has an individual lifecycle that might have consequences for the state of the TO and all analyses that are happening (Heithoff et al., 2023; Qin et al., 2022).

3.3 Quality Assurance

Another aspect to study about Twins is the quality they assure. This focuses on how good the Twinning system works. While most of this has to do with the deployment and implementation, focusing on this beforehand will result in a system that has a higher quality. Figure 3.3 shows a Feature Tree for this category.

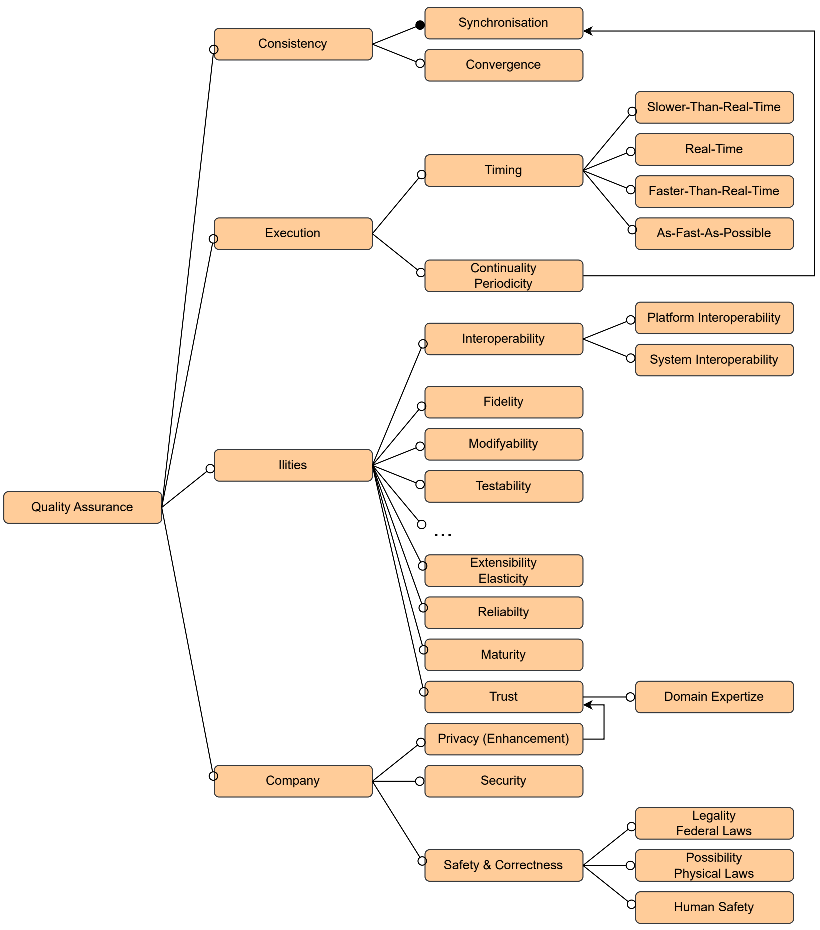


Figure 3.3: Feature Tree for the *Quality Assurance* of a Twin

3.3.1 Consistency

We need to verify that the AO and the TO are continually kept in sync. We can ensure this using *consistency monitoring* (Talkhestani et al., 2018) and by defining the allowed *deviation*.

Synchronization A TO needs to be continually and continuously be kept in sync with the AO. While we can (in a perfect world) assume that the AO and the TO are not drifting, we can commonly not incorporate all aspects that cause drift in the model. In these situations, we need to re-sync whenever there is some deviation. Alternatively, some technologies allow re-syncing every x time, making sure the AO and TO are kept up-to-date (Modoni et al., 2019). A downside of this approach is that this will require a lot of additional computations and that drift may not be detectable.

The AAS (Boss et al., 2020) also identifies information synchronization as an important aspect for Twinning.

Convergence Whereas *synchronization* mainly focuses on the drift between the AO and the TO, *convergence* does the opposite by identifying the closeness between the AO and the TO.

3.3.2 Execution

The *timing* of a Twinned system can be discussed in multiple dimensions. Either we can discuss the speed of execution of the TO (which can be *slower-than-real-time*, *real-time*, *faster-than-real-time*, or *as-fast-as-possible*) (Diakité & Traoré, 2023). Depending on which timing the TO needs, the accuracy and precision of the Twin will suffer. Some TOs cannot be executed in real-time (or faster), whereas others might not be able to slow down. In general, it is possible to state that the faster the updates can occur, the better the quality of the system becomes. Unfortunately, real-time simulation still comes with a lot of caveats (Lugaresi & Matta, 2018), especially when the system configuration changes during runtime.

Alternatively, we can also discuss the timing of evolution of the overall system. Biological systems will evolve slowly, but AI systems usually evolve fast.

By definition, the AO and the TO should *continually* update each other with information. The *periodicity* of this update will also define the overall quality of the Twinned system, albeit highly depending on the exact context. For instance, Twins for historical buildings, analysing the masonry decay (Angjeliu et al., 2020; Loverdos & Sarhosis, 2023) will have a much higher periodicity (potentially days or weeks) as opposed to robotics (Bibow et al., 2020; Walravens et al., 2022) (likely seconds, or milliseconds).

3.3.3 Ilities

The *ilities* are defined by de Weck et al. (2011) as 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 use. These properties are not the primary functional requirements of a system’s performance, but they typically involve wider system impacts. These ilities usually do not include factors that are always present in a system. Important ilities that should be discussed (and analysed) within the context of Twinning are (but not limited to):

Interoperability Describes how the Twinning System can collaborate with other existing systems and how it may be used in a larger whole (Gross, 1999). In other words, it is the “*degree to which two or more systems, products or components can exchange information and use the information that has been exchanged*” (David et al., 2025). Traoré (2023) describes different kinds of interoperability and how to deal with them. David et al. (2025) highlights the importance of interoperability in Twinning.

The AAS (Boss et al., 2020) identifies interoperability between different suppliers, manufactures and machine builders as a key concern.

Fidelity There is no good consensus on what *fidelity* actually means. Most sources use it as a synonym for *accuracy*. Yet, it is also used as "*level of polish and preparedness of a software product*" (The Refinery, 2016), "*number of parameters transferred between the physical and virtual entities, their accuracy, and their level of abstraction*" (Jones et al., 2020), "*transmission performance*" (Shi et al., 2021), "*faithfulness*" (Oakes et al., 2023; Rakove, 1996), or a "*measure of realism*" (Gross, 1999). For the purposes of this research, we will assume fidelity is equivalent to *structural validity* (*i.e.,* architectural correspondence / morphism between the model and the SuS) (Qudrat-Ullah & Seong, 2010).

Testability Carnap (1936) defines testablity as "*whether or not we know a method for testing*". Thus, we can ask ourselves if we can stress-test a system or if it can be used to run any number of scenarios. The main aspect here is to ensure there exists a method for testing the system within its production environment. Because the AO and the TO need to be valid and in sync, the TO can be used to test the AO beforehand (H. Zhang et al., 2023).

Auditablity and Traceability Auditing might require a detailed description of what happened in the system. It might be required when errors occurred, or to check its safety (see later). For this, we need to maintain traces of every part of the execution and keep track of its traceability (which can be course-grained, or fine-grained (Paredis et al., 2022)).

Extensibility / Expansibility / Scalability / Elasticity / Modifyability How many components can be added? Is the twinned system *plug-and-play*? Can we enlarge the overall system for a broader use-case? Will the system follow the current required demands? Do we upscale or downscale online (*i.e.,* at runtime), or offline?

Ferko et al. (2023) tries to increase DT standardisation by separating a Twin in its functional components. Um et al. (2017) shows how the AO can benefit from a modular decomposition. Similarly, Monteiro et al. (2023) focuses on a scalable AO that the TO has to accommodate for.

Reliability / Stability Making sure that the system's execution can be trusted. For instance, having a large mean time to failure (Maxwell & Corn, 1978). Additionally, one can analyse how error-prone a system is. Techniques like *Fault Injection* (Markwirth et al., 2021; H. Zhang et al., 2023) will help to verify this.

Maturity Which Technology Readiness Level level can be assigned to the created system (Dalibor et al., 2022)?

Trust Is the person who built the twinned system to be trusted? Is it a domain expert? Is it okay to add the system into a larger ecosystem of technologies? Can the constructed system be packaged and distributed to the outside-world? Can the TO trust the input communications from the AO (El Saddik, 2018)?

3.3.4 Company

A company, or higher-up management level might also introduce custom requirements for the Twinning system.

3.3.4.1 Privacy Enhancement

Companies are commonly really interested in their Intellectual Property (IP), hence many techniques exist to ensure *IP protection*. A common one is the usage of FMUs[1]. Ideally, their system should be considered a black-box that can be used and interacted with, without knowing the actual internal implementation(s). It stands to reason that a company which builds a Twin wants to keep the internal workings secret for the outside world. Furthermore, if this Twin is used, you ideally don't want any hacker or malicious usage that could break a volatile system.

3.3.4.2 Security

With the ever-increasing usage of Twinning, it is important to ensure they withstand any number of (cyber-)attacks by malicious parties (Balta et al., 2021).

3.3.4.3 Safety / Correctness

When building a system, sometimes there are conditions that should be considered when building the TO, whilst not being theoretically required to be there. These include (but are not limited to):

Legality / Federal Laws When a government or other official instance prevents certain executions of a system. For instance, Denil et al. (2012) describes a power window that should stop closing when it gets a counter-force of 5 Newtons (to prevent it from chopping off limbs).

Possibility / Physical Laws Some situations may be mathematically reachable (in a simulator), but ignore all laws of physics (*e.g.*, a robotic arm cannot rotate through itself, or ships cannot sail over land).

Human Safety While some conditions may not be illegal, they should be prevented because of human safety. For instance, Paredis, Gomes, and Vangheluwe (2021) describes that a human may not be in an industrial high-oven when it is turned on.

[1]https://fmi-standard.org/

3.4 Usage Contexts

Another important aspect to study is when the twin is being used and in which context. Kang and Lee (2013) defines *Usage Context* as the set of circumstances where a system is operated in. It can also be seen as any contextual setting in which a product is deployed and used (Lee & Kang, 2010). Figure 3.4 gives the Feature Tree for the usage context.

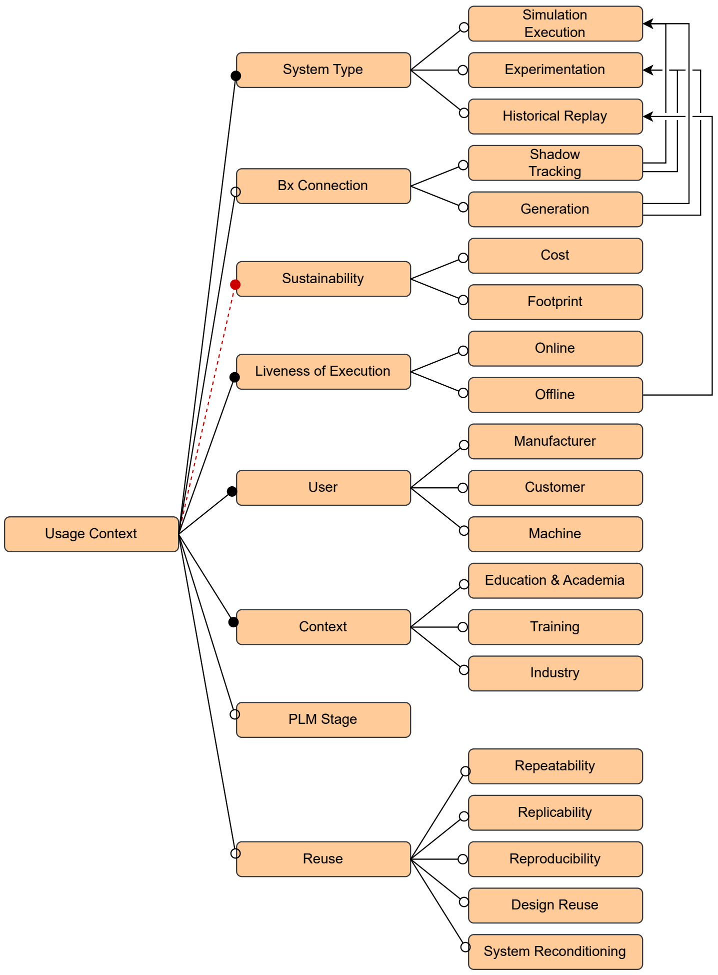


Figure 3.4: Feature Tree for the *Usage Context* of a Twin

3.4.1 System Type

How will the Twin be used? Is it mainly a *simulation* or *execution* of the TO (Damköhler, 2022), or will there happen some *experimentation* with the AO (Mahmoud et al., 2023; Z. Zhang et al., 2022)? Will the TO be used to simply replay some historical traces, or will it react to the current AO behaviour?

3.4.2 Bx Connection

This part identifies the bi-directionality of the connection between the AO and the TO. Kritzinger et al. (2018) has identified a *Digital Model* if there is no immediate connection between the two, and a *Digital Shadow* when the AO feeds the TO with sensor and environmental information. Tekinerdogan and Verdouw (2020) identify the dual of this as a *Digital Generator*. When there is an actual bi-directional connection between both the AO and the TO, Kritzinger et al. (2018) calls this a DT.

3.4.3 Sustainability

The world we live in has a lot of battles left to fight. About 3.5 billion people still live in poverty[2], preventing them the required access to education and employment. There is water scarcity and global warming ensures polluted air and a lot of greenhouse gasses. With the popularization of AI for the masses, even more pollution is being generated. According to Freitag et al. (2021), between 1.8% and 2.8% of global greenhouse gas emissions is due to ICT and digital computations[3]. Michael (2023) highlights that the biggest future for DTs appears when focusing on sustainability. Green IT is on the rise and twinning should jump on the bandwagon. Some research is already being done by analysing the energy efficiency of CPS and DTs (Bellis & Denil, 2022; Mohamed et al., 2019), sustainable manufacturing (He & Bai, 2021), and sustainable decision-making (Niloofar et al., 2021).

Additionally, we can consider the economical sustainability of having a Twinning System. It might have a high maintenance *cost*, or high emissions (*i.e.*, its *footprint*). Even when it minimizes the footprint of the AO, the Twin's cost should be minimized.

3.4.4 Liveness of Execution

The execution of the TO can happen *live* (*e.g.*, *online*, given real-time access to sensor information and state data), or *offline* (given past system execution traces). An online Twin is what is commonly desired, but an offline one can help with debugging, calibration and validation of the model.

3.4.5 User

Who will be using the twin or the twinned system? Commonly, this is a *manufacturer* when the Twin is used to create and develop the AO (Bolender et al., 2021; Lutters, 2018; Mandolla et al., 2019; Rožanec et al., 2021). When used at runtime, the user is a *customer* (Biesinger et al., 2018). Alternatively, Twinning ecosystems might ensure that a *machine* is using the Twin.

[2]https://www.worldbank.org/en/topic/poverty/overview, accessed 5th of March 2025
[3]However, given the recent interest in AI, these numbers might have increased drastically.

3.4.6 Context

Similarly, it should be considered what the context of the Twin itself is. Not all Twins are being used in an industrial setting.

Some Twinned systems are used for *education* (Um et al., 2017) or *training* (Boschert & Rosen, 2016). They allow people to work with an actual system, whilst still active in a controlled environment. This makes teaching the usage of (for instance) heavy machinery a whole lot safer.

In essence, any Twin can be used for education or learning about a complex system, even when they are not originally designed for it. Having enough data and system knowledge will enable understanding of a system. Adding a physical toy-system[4] to that, *kinaesthetic learners* (people who learn by doing (Reese & Dunn, 2007)) can even better understand the system.

3.4.7 PLM Stage

The origins of DTs can be traced back to PLM (Grieves & Vickers, 2017). It would hence make sense to maintain the equivalence between PLM and DTs (or Twinning in general). Many PLM systems are used to allow Twinning communication too (Dalibor et al., 2022). Note that a Twin is not required to be part of a single stage, but it might span multiple.

Design Section 3.2.1 describes the *design* stage of PLM. The AO does not exist yet, so technically we cannot yet consider twins.

Manufacturing The *manufacturing* stage focuses on how the system is constructed. It uses a (virtual) prototype in order to start constructing the actual SuS. We can start considering the prototype as a twin. Twinning is a common practice that appears whilst manufacturing an AO (Lutters, 2018). Additionally, there exist a plethora of Twins for the manufacturing process itself (Bolender et al., 2021; Mandolla et al., 2019; Rožanec et al., 2021).

Distribution The authors are not aware of any twinning examples specifically for the *distribution* stage. It may be possible that another company is interested in using the same Twin, or that twins are sold by a specific company, allowing distribution of Twins. Yet, this is different from a Twin that is used for the distribution process.

Operation Most twins are used in the *operation* or *usage* stage. Here, the AO is running in parallel to the TO. Maybe the AO starts to deviate (because of its specific usage context), allowing the TO also to deviate.

[4]Which is not necessarily a toy, but the actual system.

End-of-Life Decommissioning is an important part of any SuS. Sometimes parts can be reused in the same, or a different system. Being able to identify this will reduce waste, increasing the sustainability of these systems.

Mouflih et al. (2023) discusses the End-of-Life of an internet box. The remanufacturer has access to the information of the entire life-cycle of the internet box in order to identify the parts that are still reusable.

3.4.8 Reuse

Some Twinning Systems are created for a very specific use-case, yet others might be able to be implemented in many different scenarios and contexts. Reuse focuses on how a Twinning Systems might reappear in different scenarios. This can happen when the TE as a whole (or subcomponents thereof) are generic enough to be reused in other systems, or maybe even a future version or variant of the same system. Additionally, it is possible that such components or models can be shared over the entire life-cycle and can reappear at different PLM stages. Hong (2021) and Plesser (2018) define *repeatability* (the same team can produce the same results with the exact same experimental setup), *replicability* (a different team can produce the same results with the exact same experimental setup) and *reproducibility* (a different team can produce the same results with a different experimental setup) in the context of scientific experimentation.

When a Twin is used for optimization of an existing system, allowing the AO to perform better, *system reconditioning*[5] might occur. Finally, there is also *design reuse*, in which the current system design is reused to tackle a different problem (Landahl et al., 2018). Via a well chosen system architecture, design reuse can be implicitly enabled.

3.5 Conclusions

Research Question 1 asked "*What are the most common reasons and definitions for (creating) DTs?*". This chapter answers this question by outlining a comprehensive breakdown for understanding the key purposes and requirements involved in the development and application of DTs, and, by extension, Twinning.

The chapter described the usages and contexts of the Twinning Paradigm and which problems they can answer, mainly focusing on the individual goals and PoIs. Using Kang and Lee (2013)'s description of variability in the Problem Space, these requirements are categorized into *Goals*, *Quality Assurance*, and *Usage Contexts*. The chapter also provides a structured approach to address the various needs of DT systems – ranging from design and operational efficiency to maintenance and visualization. The examination of goals such as virtual prototyping, state estimation, predictive maintenance, and anomaly detection highlights the extensive capabilities of DTs in optimizing system performance and decision-making. Furthermore, the emphasis on synchronization, fidelity, and other ilities ensures that DTs maintain high operational standards and adaptability in diverse contexts, including industrial, educative, and sustainability efforts. Through

[5]https://sustainable-eng.com/what-is-retro-commissioning/, accessed 5th of March 2025

these detailed explorations, this chapter lays a robust foundation to start implementing Twinning Systems effectively across multiple domains.

This chapter identifies some (non-exhaustive) pointers for identifying the exact purposes (*i.e.*, stage Ⓐ) of building Twinning Systems. Chapter 4 delves into stage Ⓑ and shows that certain goals might have an influence on which components are required in this system. In Chapter 7 and Chapter 8, this categorization will be applied to a few practical examples.

"When you have eliminated all which is impossible, then whatever remains, however improbable, must be the truth."

– Sir Arthur Conan Doyle,
The Case-Book of Sherlock Holmes

Chapter 4

Conceptual Reference Architecture

In the literature, there are a lot of suggested architectures for constructing DTs, as stated in *Research Question 2*. Yet, despite the plethora of architectures, the lack of an exact definition for *Digital Twin* results in many different interpretations. While some sources consider only the TO as the *Digital Twin*, others consider the full Twinning system as such. Therefore, the *Research Question* asks if we can unify these. Such a unification allows researchers to have a common basis to start working on.

In reality, each constructed Twinning system is an *experiment* that is being executed short-term, long-term, or indefinitely. We can consider the full experiment as our main component, *e.g.*, a *first-class entity*. Here, we consider an experiment an intentional set of (possibly hierarchically composed) activities, carried out on a specific SuS in order to accomplish a specific set of goals. Each experiment should have a description, setup and workflow, such that it is *repeatable* (Plesser, 2018). We therefore carry out Twinning Experiments (TEs). A TE contains an AO and a TO, and ensures that both are *continually* synchronized. TEs can hence be considered first-class entities in the realisation of a Twinning architecture.

In this chapter, we will present a *conceptual* reference architecture, including its many variations and possibilities. Note that this conceptual architecture can also be seen as a *functional* architecture (SEBoK Editorial Board, 2023). It is meant to be a high-level and abstract representation of the many possibilities of a TE, allowing users to decide on a high level which components to include in their system. A mapping will be shown between our architecture and the most common architectures, obtained from the literature.

Note that this architecture model can change throughout the life-cycle of the system. For instance, it may be used to design and validate a model, after which the model is altered to get another model (Angjeliu et al., 2020).

4.1 Yet Another Twinning Architecture

Figure 4.1 shows the conceptual architecture that encapsulates the main components required, as previously defined in the literature (Dalibor et al., 2022; Kritzinger et al., 2018; Tao & Zhang, 2017). Following the IIRA's (and the RAMI 4.0's) *functional viewpoint*, the components are conceptualized in this general architecture.

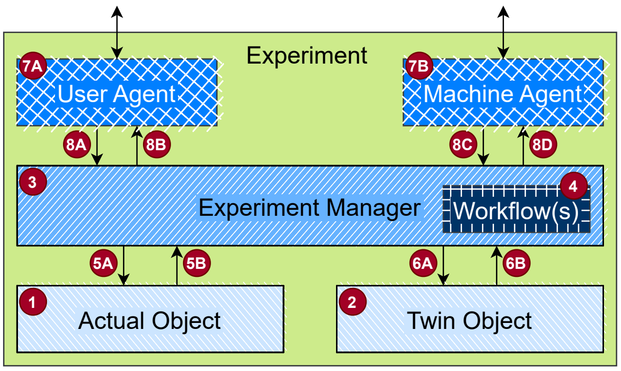


Figure 4.1: Generic (conceptual) TE architecture with presence conditions.

To further explain the figure, each component has been annotated with specific *variation points*, also known as *presence conditions* (*i.e.*, parts that can be enabled or disabled).

The AO ① and TO ② are given a behaviour via the *operational semantics* of the formalism in which they are modelled (not pictured). This can be a neural network, the execution of code, observed or controlled real-world behaviour… It is important to note that ① is an *abstraction* of a specific *view* on the actual world (and the environment) in which the SuS is active.

③ is the Experiment Manager (EM) (or *Orchestrator* in the case of black-box components). It contains a set of *Workflows* ④ that indicates *how* the experiment is to be executed. Because the experiment is created for a specific set of requirements, the requirement's logic is contained in ③. For instance, if we only want to have a dashboard to visualize the current state, the collection of this state is done by the EM. If instead our goal is anomaly detection, the EM needs to compute the distance between behaviour observed by the AO and computed by the TO– typically over a moving time window – and produce a notification when this distance exceeds a given threshold.

⑤ and ⑥ denote the communication between the EM and the AO or TO, respectively. Note that the downward communication may be interpreted in a really broad manner. It may consider the instructions that need to be sent to the objects, to launch or halt their individual executions. Alternatively, it may also send data to update the objects (*e.g.*, for a *Digital Generator*, *Digital Shadow* or *Digital Twin* (Tekinerdogan & Verdouw, 2020)). The upward communication can be the data that the sensors in this AO or TO have captured.

The *Orchestrator* can communicate with a *User Agent* (7A) or *Machine Agent* (7B). These are access points for a user or another system to obtain information about this experiment, or to send new information to this experiment. This communication happens through an exposed Application Programming Interface (API) of the User Agent and Machine Agent respectively. The communication ⑧ between the Agents and the *Orchestrator* is bidirectional when ⑦ can steer the twin, and one-directional in the case of a *Digital Model* or a *Digital Shadow* (Kritzinger et al., 2018).

Note that multiple TEs can be executed at the same time. Chapter 6 goes further into detail on this topic, yet it is important to already show how the orchestration between multiple TEs can work. In Figure 4.2, the top-level architecture for the orchestration of multiple TEs is shown. A top-level *Orchestrator* receives input (from top-level User Agents or top-level Machine Agents) and spawns new TEs. Note that these experiments can be short-running, long-running, or never-ending.

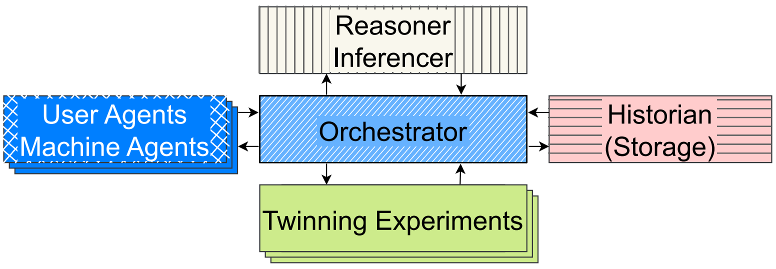


Figure 4.2: Orchestration of the conceptual architecture; adapted from Paredis and Vangheluwe, 2022.

Data gathered from the experiments is stored in the *Historian*. This is a blackboard (append-only, to support versioning and traceability) data lake that contains all historical data of all TEs. Cederbladh et al. (2023) identifies how such a *Historian* can be used. A User Agent sends a question Q to the *Orchestrator*, which may modify Q through some high-level reasoning. Next, the *Historian* is searched for an answer A. When no such answer can be found, a new TE E can be spawned, such that A can be obtained. Next, $\langle Q, E, A \rangle$ is stored in the *Historian* for later use.

If the experiment that must be spawned was already carried out in the past, the top-level *Orchestrator* might (driven by input from the *Reasoner*) collect the answers from the *Historian* instead (Mittal et al., 2023). This is shown as an example on the timeline in Figure 4.3. Some questions obtained from the User Agent or Machine Agent spawn *Reasoning* processes, others launch TEs. *Experiment 1* is a short experiment and *Experiment 2* is a never-ending experiment that continually shares information with the *Orchestrator*.

4.2 Common Architecture Variations

Note that each component and connection in Figure 4.1 may be present or absent, depending on the results from the requirement selection. This yields (at a naive first glance) $2^{13} = 8192$ different variations of this figure. The *Workflow(s)* component is only present if the EM is present.

Of course, many of these variations are useless within the context of Twinning. For instance, removing a component should not yield any dangling edges. And at least the AO or the TO should be present. They can both be there, but a TE without either does not make sense. Also, the EM should always be present on a conceptual level. Practically, it may be empty, but conceptually, the EM controls the full TE. When these constraints are added, a mere 240 useful variations remain.

Ideally, the results from the goal selection stage provide a subset of acceptable possibilities in this stage (*i.e.*, the architecture definition stage), which can be further limited by more

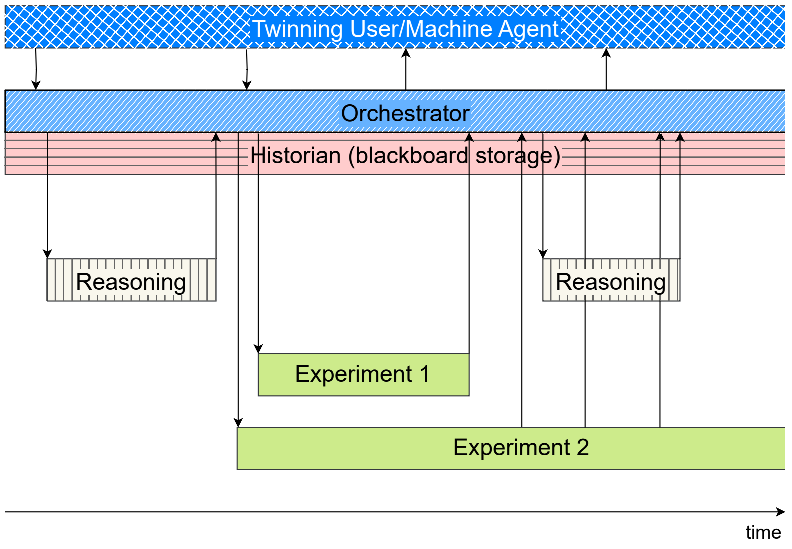


Figure 4.3: Timeline of TEs; adapted from Paredis and Vangheluwe, 2022.

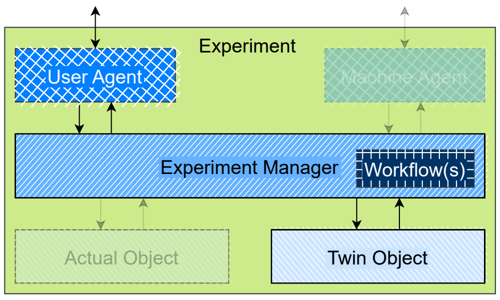
custom requirements. Each of the resulting variants should fully represent a conceptual architecture of the desired TE.

A small analysis of the literature was conducted. After studying the architectures of 95 works related to DTs, there were clearly 4 variations that were used the most often. They are pictured in Figure 4.4. A fifth variation (Figure 4.4e) appears when we simply focus on measuring experiments. This architecture is often not considered within the scope of DTs, but is fairly commonly used in practice. Of course, this is not a detailed analysis to identify these for certain, but they can help in painting the picture.

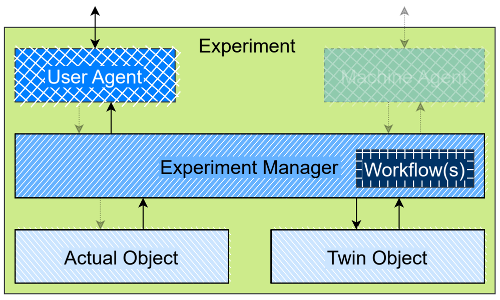
Figure 4.4b shows a variation where there is no Machine Agent present, and the user, nor the EM can influence the AO. This architecture was the most commonly used in the analysed literature, with over half of the sources using it. It basically represents a *Tracking Simulator*, or a *Digital Shadow* (cfr. Kritzinger et al. (2018)). Note that when there are only sensors in the AO; this system will be at most a *Digital Shadow*. If there are also actuators, we may have a DT.

The other architectures appeared rarely. Variations 120 (Figure 4.4c) and 240 (Figure 4.4d) are almost the same. All components are present, but in variation 120, the user cannot influence the TE. Finally, there also appear variations like Figure 4.4a, where there is no AO. This is clearly indicative of a simulation-only system, with potential user interaction.

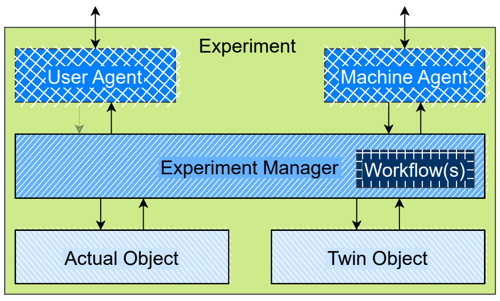
Variation 186 (Figure 4.4e) shows a simple human-influenced measuring experiment. The AO is instructed to behave a certain way, and the user can analyse this behaviour. This is typical for a calibration or validation experiment.



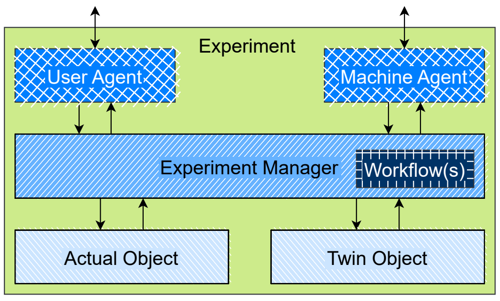
(a) Variation 183: without AO and Machine Agent.



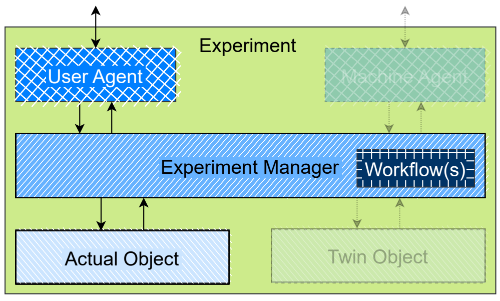
(b) Variation 69: without Machine Agent, or influence upon the AO.



(c) Variation 120: User Agent cannot influence the TE.



(d) Variation 240: everything is present.



(e) Variation 186: without TO and Machine Agent.

Figure 4.4: Five architectural variations for the conceptual reference architecture.

4.3 Mapping the Architecture

In order to accurately map the presented architecture onto the ones presented in the literature, a distinction needs to be made between the high-level (also known as conceptual or functional) architectures, and the realized architectures (*i.e.,* software architectures or deployment diagrams).

4.3.1 Conceptual/Functional Mapping

Let's first have a look at the architectures that focus on the full TE as a whole.

Most notably, Kritzinger et al. (2018) has shown a high-level separation into *Digital Model*, *Digital Shadow* and *Digital Twin*, based on the communication between the AO and the TO. Tekinerdogan and Verdouw (2020) goes further by also introducing a *Digital Generator*. The connections and direction of the data flow in the conceptual architecture define which categorization can be applied. For instance, if $\textcircled{5B}$ does not exist, we consider it a *Digital Generator*. If $\textcircled{5A}$ is omitted, the TE is a *Digital Shadow*.

In Paredis, Gomes, and Vangheluwe (2021), we have identified that the TO itself should only be a model of the behaviour of the AO. Any additional processing should be separated into another component – then erroneously called the *External Object*, but later corrected to the *Assimilation Object* or the *EM*. Other sources have identified the same separation of concerns and considered this external component as a set of *Services* or *Tools* (Madni et al., 2019). Llopis et al. (2023) separates these into *Service Components* and *Analysis Components*. In essence, these architectures are all equivalent to the 5D architecture (Tao & Zhang, 2017). Figure 4.5 shows a mapping of the components of Figure 4.1 onto this 5D architecture.

4.3.2 Mapping through Realization

Tekinerdogan and Verdouw (2020) not only introduces the *Digital Generator*, but also opens the "boxes" and shows the expected contents of the AO and the TO. For the TO, a simplified MAPE-k is executed in the modelled space. Bibow et al. (2020), Bolender et al. (2021), Eramo et al. (2021), and Michael and Wortmann (2021) focus in more depth on the adoption of the MAPE-k loop in the TO itself. Data collected from sensors of the AO is sent over $\textcircled{5B}$, and commands for the actuators over $\textcircled{5A}$. A comparator or analysis component and a decision or plan component are part of the EM and the associated *Workflow(s)*. Runtime knowledge from the current TE execution is also located in the EM. Overall knowledge of the system can be found in the top-level *Historian*. Sometimes, multiple models are referred to. Each of these models is their own TE, which are all coordinated through the top-level *Orchestrator*.

Another architectural axis that appears is a layered approach (Kibira et al., 2021; Redelinghuys et al., 2019), where the deployed technology stack is included. The same components can loosely be identified here. Some arrows have become their own layers

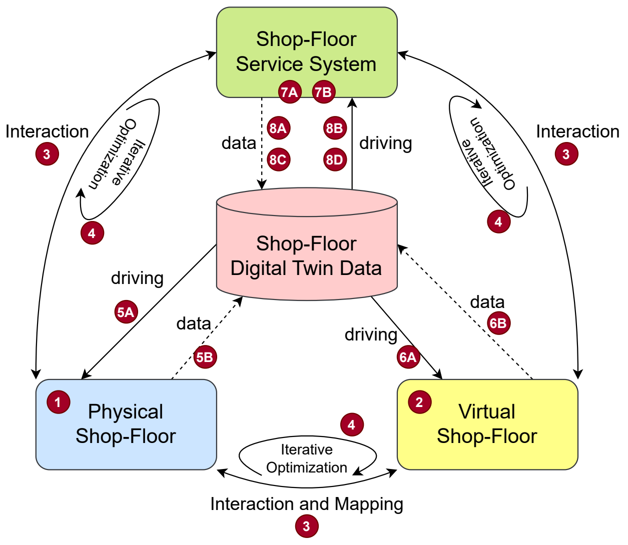


Figure 4.5: The 5D architecture, based on Tao and Zhang (2017), as shown in Figure 2.3, annotated with the presence conditions from Figure 4.1.

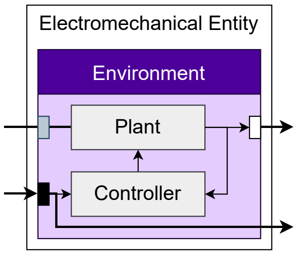
and the EM might not be deducible, other than through documentation and execution traces. This highly depends on which components are merged due to realization.

The AAS also has a layered architecture. From the *Business Layer*, they will only look at the overall TE, but specifically the *Workflow(s)* and the behaviour of the EM. The presented architecture can act as the *Functional Layer*. The *Information Layer* is encoded in the EM, as well as the *Historian*. The *Communication Layer* details the Machine Agent, the *Integration Layer* is defined by the deployment and the *Asset Layer* is the physical asset that the AO is a view of. Notice the relationship between the RAMI 4.0 layers and the IIRA Architecture Framework (Industry IoT Consortium, 2022).

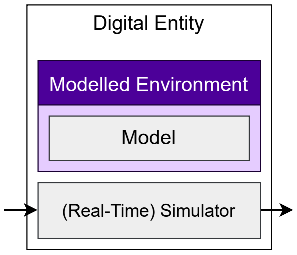
4.4 Actual Object and Twin Object

The AO and the TO themselves (including their operational semantics) can be separated into more specific architectures, based on the desired goals w.r.t. their PoIs. This is illustrated in Figure 4.6. A black box is an input, a white box an output, and a darkgrey box an inout port. Each inout port is connected over an arrowless connection, implying an external influence on the components, outside of the control of a user (*e.g.,* the sun is shining on a sensor). Full arrows indicate an influence relationship between components. A striped arrow is an observation (when connected to an output), or a manual influence (when coming from an input). For the AO, the incoming and outgoing arrows in the subfigures of Figure 4.6 relate to ⑤ in Figure 4.1. Similarly, for the TO, the incoming and outgoing arrows represent ⑥.

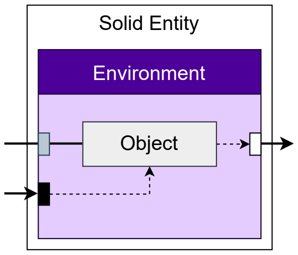
Dalibor et al. (2022) has identified that there also exist Twinning systems that do not link a *physical* SuS to a *digital* model. The authors have identified *biological beings*, *individual*



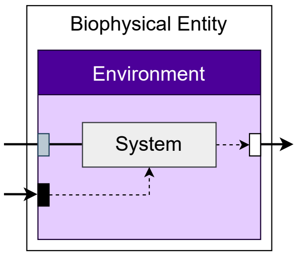
(a) Electromechanical Entity.



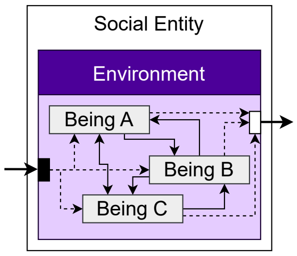
(b) Digital Entity.



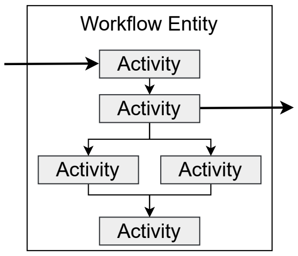
(c) Solid Entity.



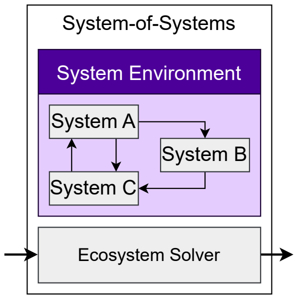
(d) Biophysical Entity.



(e) Social Entity.



(f) Workflow Entity.



(g) System-of-Systems Entity.

Figure 4.6: Seven different entities that AOs and TOs can be.

systems, *processes*, *products*, *system of systems*, and *other counterparts*. In fact, we hypothesize a *non-exhaustive* list of 7 different kinds of entities to which both the AO and the TO may conform to (Paredis et al., 2024). Depending on which type of entity the AO and the TO are, different deployment techniques are required with their own complexities. Of course, not all AOs and TOs neatly fit into one category. There may be some overlap.

4.4.1 Electromechanical Entity

Commonly, in a DT, the AO is considered *physical*, which implies that it is a SuS that is active in the real, physical world. It focuses on CPSs and other industrial machines (especially within the context of Industry 4.0). These are some mechanical devices that may have some electronic parts, yet they do not exist in a purely digital world.

When they appear as an AO, they can be cars (Alvarez, 2019; Barosan et al., 2020), machines (Bibow et al., 2020; Mandolla et al., 2019), robots (Marah & Challenger, 2023; Marion, 2021; Paredis & Vangheluwe, 2021; Walravens et al., 2022) and other large systems.

A TO *Electromechanical Entity* may concern analog computers that use mechanics to mimic real-world behaviour, or physically built training simulators. The Apollo 13 Space Capsule Training Simulator (Ferguson, 2020) was also this kind of entity.

One possible architecture for such a real-world system, focusing on a plant-controller feedback loop is given in Figure 4.6a, but other alternatives exist as well. This system is active in a (subset of) the real-world environment, which implies that this should also be incorporated in the entity. Additionally, there may be environmental influences we did not account for. For instance: a really sunny day might yield a sensor to produce invalid results. The operational semantics of this entity equates to its behaviour in the real-world (*i.e.*, physics).

4.4.2 Digital Entity

A DT has its TO in the digital world, which is identified as a *Digital Entity* in Figure 4.6b. Here, a model in some formalism is given operational semantics by executing a solver (or a simulator). This may potentially be a real-time simulator, or an as-fast-as-possible simulator, depending on the chosen goals.

While it is common for a *Digital Entity* to exist as a TO (X. Feng et al., 2023; Moya et al., 2020; Paredis & Vangheluwe, 2021; Utzig et al., 2019), there are also occurrences as an AO. There exist Twins for running software (Heithoff et al., 2023), running simulations, virtual worlds (*i.e.*, BIM, CAD,. . .) (Angjeliu et al., 2020) and data (*i.e.*, historical traces, experiment results,. . .).

4.4.3 Solid Entity

A *Solid Entity* is an arbitrary physical object that has no behaviour on its own. It can be influenced by users, machines, or the environment, but it does not do anything. This is illustrated in Figure 4.6c.

Bonney et al. (2021) discusses how a TE of a metal shelf can be constructed and Wang et al. (2018) focuses on reinforced plastics. The geometrical variation problem in the construction of sunroofs was analysed by Wärmefjord et al. (2017). Similarly, Mukherjee and DebRoy (2019) constructed a DT for qualification of metallic 3D printed objects. Another example is the maintenance of masonry structures and buildings (Doellner et al., 2023; Domaneschi et al., 2023; Mukherjee & DebRoy, 2019). 3D point clouds for smart city cases also commonly can be categorized as *Solid Entities* (Ballouch et al., 2022; Münzinger et al., 2022).

For the TO, the oldest *Solid Entity* example is a *Sand Table* (see also Chapter 10), which have been used since the antiquity and are still being used (albeit mainly in movies and entertainment) for military planning and wargaming (D. E. Smith, 1925; Wisher, 2001). Antwerp Zoo teaches children about deforestation and its impact on the tamarin population using a miniature model of the rainforest and decommissioned fire-hoses that represent roads or highways. Baalbergen et al. (2023) mentions that civil registration under Napoleon created *Paper Twins*. Boletsis (2022) discusses a 3D physical map on which projectors colour the textures. This is a common installation in museums; as well as light spectacle tours in cities. Architects build scale models of houses to test their structural soundness, capacity, light influence, . . . The right mineralogical combination of soil can even mimic the surface of Mars (Long-Fox et al., 2023).

4.4.4 Biophysical Entity

The *Biophysical Entity* is at the cross section of biology, chemistry, and physics. It considers people and animals (as a complex biochemical system), organs, plants, crops, liquids, chemical processes, medicine, weather phenomena. . . In a way, the *Biophysical Entity* is a *Solid Entity* that has some sort of autonomous behaviour, as can be seen in Figure 4.6d.

While it is generally not easy to act upon *Biophysical Entities*, they are commonly studied as AO within the Twinning domain. David et al. (2023) discusses Cyber-Biophysical systems within the context of DTs. Other sources focus on greenhouses (Howard et al., 2021), agriculture (R. et al., 2023; Shrivastava et al., 2020; Tekinerdogan & Verdouw, 2020; Verdouw & Kruize, 2017), aquaponics (Mahmoud et al., 2023), chemical reactions (Silber et al., 2023), the sloshing movement of water (Moya et al., 2020), the perception of colour on colour-changing materials (Yuan et al., 2022), or life sciences (Antunes, 2023; El Saddik, 2018; Knibbe et al., 2022). Alternatively, there is also the *Earth Digital Twin* (also known as *Digital Twin of the Earth*), which focuses on meteorology and sustainability of our planet (Bauer et al., 2021; European Space Agency, 2021).

It is generally close to impossible to influence a *Biophysical Entity* and use it as a TO. Yet, Jayed and Carlomagno (2024) focuses on olfactory recovery through 3D structures, colour and aroma to mimic another perception of taste. In other words: a model of some food tricks the brain into experiencing the taste of it. Badawi et al. (2021) uses the idea of DNA structures as unique identifiers in DTs, specifically applied to smart cities. The field of physics has this notion of *Universality*, which identifies that "*under certain circumstances behaviour in one physical system is very similar to that in [another]*"(Hendy, 2015). This appears, for instance, in the relationship between the Coulomb force and gravity. While this theoretically fits the Twinning Paradigm, we cannot *practically* use this relationship for Twinning.

4.4.5 Social Entity

The *Social Entity* (Figure 4.6e) considers people and animals within a social context. The human or animal is not considered biochemically. It mainly focuses on the interaction between beings, and the behaviour of the individual. Dalibor et al. (2022) identified this as *Biological Beings*.

For the AO, examples could be employees in a business structure (Graessler & Poehler, 2018), citizens in a city (Traoré, 2023), sports players (Balachandar & Chinnaiyan, 2019), South-African wildlife (Fergus et al., 2023), cows in a cattle (Tekinerdogan & Verdouw, 2020; Verdouw & Kruize, 2017), a swarm of fish (Joordens & Jamshidi, 2018), or actors and animals using motion capture (Berti et al., 2023).

There also exist TO examples. Both Tero et al. (2010) and Topuzoglu et al. (2019) used slime molds to construct an optimized railway network, thus modelling a city through biology. A more imaginative example is acting. Actors that accurately portray historical figures can technically also be seen as a model. Medical simulants, or people partaking in family composition activities can also *technically* be considered *Twins*.

4.4.6 Workflow Entity

The *Workflow Entity* concerns all workflow-based systems (*i.e.*, a system in which a sequence of activities is studied). Dalibor et al. (2022) calls these *Processes*. An example is shown in Figure 4.6f.

Rambow-Hoeschele et al. (2018) have business processes as their AO. Karakra et al. (2018) focuses on medical processes and Popa et al. (2018) identifies recycling processes.

Looking at TO examples, more imagination is required. A set of model transformation rules can be considered a model for the behaviour of a model. In the literature on Twinning, no immediate examples were found of a *Workflow Entity* as a TO.

4.4.7 System-of-Systems Entity

Figure 4.6g shows a *System-of-Systems Entity*. For instance, collaboration of multiple machines in a factory (Biesinger et al., 2018). If this is digital, this will likely be solved using co-simulation. If it is in the real-world, its behaviour is the union of the sub-system behaviour, including the communications.

In terms of a TO, we can consider systems like *boids* as a *System of Systems Entity*. In the literature on Twinning, no immediate examples were found of a *System-of-Systems Entity* as a TO.

4.5 Conclusions

Research Question 2 poses "*Given the large number of existing DTs in the literature, can we unify?*", and this chapter has mainly focuses on that in terms of the architectures. We believe it is indeed possible to unify under a common architecture, ensuring the necessary variability is still possible. This chapter has focused on stage Ⓑ by presenting a new *conceptual* Twinning architecture, using presence conditions, based on the literature, as stated in the *Research Question*. A TE is considered a first-class entity, managed by an EM. Both the AO and the TO can be specific entities, based on their exact behaviour. This architecture can be linked to numerous existing architectures in the literature. In fact, there are only a handful of variations that are practically being used. Additionally, top-level orchestration of multiple TEs is mentioned, but will be discussed in more detail in Chapter 6. Chapter 7 and Chapter 8 will show how this architecture (and its many variations) can be used in practice. Chapter 5 will briefly go over stage Ⓒ.

"God creates dinosaurs, God kills dinosaurs, God creates man, man kills God, man brings back dinosaurs."
– Michael Crichton, Jurassic Park

Chapter 5 Technologies and Deployment

In order to fully realize a TE, it still needs to be deployed. However, this is the stage where the largest number of choices. There is a vast body of research on theoretical frameworks for DTs, but their practical implementations and deployment is still lacking (Hassan & Aggarwal, 2023). Jeong et al. (2022) has identified five layers for the deployment of Twinning Systems. The authors show a step-by-step approach to practically built a Twinning System, specifically focusing on the deployment and implementation aspect.

It is impossible to chart the full level of variability that appears at the level of deployment. This chapter analyses the deployment techniques used in the literature, as a rudimentary overview of the plethora of possible variations, as well as an identification of the most common approaches. This way, it aims to find some common ground for system engineers to start from.

5.1 State-of-the-Art

The ISO/IEC 12207 international standard (R. Singh, 1996) describes the life-cycle of software processes, from the initial idea until retirement. Yet, it misses an important aspect of deployment. Similar as the individual phases for creating software, there are a lot of phases for actually deploying a system.

The IIRA Architecture Framework (Industry IoT Consortium, 2022) bridges this gap by introducing the *Implementation View*. A large issue is that a lot of components are black-boxes when considering implementation details. There is little to no access to information about the internal workings. Commonly, this is due to IP protection.

There is hence the question of which components to use, but also *"does this component do what I want it to do?"* Additionally, we need to know where the component should be physically located on the device, or how to link it to other components.

The AAS (Boss et al., 2020) identifies deployment as an entire spectrum in which a lot of decisions have an impact on the eventual technologies. A minimal latency might imply using the edge as opposed to the cloud. There are multiple techniques required for accurately deploying DTs. Schäfer et al. (2021) explored this deployment in the context of the AAS.

Even in the literature, the question of deployment arises, commonly combining IoT with DTs and blockchain (X. Feng et al., 2023). Daniel Lehner explored multiple deployment techniques and tools for DTs[1]. Bellis and Denil (2022) have focused on the sustainability consequences of actually deploying DTs.

5.2 Literature Analysis

An analysis of 95 papers, focusing on DT technologies helped to identify the most common technologies used for Twinning. Some papers presented without use-cases, whereas others discussed multiple use-cases. In total, 560 references to technologies were identified and a *non-exhaustive* list of 124 unique technologies[2] was constructed. It is important to note that it is perfectly possible some references were overlooked in this analysis. The relatively small sample size might not project the most realistic statistics, whilst also being highly prone to bias. Nevertheless, the obtained results are enough to paint a picture as to how variability of Twinning appears in the deployment stage. It shows which parts should be discussed when considering the deployment of Twinning.

We can identify five main categories for these references: *Generic Methodologies, Concepts and Buzzwords* (28.34%), *Languages* (32.44%), *Tools, Frameworks and Technologies* (17.47%), *Communication Protocols* (10.52%), and *Hardware* (11.23%). Note that the summaries in the upcoming sections will only focus on identified technologies in their corresponding category.

5.2.1 Generic Methodologies, Concepts and Buzzwords

This category focuses mainly on generic technology descriptions, oftentimes without exact details. This either happens as a buzzword, or to provide a generic insight into the use-case. In Figure 5.1, the distribution of these generic technologies is shown, for use-cases that mentioned these.

It is not surprising that (Industrial) IoT appears the most often in these solutions (25.2%), immediately followed by AI solutions (mainly Machine Learning and Reinforcement Learning) and more generic Big Data approaches (23.3%). Dalibor et al. (2022) has identified that most DT research is focuses in the manufacturing domain, albeit potentially biased by the widespread application of DT in manufacturing. Additionally, there is a lot of data required for constructing and maintaining TEs, enabled by the recent rise in interest in AI solutions.

15.1% of use-cases in this category explicitly mention the use of a 3D world, either obtained via a scan, or by 3D modelling, which is closely related to AR, VR, and XR (6.9%). 8.8% clearly indicate the usage of cloud, edge or fog computing and 5.7% mentions using ontologies.

[1]https://derlehner.at/my-view-on-digital-twins/, accessed September 5th, 2024.

[2]Some of these were merged into one because they belong to the same ecosystem, or identify the same technique or concept, like the Eclipse ecosystem, or the merge of CBDs and MatLab SIMULINK.

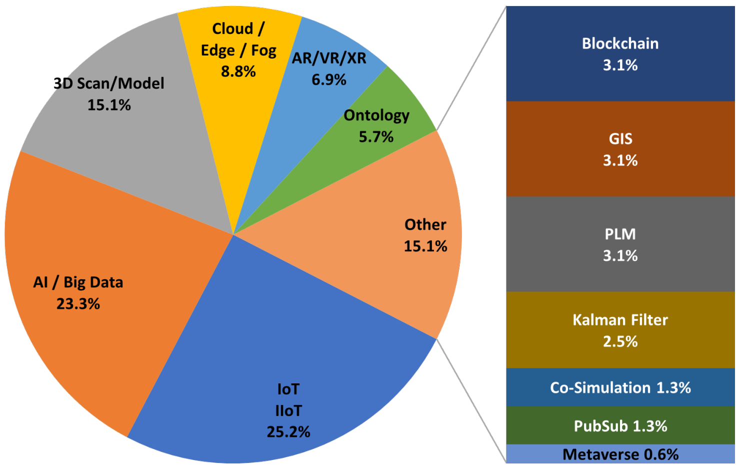


Figure 5.1: Distribution of identified generic technologies.

PLM, Geographic Information System (GIS) and blockchain all appear 3.1%. Even though the latter can be surprising, Banerjee et al. (2023) highly argues the usage of blockchain for DT solutions. Techniques as Kalman filters (2.5%), co-simulation (1.3%), and publish-subscribe (1.3%) appear to be relatively popular as well. In fact, there are a lot more use-cases that use publish-subscribe without mentioning it explicitly. 0.6% identifies the Metaverse[3] as a potential solution.

There is therefore no real consensus on which general technologies work best. However, it is common to combine IoT, AI and 3D techniques.

5.2.2 Languages

This category encapsulates all (domain-specific) modelling languages, programming languages, markup languages and general standards. Most of the technological mentions in the use-cases belong in this category. Figure 5.2 summarises the results, for all use-cases that had a mention of a language.

Mentions of General Purpose (Programming) Languages appear the most often in these use-case descriptions – about 28.7%. This category groups Python (6.1%), Java (5.5%), C# and .NET (3.3%), C and C++ (2.2%), R (1.1%), Ada (0.5%) and G-code (0.5%), but also XML and JSON (7.2%), and HTML and PHP (2.2%). For readability, the figure omits these details.

The Domain-Specific Language category (3.9%) identifies smaller languages belonging to a specific domain. EDI-Teelt, ALISA, PackML, SOIL, SOML++, SystemC AMS, and Jason all appear once in the data.

Following General Purpose (Programming) Languages, CAD solutions (13.8%) are the

[3]Not the one from Meta.

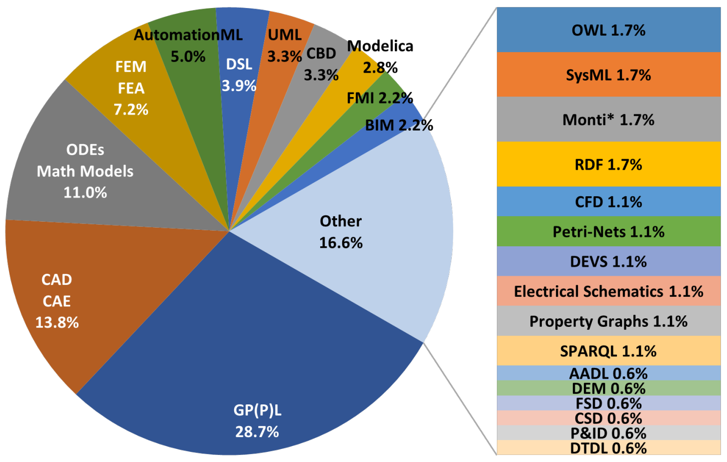


Figure 5.2: Distribution of identified languages.

most prominent in the observed papers for this category. This is closely related to the relatively high 3D model mentions in the previous category.

ODEs and general mathematical models appear in 11.0% of use-cases here. Particle modelling approaches appear 9.9%. They include Finite Element Modelling and Finite Element Analysis (7.2%), Computational Fluid Dynamics (1.1%), and Discrete Element Method (0.6%).

The markup languages make up for 13.5% of these references. The identified mentions include AutomationML (5.0%), UML (3.3%), SysML (1.7%), MontiCore (1.7%), AADL (0.6%), Composite Structure Diagram (0.6%), and the Digital Twin Design Language (DTDL) (0.6%).

Common modelling languages used for the use-cases are CBDs and MatLab SIMULINK (3.3%), Modelica (2.8%), Petri-Nets (1.1%), DEVS (1.1%), Electrical Schematics (1.1%), Property Graphs (1.1%), Forrester System Dynamics (0.6%), and Piping and Instrumentation Diagram (0.6%). This totals for 11.7%.

Both BIM and the FMI appear in 2.2% of the cases. OWL and RDF logically appear both in 1.7% of the cases in this category, and SPARQL in 1.1%.

As a result, we can state that any language that can be used to achieve your goal(s) can be used in the creation of a Twinning System. There is no golden standard. This solution highly relates to Multi-Paradigm Modelling (MPM).

5.2.3 Tools, Frameworks and Technologies

There are a lot of mentions on the software-level tooling used to realize the use-case. In fact, 50 (40.3%) of the unique technologies fall within this category; 30 of which appear only once in the data. For readability, Figure 5.3 shows a higher-level separation of technologies for mentions in these use-cases.

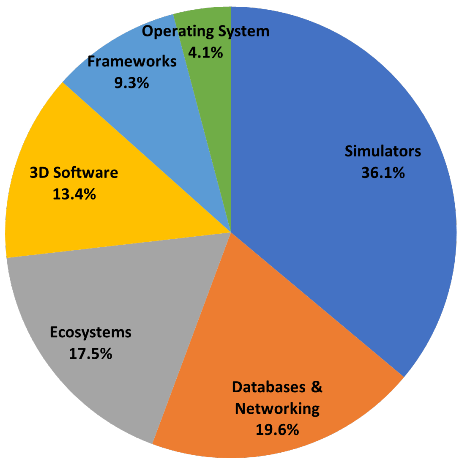


Figure 5.3: Distribution of identified communication protocols.

Simulators (and General Software) account for 36.1% in this category. These include (but are not limited to) MatLab SIMULINK (7.2%), AnyLogic (3.1%), Protégé (3.1%), Abaqus (3.1%), JADE (2.1%), OSATE (2.1%), Ocarina (1.0%), Papyrus (1.0%), Ptolemy II (1.0%), Rhapsody (1.0%)... Additionally, R Studio (2.1%) and Visual Studio (1.0%) are also included here.

Databases and Networking (19.6%) identifies all tools and technologies that ensure communication and the storing of data. Technologies like SCADA (5.1%), Kubernetes (3.1%), RabbitMQ (3.1%), Kafka (2.1%), MongoDB (2.1%), OpenHAB (1.0%), Redis (1.0%), and Oracle DB (1.0%) belong to this sub-category.

Ecosystems (17.5%) includes collections of tools that are presented under a common umbrella. This is usually a software bundle from a specific organisation. It includes Eclipse (6.2%), Siemens (5.1%), Amazon (3.1%), and Microsoft (3.1%).

3D Software (13.4%) includes all software that has a main focus on 3D visualisation and computation. Unity 3D (7.2%) is the most commonly used, followed by Simio (2.1%), Delft3D (1.0%), Grasshopper (1.0%), Verosim (1.0%), and Xsens Studio (1.0%).

Frameworks (9.3%) groups specific addons, plug-ins, libraries, SDKs, SDEs and other extensions to existing tools and languages. It includes Vuforia SDK (2.1%), ArcGIS (1.0%), Cesium Map Engine (1.0%), React (1.0%), Web3.js (1.0%), and Flask (1.0%). This can also include specific APIs like Google Carthographer (2.1%).

Interestingly, merely 4 use-cases explicitly present the Operating System on which their deployed system runs. Even though it's possible to rather implicitly obtain this information, it is counter-intuitive how little it is mentioned. One would suspect that the choice of Operating System is a pertinent and highly important variability point in constructing TEs. Its omission in the literature shows that this variability has become obsolete, or is often wrongly ignored.

We can conclude that simulators are highly common in the construction of Twinning Systems[4]. Additionally, it is useful to maintain databases and focus on the network connection. These are highly necessary for a continually updating system (as is the case for systems in the Twinning Paradigm).

5.2.4 Communication Protocols

One of the core aspects of a TE is the communication between the AO and the TO. Figure 5.4 shows the most common protocols used in the analysed subset of use-cases.

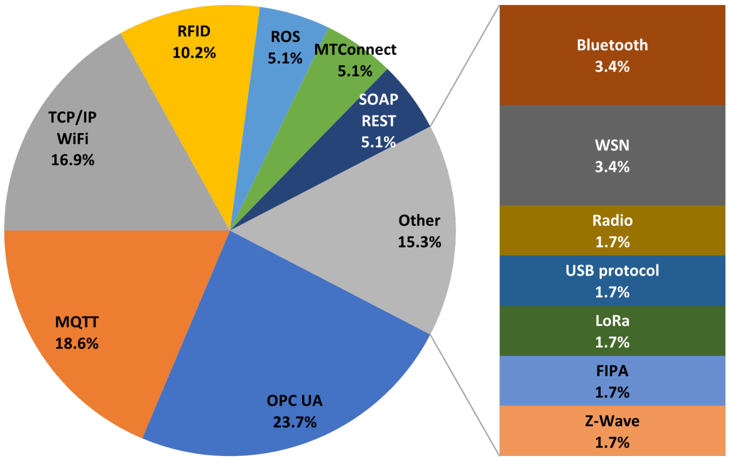


Figure 5.4: Distribution of identified communication protocols.

The most prominent technology used is OPC UA (23.7%), followed by MQTT (18.6%) and TCP/IP or WiFi (16.9%). RFID communication is used in 10.2% of these cases, however often not as a communication between the AO and the TO. ROS, MTConnect, and SOAP or REST appear in 5.1% of these cases.

Less common techniques like BlueTooth (3.4%), Wireless Sensor Network (WSN) (3.4%), radio communication (1.7%), wired communication via USB (1.7%), Long Range (1.7%), the Foundation for Intelligent Physical Agents (FIPA) (1.7%) standard, and Z-Wave (1.7%) are also used.

Hence, there is no real common standard for communications, and all technologies that accomplish the desired task(s) can be used. Preference is usually given to more common and heavily supported protocols. They also better ensure the continual communication between an AO and a TO.

[4]Or, the papers were biased to the M&S domain.

5.2.5 Hardware

This category focuses on the explicit mentions of certain hardware that was used. On the hardware level, there is an even larger explosion in possibilities, as the number of specific sensors, brands and mounting details will yield different products in the overall product family. They were not all individually counted. Figure 5.5 shows an overview of the cases that mentioned hardware.

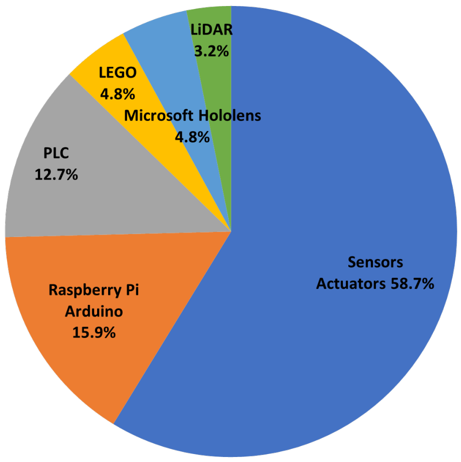


Figure 5.5: Distribution of identified hardware.

Almost all use-cases that mentioned hardware, identified a unique technology, with a very limited intersection in different technologies. This is identified in the Sensors and Actuators section of the plot (58.7%), which highlights that the use-case used a specific sensor or actuator.

In 15.9% of the hardware occurrences, a Raspberry Pi or an Arduino was used as a core computational component. 12.7% used a Programmable Logic Controller (PLC) and 3.2% used LiDAR techniques. Linking to the previously identified AR/VR/XR category, 4.8% has identified to use the Microsoft Hololens.

4.8% of these use-cases used LEGO as hardware, presumably due to its cheap and universal connectability. This allows fast prototyping and proof-of-concept. The same reasoning was applied when constructing our original LFR (Paredis & Vangheluwe, 2021).

Given that most hardware mentioned identifies really specific solutions, we can see the same pattern as in the previous sections: system engineers use the most appropriate components for their individual use-cases. Sometimes, more common hardware is used, but this happens rarely.

5.3 DEVS Simulation

The main purpose of this chapter was trying to identify if there exists some common ground for technologies or deployment concerning Twinning Systems, but as this small analysis has proven, this is not the case. In fact, mostly the MPM methodology is followed in using the most appropriate technologies for the specific use-cases. Hence, it might make more sense to figure out a way to quantitatively support deployment choices – yielding *Research Question 4*.

With this many possibilities, selecting specific tools, frameworks, modelling languages and technologies might seem arbitrary. It is likely that the system engineers make these selections based on the available technologies, expertise and economical situation. However, it is likely that the exact technologies first need to be analysed before they can be applied. Luckily, MBSE offers a solution for this. Assuming that the conceptual system architecture exists and the desired goals are well defined, a real-world problem can be transformed to the simulation world, enabling deployment space exploration. This is a technique that also will be used in Chapter 8.

Multiple technologies can be experimented with in order to obtain the best "theoretical" solution to the presented problem. The downside of this approach is that it requires a modelled version of all the desired technologies.

When the technology already exists for simulation purposes, model transformations, embedding and co-simulation may be used to ensure a valid representation of this technology in the simulated world. However, when a model for the technology does not exist yet, this becomes much more difficult and will likely involve a collaboration between the running simulation and the real technology.

For instance, a communication protocol has an explicit description of its behaviour. This description can fully be implemented in the simulated world, such that the exact same logic is used, thus representing the real technology. Unfortunately, this approach is costly and requires a focus on a specific technology, instead of the total system's behaviour. However, communication protocols commonly focus on the transfer of information, as opposed to its modification. This is why they can be represented with a simple delay of this information. Much better would be a randomized sample from realistic delays, which may have been obtained by doing a lot of experiments with the real technology.

5.4 Concusions

This chapter briefly shows an analysis of 95 papers that explicitly mention certain technologies used for DTs in the literature. While there are a few standards, like the ISO/IEC 12207 international standard (R. Singh, 1996), the IIRA Architecture Framework (Industry IoT Consortium, 2022), the RAMI 4.0 (Hankel & Rexroth, 2015), or the AAS (Boss et al., 2020) that mention deployment of DTs, very few academic use-cases actually make use of them. Mostly, DTs are constructed and orchestrated with the tools, frameworks, and knowledge available for the system engineers. Stage Ⓓ explicitly focuses on the deployment, but due to the vast explosion in possibilities, it is impossible to discuss all options.

Chapter 7 and Chapter 8 show how some use-cases can be practically deployed, using deployment diagrams. The latter chapter will especially focus on how we can do *deployment space exploration*, using DEVS (see Section 5.3), on a simplified use-case.

"To every problem, there is a most simple solution."
– Agatha Christie, The Clocks

Chapter 6 Combining Twinning Architectures

Up to this point, we have assumed a single requirement is used as a core goal for a Twinning System. However, this is often not the case. Many Twinning Systems are built to accommodate multiple requirements from the Feature Trees presented in Chapter 3. Hence, multiple TEs need to collaborate to achieve an overarching set of requirements, sometimes referred to as a *Digital (Twin) Ecosystem* (Hofmeister et al., 2024; Nativi et al., 2021; Rivera et al., 2022; Rosen et al., 2019). As adoption of Twinning increases, the need to combine TEs arises. The core question here is how to combine, compose, and federate such systems; effectively yielding *Research Question 5*: *"How can we combine multiple DTs into a larger system?"* Multiple TEs may correspond to different requirements and PoIs; may correspond to different components in an architecture; may represent a system at different levels of detail, abstraction, or fidelity; may be used at a type, aggregate or instance level; *etc.* Maybe there are even over-arching TEs that combine multiple sub-TEs. This would not only allow the creation of TEs of systems, but also TEs of Systems of Systems.

Rivera et al. (2022) describes cooperating DTs within the context of a Smart Urban Transit Digital Ecosystem. Nativi et al. (2021) discusses DT ecosystems within the scope of DTs of the Earth, mainly focusing on data integration, whilst remaining abstract in terms of realisation. In Hofmeister et al. (2024), it is discussed that the interconnection between these models ensures an ecosystem of connected DTs. Akroyd et al. (2021) delves deeper and discusses The World Avatar project, which aims to create a *Digital Avatar* of the world to model the interoperability of the isolated but conceptually connected domains that drive our planet. Boschert et al. (2018) and Rosen et al. (2019) introduce NexDT, the next generation DT, which focuses on semantic technologies and evolution. Additionally, Rosen et al. (2019) describes how NexDT can be used to create ecosystems. However, Gallego-García et al. (2022) uses this name to refer to the total set of requirements and services of a DT.

In essence, the idea of an ecosystem stems from biology, where multiple autonomous entities share the same environment (Blew, 1996). This was later extended to the business and software domain. Due to the level of control we have in an overarching system of TEs, we will call this *Twinning Federation*, rather than *Twinning Ecosystems*. Federation takes care of the interoperability of independently developed and evolving TEs, regarding data, ownership management, and orchestration of services. This orchestration is of particular importance when time-management is required as in the case of co-simulation (Gomes et al., 2018).

Marah and Challenger (2025) discuss federation in a Twinning context, providing the necessary background and reasoning for this term. They identify *Vertical Federation* (*i.e.,* DT parents that have DT children), *Horizontal Federation* (*i.e.,* multiple DTs on a same level that need to collaborate in one way or another), and *Hybrid Federation* (*i.e.,* a combination of the two).

Jeong et al. (2022) describes the different technologies required for federation as the next stage in creating better DTs. The authors identify that different technologies might yield different TOs. Furthermore, because the large complexities involved, they state that there should be a step-by-step layered implementation approach.

Arcaute et al. (2021) emphasises that there should be "*different kinds of twins*". The authors identify that each purpose might yield another TO, which need to be organised.

In Oakes et al. (2021), the notion of *DT Constellations* and *DT Slices* is introduced. The former defines the collection of all the functionalities that the overall system is desired to have, and the latter a specific subset of these functionalities. The overall Twinning System can therefore be constructed by combining DT Slices.

In this chapter, these various reasons for combining TEs will be explored and some architectural solutions given.

6.1 Federation of Twinning Systems

As our understanding of Twinning increases, we realise that a single TO is often not sufficient to provide the desired services such as monitoring and dashboarding, anomaly detection, fault diagnosis, what-if analysis, optimisation . . . Oftentimes multiple systems need to collaborate, implying that their TOs should also collaborate. Rather, we need to consider and support the federation of Twinning. This is characterised by different types of relationships between the constituent TEs. Additionally, these constituent TEs and their relationships might evolve over time.

Federation enables data aggregation from multiple sources to form a comprehensive view of the entire system. In the context of Twinning, combining data from various interconnected components or systems makes it possible to monitor and analyse the systems more effectively. For instance, a manufacturing system might have TOs for individual machines. Federation would allow gathering data from all these TEs to understand the overall production process. Thus, according to a specific property, as an example, anomaly detection or fault diagnosis could be applied.

Different kinds of relationships between TEs lead to different dependencies between these TEs. These dependencies determine the requirements for a family of federation architectures. We will hence identify and discuss the most important kinds of relationships between TEs.

6.1.1 Multiple Properties of Interest

Services provided by a TE such as anomaly detection, optimisation, etc. always pertain to specific PoIs. PoIs consider the specific system properties that are of concern to certain stakeholders. PoIs can thus be used in the formalisation of the stakeholders' requirement(s). Typically, different PoIs correspond to different views on a system. Example PoIs are: safety (a Boolean property), energy efficiency (a numerical property), architectural feedback (a structural property) and frequency response curve (a behavioural property). TEs are often constructed specifically to observe and compute a specific PoI within the context of a set of requirements.

One is often interested in more than one PoI. It hence makes sense to combine multiple TEs, each providing a service pertaining to a single PoI. Common examples are the logical combination of multiple Boolean PoIs, and multi-criteria optimisation with the individual criteria provided by individual PoIs. This is shown graphically for the AOs in Figure 6.1. Note that we can also combine TOs with multiple PoIs in the same way.

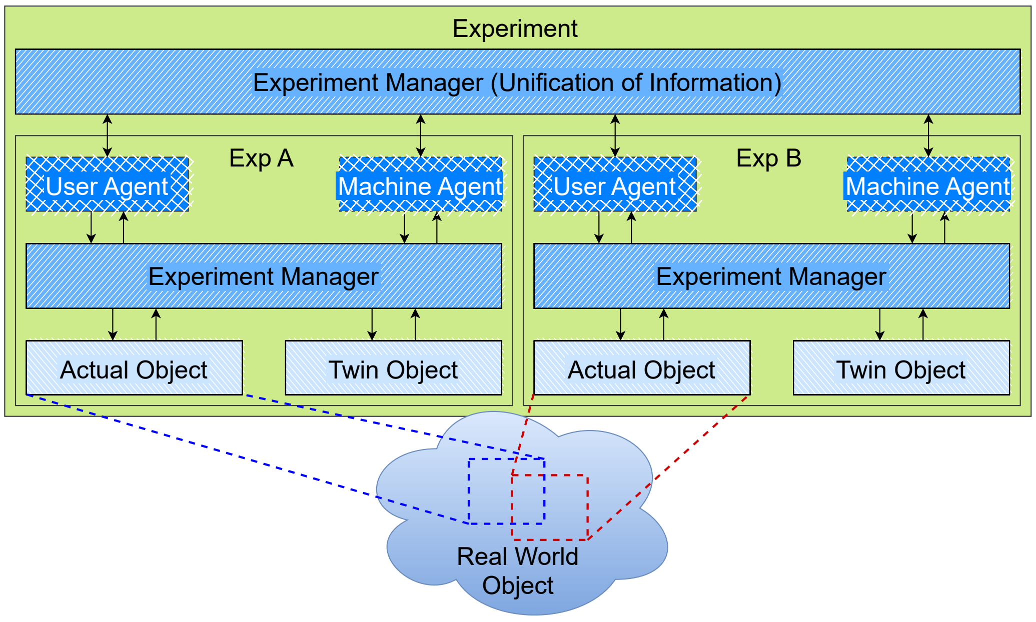


Figure 6.1: Combining TEs with multiple PoIs.

In the figure, two experiments (*Exp A* and *Exp B*) are shown, where each experiment focuses on a different PoI. Their AOs represent the same real-world object, but due to the difference in PoI, they might focus on a different set of sensors, actuators, or components. The TOs are constructed to be aligned with the AO, as well as the PoI that must be analysed. The top EM will likely do a unification of data in order to obtain the desired (potentially emergent) PoIs.

Figure 6.2 provides an example timeline for the execution of multiple TEs with different PoIs. Notice how *Exp A* and *Exp B* are instructed by a higher-level EM, which in its turn communicates the collected data to the top-level *Orchestrator*. Internally, *Exp A* and *Exp*

B both launch their AO and TO, according to the information in their own EMs.

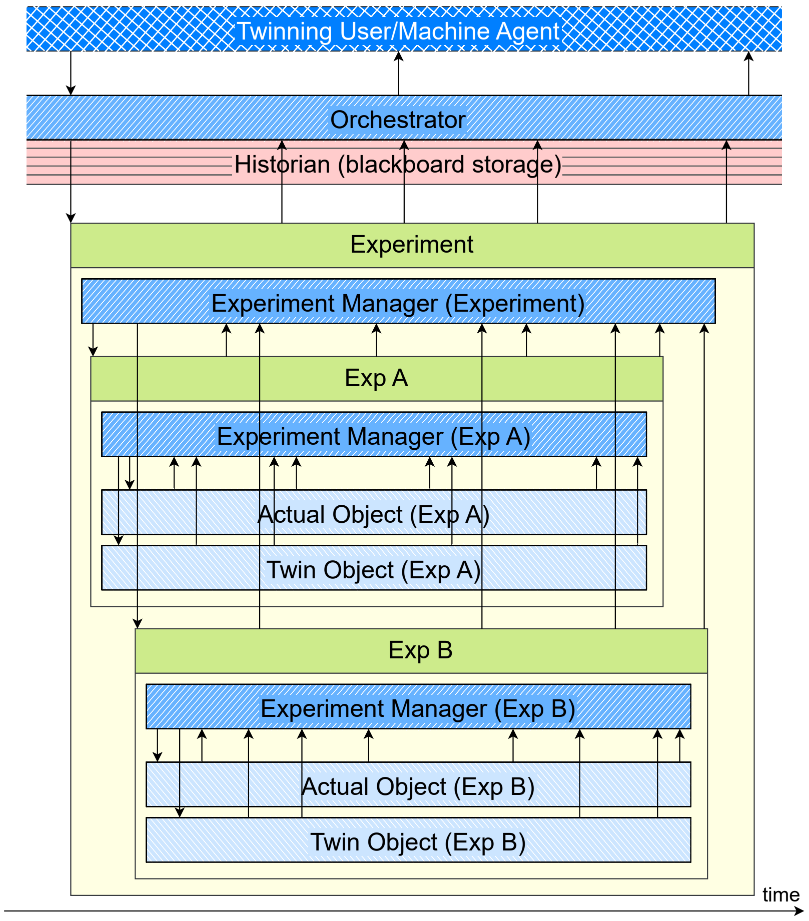

Twinning User/Machine Agent
Orchestrator
Historian (blackboard storage)
Experiment
Experiment Manager (Experiment)
Exp A
Experiment Manager (Exp A)
Actual Object (Exp A)
Twin Object (Exp A)
Exp B
Experiment Manager (Exp B)
Actual Object (Exp B)
Twin Object (Exp B)
time

Figure 6.2: Combining TEs with multiple PoIs timeline.

Also, it is important to notice that the communication rate of *Exp A* and *Exp B* are not related, and may even vary over time. We will assume that the top EM will always use the latest known information from the experiments to execute upon. An example implementation for this behaviour can be found in Algorithm 6.1.

Imagine that we would like to build a TE for the movement of vessels in a port. *Exp A* here can contain a part of the docks and locks, whereas *Exp B* contains the physical movement of the vessels on the water, such that the overall union of *Exp A* and *Exp B* entails the full port. The conceptual architecture presented in Figure 6.1 can then be used to construct this system.

Algorithm 6.1 Pseudocode for EM with multiple PoIs.

```
start $ExpA$ and $ExpB$
while not finished do
    $A \leftarrow$ Collect data from $ExpA$
    $B \leftarrow$ Collect data from $ExpB$
    yield union of $A$ and $B$
end while
```

6.1.2 Multiple Independent TEs

The most simplistic combination of TEs happens when they are in the same environment, but there is no conceptual relationship between them. They might have the same PoIs, but due to their lack of relationship, this does not matter. This often does not happen, given that it makes very little sense to combine *independent* TEs. Figure 6.3 shows this visually for the AOs. The same can happen for TOs: two independent models in the same formalism can co-exist.

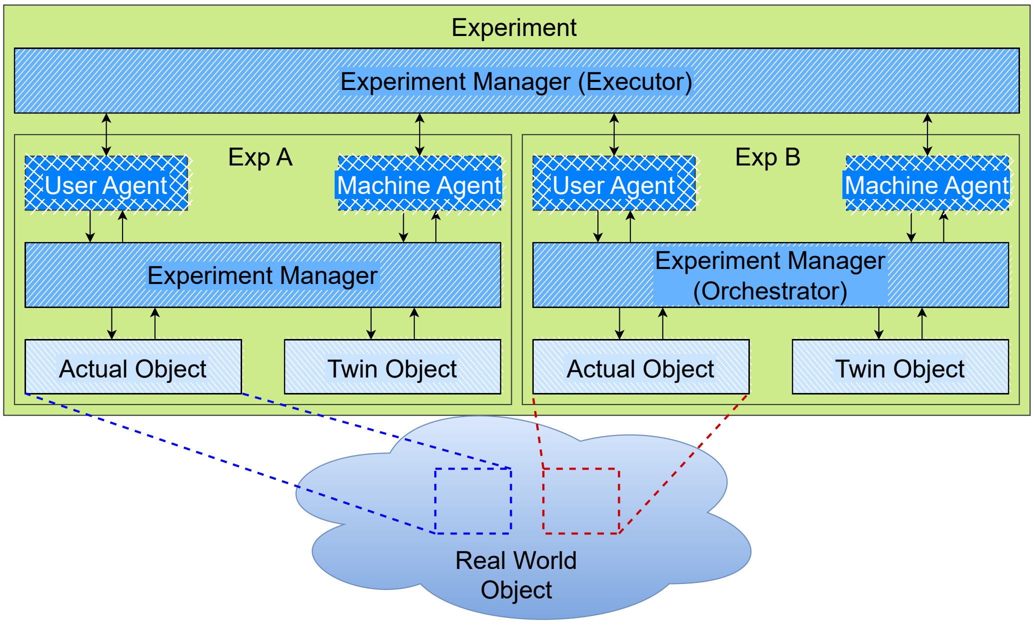


Figure 6.3: Combining independent TEs.

An execution timeline of such a system can look incredibly similar to Figure 6.2. However, given that there is no relationship between *Exp A* and *Exp B*, the top EM launches both experiments and simply collects data. This is also shown in Algorithm 6.2. In an ideal scenario, *Exp A* and *Exp B* run on different threads.

Returning to the port example, we can apply this relationship when *Exp A* and *Exp B* represent ports at the other side of the world from each other, assuming there is no interaction between them.

Algorithm 6.2 Pseudocode for EM for independent TEs.

```
start $ExpA$ and $ExpB$
while not finished do
    $A \leftarrow$ Collect data from $ExpA$
    yield $A$
    $B \leftarrow$ Collect data from $ExpB$
    yield $B$
end while
```

6.1.3 Type vs Multiple Instances

Imagine being a car manufacturer. You create cars and you want to use Twinning to increase their performance. Let's say that some of the cars are being used exclusively on dirt roads, whereas others are used in a city, and even others exclusively on a racing track. It stands to reason that each of these cars evolves differently and should be orchestrated differently. Even though each TE of a car might work on an individual (or instance) level, you might benefit from having a TE of the overall behaviour of the cars. This way, you can analyse general issues, or identify caveats to solve in the next version of the car.

We make the distinction between the Type and the Instance. The Type is the theoretical TE (*e.g.*, the general car itself), whereas the Instance is the individual TE in a specific environment (*e.g.*, the car on the racetrack, or the car on dirt roads). Likely an aggregation of data is required to obtain the necessary information to analyse the Type. This is shown for the AOs in Figure 6.4. There can also be multiple TO instances of the same type.

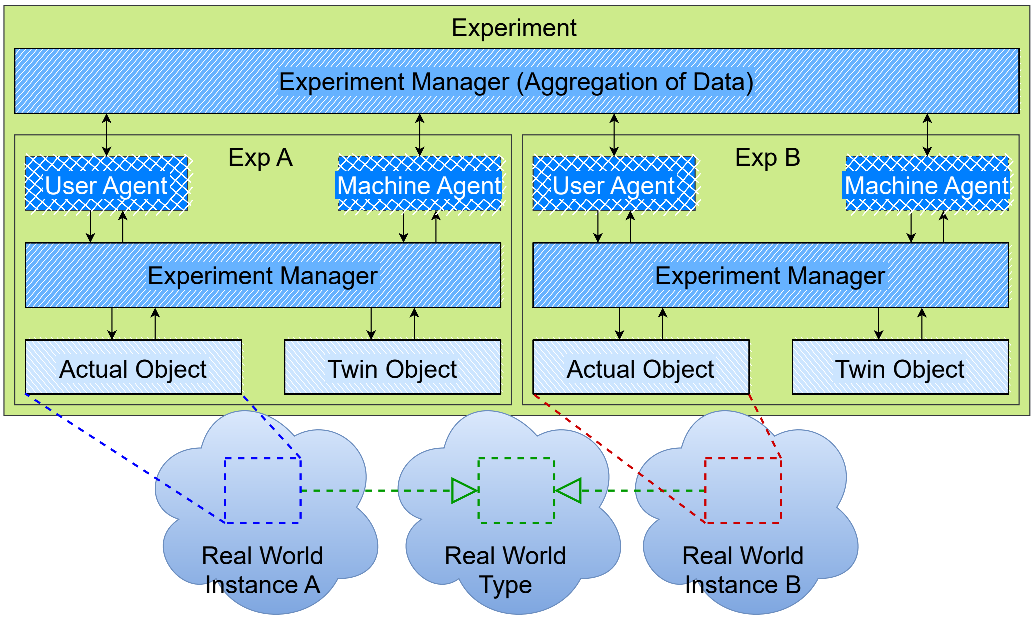


Figure 6.4: Aggregating data from multiple Instances of the same Type.

In the figure, two experiments (*Exp A* and *Exp B*) are *independent* experiments of the same real-world type. However, the environment in which they are used differs (visualised with the different clouds). Behind the scenes, both *Exp A* and *Exp B* are based on the same abstract TE.

Again, the difference here lies in the behaviour of the EM. Figure 6.2 is still a representative example. One major difference is that a window of data from both *Exp A* and *Exp B* needs to be maintained, such that we can aggregate the data (possibly by using a filter), as shown in Algorithm 6.3.

Algorithm 6.3 Pseudocode for EM for instance vs type TEs.

```
start $ExpA$ and $ExpB$
while not finished do
    $A \leftarrow$ Collect data from $ExpA$
    $B \leftarrow$ Collect data from $ExpB$
    $Type \leftarrow$ aggregate($A, B$)
    yield $Type$
end while
```

Tesla aims to have a TO for every build car (H. Zhang et al., 2017), allowing them to do predictive maintenance on an instance level (Alvarez, 2019). However, they likely also aggregate all data for their cars in order to update future models.

Similarly, in our port example, the Type can represent our port and *Exp A* a port in Belgium and *Exp B* a similar port in, for instance, Denmark.

6.1.4 Architectures Connecting Multiple Components

If the AO has structure –or is at least modelled as such– in the form of a network of interacting components, we can construct or re-use the TEs for each of the components. To provide services of the whole composed system, one approach is to coordinate the TOs of the individual components or subsystems. This kind of federation assembles the sub-experiments into a single, composite TE. The top EM encodes how different components interact with each other and with the environment in which the overall system operates. This is visualised in Figure 6.5, where each AO represents a system that *depends* on another. Similarly, there can be multiple TOs that depend on each other, yet are executed separately.

For example, consider applying federation in a smart factory context, which might have TOs for conveyor belts, robotic arms, warehouse control, power consumption, *etc.* In this context, federation enables the TO of the entire factory to analyse how these different aspects of the factory interact and influence each other, based on the TOs of the individual machines. This idea closely relates to co-simulation (Gomes et al., 2018), where multiple dependent models are simulated simultaneously. In fact, co-simulation appears if the TOs are to be combined this way. Visually, a timeline for such an experiment is shown in Figure 6.6.

Algorithm 6.4 shows some example code for the top EM, using a simple iteration loop to synchronise *Exp A* and *Exp B*. Different co-simulation techniques might yield better results.

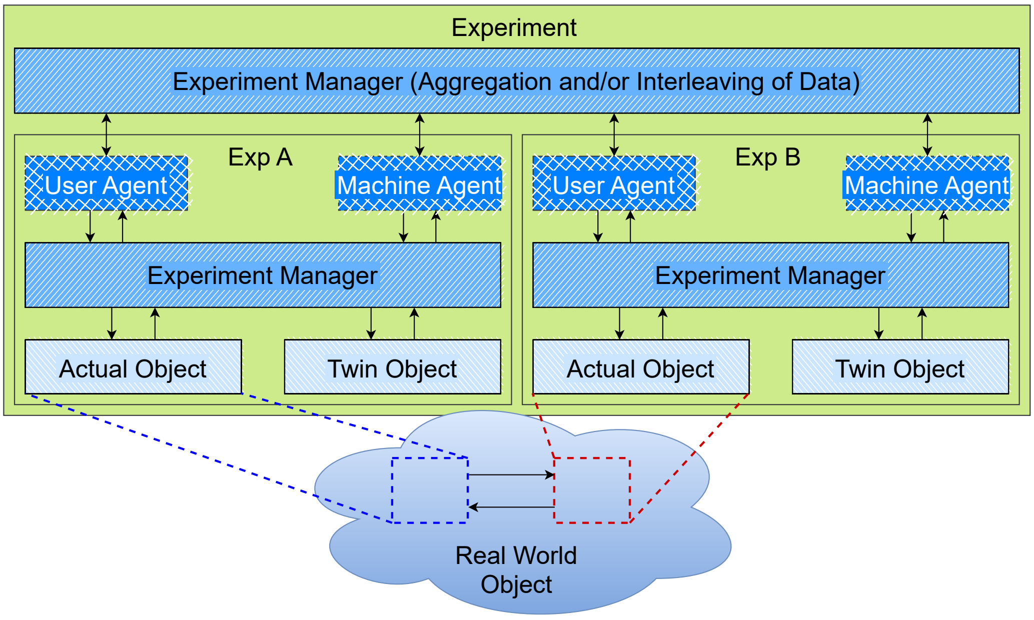


Figure 6.5: Aggregating data from multiple dependent TEs.

Algorithm 6.4 Pseudocode for EM for dependent TEs.

```
start $ExpA$ and $ExpB$
while not finished do
    while fixed point not reached do
        $A \leftarrow$ Collect data from $ExpA$
        Send $A$ to $ExpB$
        $B \leftarrow$ Collect data from $ExpB$
        Send $B$ to $ExpA$
    end while
    yield aggregate$(A, B)$
end while
```

The port in Antwerp is a port that exists both on the left bank, as well as on the right bank. When we have a separation of the banks in terms of simulation (*e.g.*, because of political reasons), it might be the case that *Exp A* represents the left bank and *Exp B* the right bank. The interaction between them is the movement of vessels, as well as a potential coordination of pilots and tugboats.

6.1.5 Multiple Formalisms/Languages

Even a single PoI may be obtained in many different ways such as a lookup in historical data traces or through simulation of a set of differential equations. The language to use is determined by its availability, computational performance constraints, or cognitive explainability. This type or formalism heterogeneity in TEs thus also needs to be man-

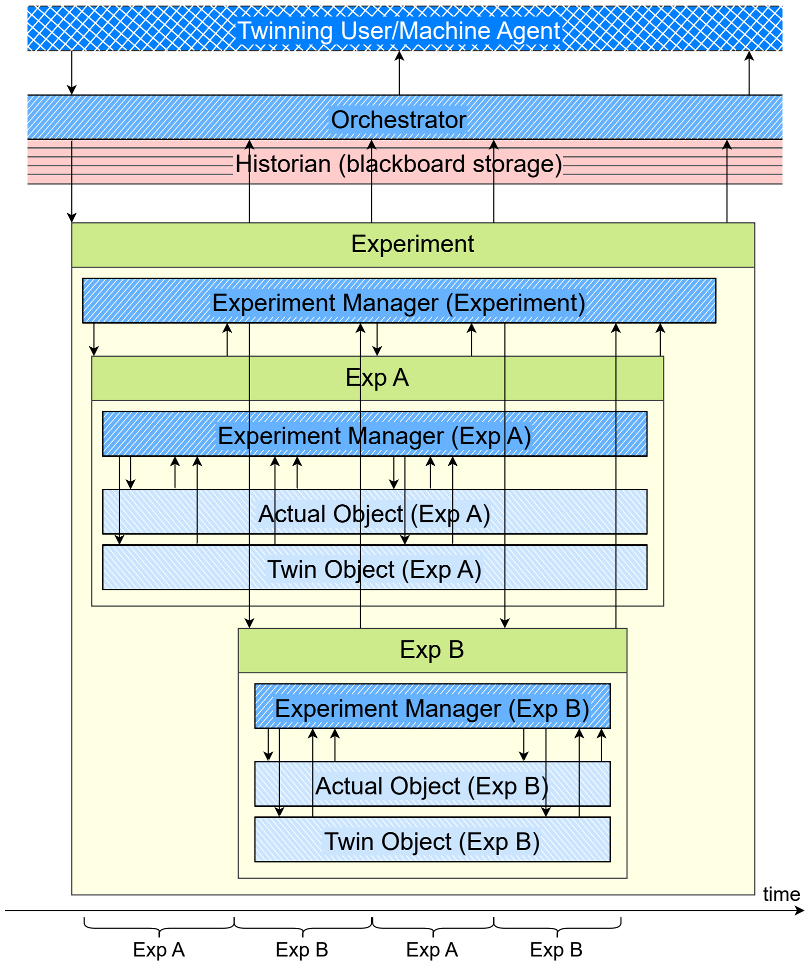

Twinning User/Machine Agent
Orchestrator
Historian (blackboard storage)
Experiment
Experiment Manager (Experiment)
Exp A
Experiment Manager (Exp A)
Actual Object (Exp A)
Twin Object (Exp A)
Exp B
Experiment Manager (Exp B)
Actual Object (Exp B)
Twin Object (Exp B)
time
Exp A
Exp B
Exp A
Exp B

Figure 6.6: Combining dependent TEs timeline.

aged, through federation. This may require various kinds of adaptation and in particular semantic adaptation (Denil et al., 2015).

Figure 6.7 shows this, where two experiments focus on the exact same AO, but have different TOs. In reality, the AOs do not need to be an exact match, but should at most differ in PoIs. The top EM should interpolate the information from all experiments and potentially interleave execution. On the TO's side, we are basically doing co-simulation, as visualised in Figure 6.6.

The most simplistic implementation would be a simple aggregation of the data, as shown in Algorithm 6.5.

The safety of the trajectories in a port might be modelled using Petri-Nets, whereas the overall movement could be constructed using a Discrete-Event simulation like DEVS (Zeigler et al., 2018).

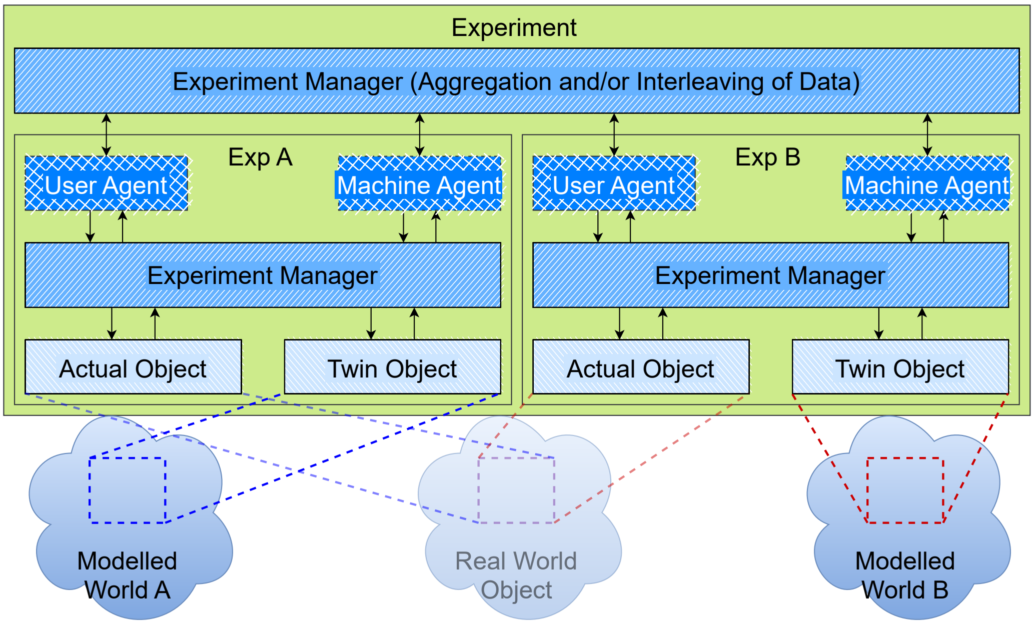


Figure 6.7: Combining TEs with multiple formalisms.

Algorithm 6.5 Pseudocode for EM with multiple formalisms.

```
start $ExpA$ and $ExpB$
while not finished do
    $A \leftarrow$ Collect data from $ExpA$
    $B \leftarrow$ Collect data from $ExpB$
    yield aggregate$(A, B)$
end while
```

6.1.6 Multi-Abstraction/Detail/Fidelity

Multiple TOs are often constructed for a single (sub-)system, providing services pertaining to a single PoI only. These TOs only differ in their level of abstraction, detail, and fidelity. Franceschini, Challenger, et al. (2019) and Franceschini, Van Mierlo, and Vangheluwe (2019) showed the usefulness of using multiple abstractions for the same system by simulating car traffic, with individual cars at a low level of abstraction, and traffic jams at a higher level of abstraction. Similarly, Badawi et al. (2021) collects data for cities at a macro and a micro level.

Substitutability of a lower abstraction TO by a higher abstraction TO, while the PoI remains unaltered, is the main goal when using multiple abstractions. Often, but not necessarily, lower abstraction is a result of higher detail, at the cost of more computation power required. A higher abstraction might be computationally inexpensive, but results in mere approximations of the real scenario. A distance metric for the PoI is then required to ensure validity of the TO.

The need for (real-time) performance often drives the choice of abstraction, detail, and fidelity in a TE. This can also be impacted by cognitive performance. Some questions are namely best answered at a particular abstraction level. For instance, the user may not be able to comprehend too much detail. Aggregate information is often easier to use as a basis for logical reasoning and decision-making.

The level of detail, abstraction, and fidelity may be dynamically changed, adapting to a changing system, environment or to (simulation) performance needs (Franceschini, Van Mierlo, & Vangheluwe, 2019). This requires transitioning between different TOs. This can again be achieved through an appropriate federation architecture where an orchestrator takes care of the transitioning.

Conceptually, this is the same as Figure 6.7. However, Figure 6.8 provides a little bit more detail on the relationship between the TOs. Furthermore, the top EM has to select the correct experiment to execute, depending on the choices of adaptive abstraction, as opposed to aggregating the data.

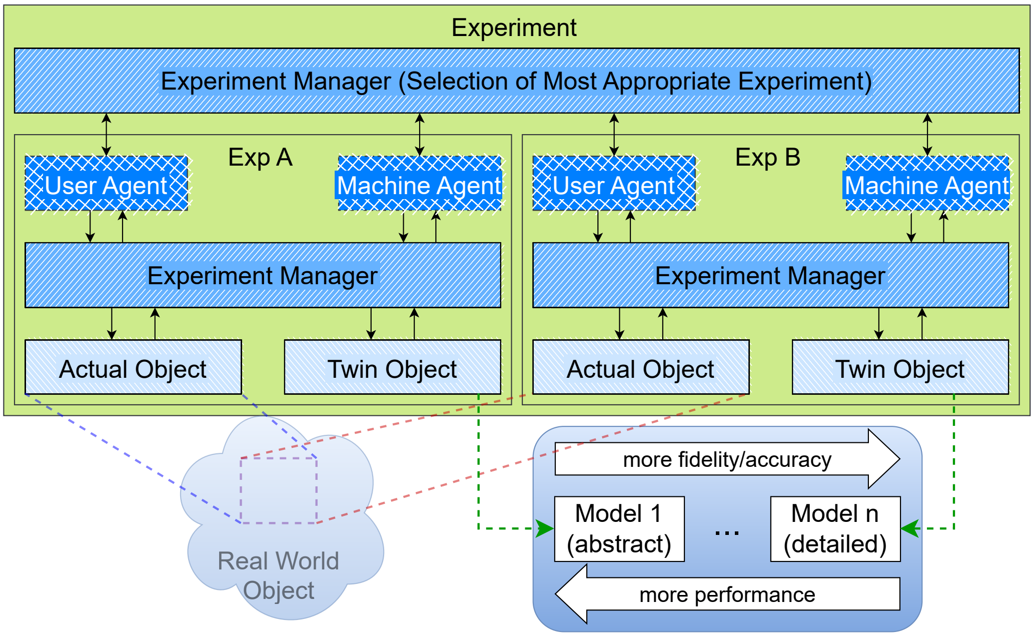


Figure 6.8: Combining TEs with multiple levels of abstraction.

In Figure 6.9, it is shown what a timeline that applies adaptive abstraction may look like. First, *Exp A* is called to execute the system according to the abstraction level A. After a while, *Exp B*'s abstraction level needs to be used (*e.g.*, due to more efficiency, or more accuracy). Then, after a while, *Exp A* is to be used once more. This is also shown in Algorithm 6.6. Depending on the level of abstraction required, a different result is obtained.

Just like it is explained by Franceschini, Van Mierlo, and Vangheluwe (2019) with cars and traffic jams, a similar approach can be taken for water traffic. The main points of congestion will be when vessels are waiting at the locks.

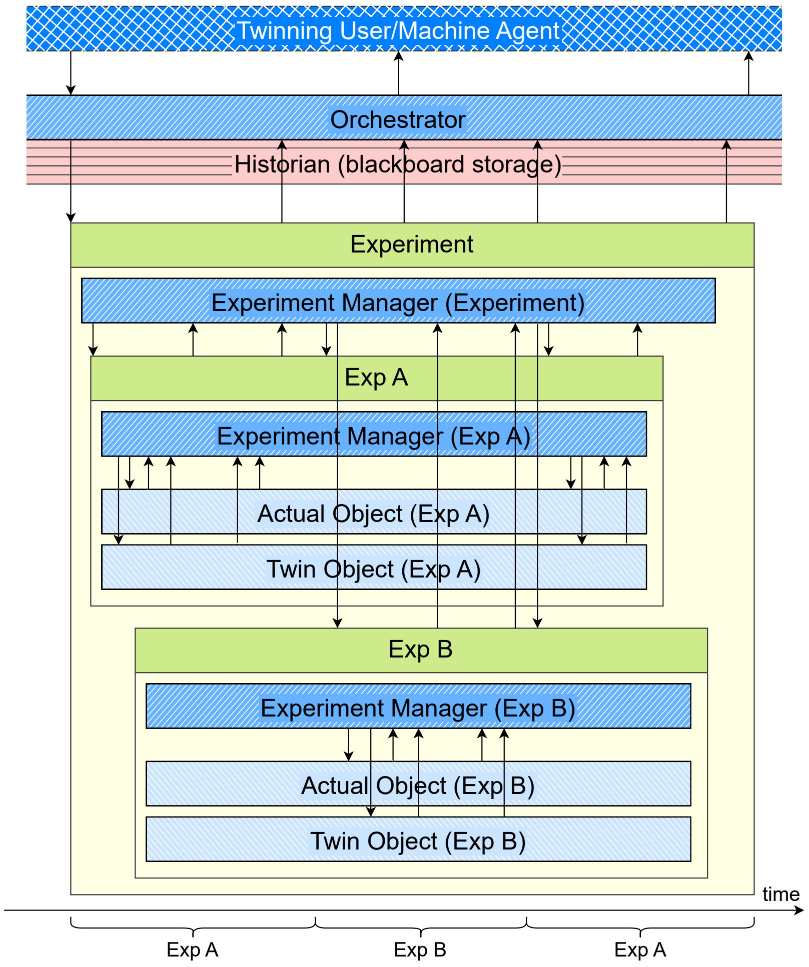

Twinning User/Machine Agent
Orchestrator
Historian (blackboard storage)
Experiment
Experiment Manager (Experiment)
Exp A
Experiment Manager (Exp A)
Actual Object (Exp A)
Twin Object (Exp A)
Exp B
Experiment Manager (Exp B)
Actual Object (Exp B)
Twin Object (Exp B)
time
Exp A
Exp B
Exp A

Figure 6.9: Combining TEs with multiple states timeline.

Algorithm 6.6 Pseudocode for EM with multiple abstractions.

```
start $ExpA$ and $ExpB$
while not finished do
    $Mode$ ← Determine level of abstraction
    if $Mode$ is "A" then
        $A$ ← Collect data from $ExpA$
        yield $A$
    else if $Mode$ is "B" then
        $B$ ← Collect data from $ExpB$
        yield $B$
    end if
end while
```

6.1.7 Multiple Life-Cycle Stages

TOs can be used throughout the entire life of an AO. This may cover different PLM stages such as design, manufacturing, assembly, operation, maintenance, refurbishing, *etc.* of a product with intricate control flow between those stages, often including feedback loops at different time scales. As drastic changes occur between different life-cycle stages, changes need to be made to TOs too. Note that those changes may actually be what defines the stages, as an emergent property. There may be changes in PoIs, in components or architecture, and in abstraction, detail, or fidelity. A federation architecture can manage these, as shown in Figure 6.10. Conceptually, the same happens as in Figure 6.7 and Figure 6.8, only here, the relationship of the models is dependent on the exact life-cycle stage of the AO.

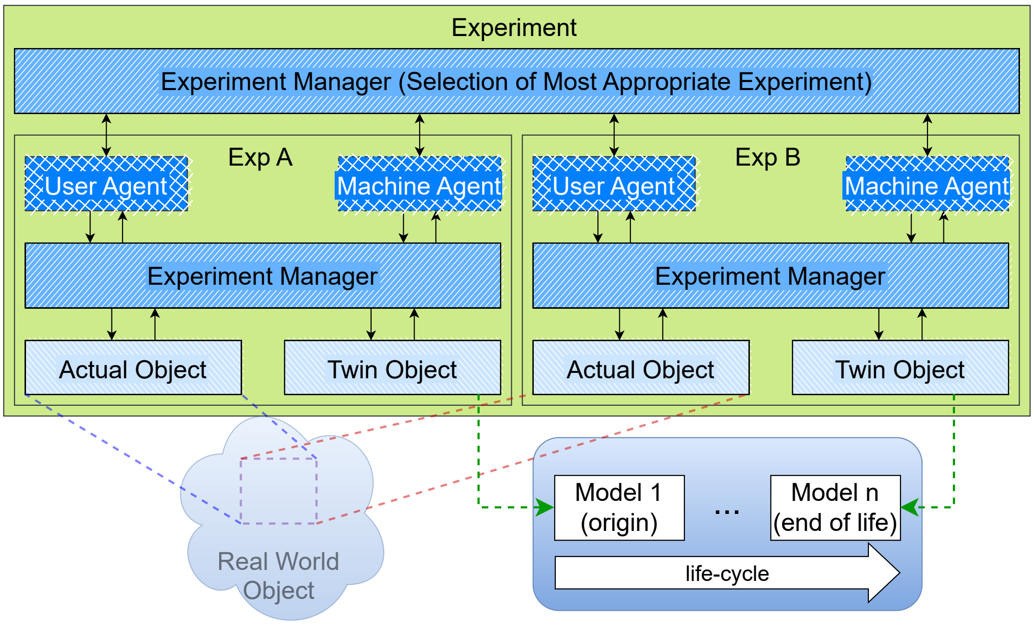


Figure 6.10: Combining TEs with multiple life-cycle stages.

Additionally, an AO always evolves over time. For biological systems this is clearly a continuous process, as plants keep growing and cells keep multiplying. But even in industrial contexts, we can identify wear and tear, and replacement of parts. Ensuring this often coincides with a predictive maintenance PoI requirement. In this sense, we might identify more stages, such that the full validity range (Van Acker et al., 2024) of all these stages covers the full behaviour of the AO.

Note the behavioural similarities to the timeline in Figure 6.9. First, *Exp A* defines the behaviour of the system, but after a while, the model is not accurate anymore, so *Exp B* needs to be used. Ideally, there is a clean transition between the two. Next, maybe another experiment is needed for the next phase. Note that the timeline shows returning to *Exp A*, which is not impossible, but will happen rarely. The example given in Algorithm 6.7 also shows the selection of specific states in the life-cycle.

Algorithm 6.7 Pseudocode for EM with multiple life-cycle stages.

```
start $ExpA$ and $ExpB$
while not finished do
    $State \leftarrow$ Determine life-cycle stage
    if $State$ is "A" then
        $A \leftarrow$ Collect data from $ExpA$
        yield $A$
    else if $State$ is "B" then
        $B \leftarrow$ Collect data from $ExpB$
        yield $B$
    end if
end while
```

In Wagle et al. (2023), multiple states are used to identify the current behaviour of the battery of an electric vehicle. This is a very rudimentary implementation of this kind of federation, and a similar approach can be applied to electric yachts. On a higher level, a full port can change in business throughout the year, which might result in a different executional behaviour.

6.1.8 Multiple Copies for Redundancy

In complex systems, *redundancy* is essential to maintain reliability and availability. Federation can contribute to redundancy by allowing the creation of multiple TOs for the same AO. For instance, a simulation model can be deployed in the cloud (because of the desired computation power), but another model can run at the edge for redundancy (as a backup, in case the connection is down)[1]. Redundancy ensures that alternative TEs can continue executing critical processes or providing services and backup for those encountering issues. This is particularly valuable for critical systems where downtime can have severe consequences. However, in such a case, some requirements and criteria should be met to deploy a valid and high degree of redundancy between different

[1] In Stage Ⓐ, you identify that you require redundancy, which yields this combination at a conceptual level in Stage Ⓑ, and then Stage Ⓓ is able to deploy one in the cloud, and one on the edge.

TEs. The management of multiple copies of a same TE can be achieved by a federation architecture, as shown in Figure 6.11

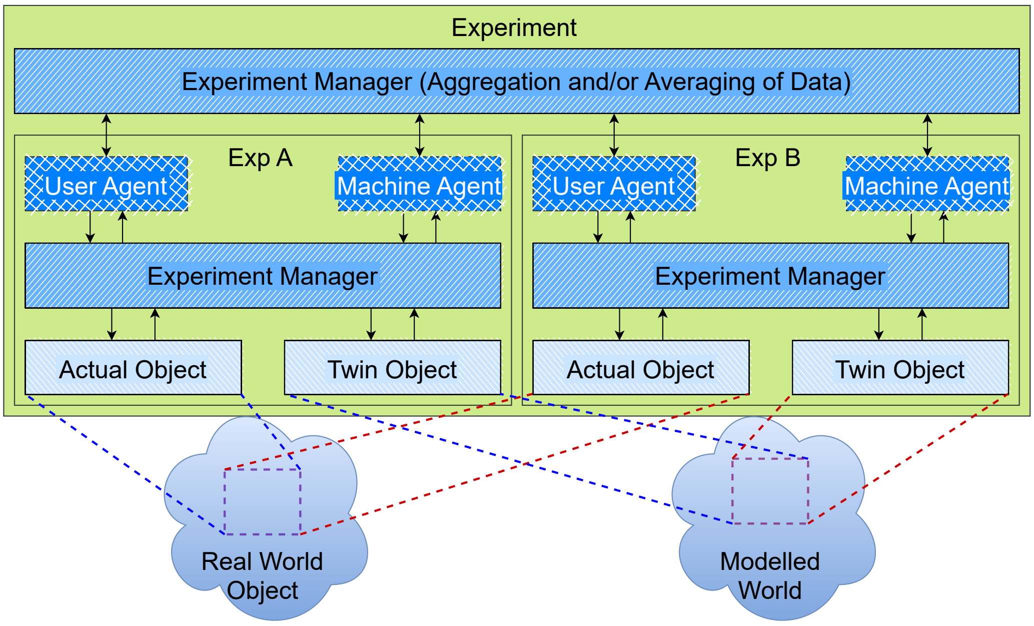


Figure 6.11: Combining TEs with redundancy.

Again, the timeline in Figure 6.2 can be used as an example execution. There is no requirement that both experiments communicate at the same rate, but the top EM will work best if both *Exp A* and *Exp B* are at least synchronised in a way. Algorithm 6.8 shows an example implementation in the top EM for accounting for multiple copies. Of course, more complicated implementations (for instance by using majority voting, or time-series analysis) are also possible.

Algorithm 6.8 Pseudocode for EM with multiple copies.

```
start $ExpA$ and $ExpB$
while not finished do
    $A \leftarrow$ Collect data from $ExpA$
    $B \leftarrow$ Collect data from $ExpB$
    yield average$(A, B)$
end while
```

In terms of completeness, we can run two similar simulations of a port to obtain an average result.

6.1.9 Multiple Combinations of Multi-*

As of now, this section has described different ways of combining TEs, focusing on specific situations. In reality, many of the presented approaches will likely need to be combined, making the top-level EM more complicated, when there are more different combinations.

Because of the holonic nature of the presented architecture, these combinations may happen as shown in Figure 6.12. Here, three experiments (*Exp A*, *Exp B* and *Exp C*) are combined in the over-arching experiment *Exp A-C*. Similarly, the experiments (*Exp D*, *Exp E* and *Exp F*) are combined in the over-arching experiment *Exp D-F*. *Exp A-C* and *Exp D-F* might themselves need to be combined according to another method.

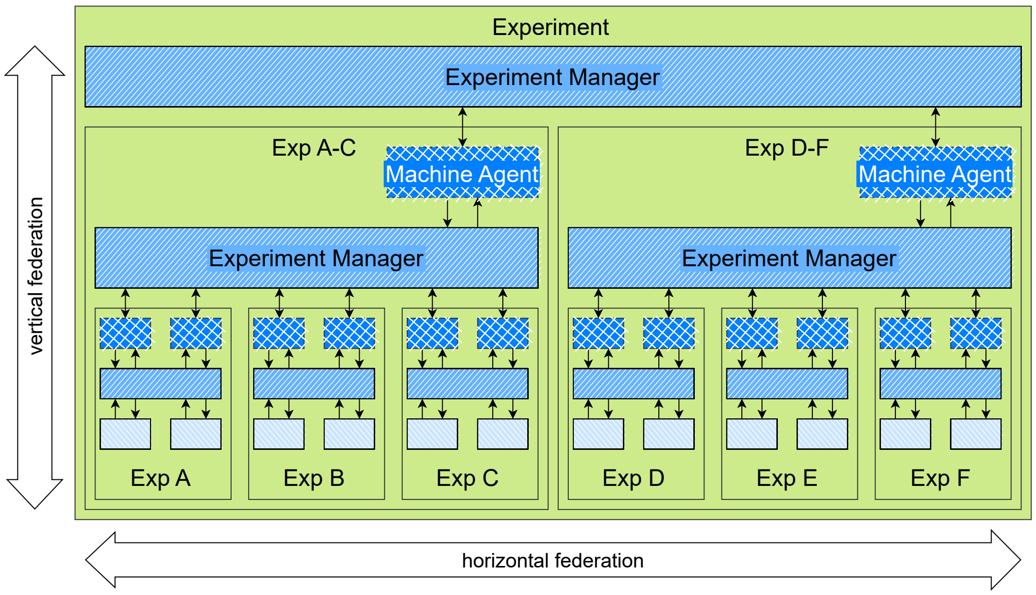


Figure 6.12: Hierarchically combining TEs.

We can keep hierarchically combining TEs using this approach. The TEs on the same hierarchical level (*i.e.*, experiments *Exp A*, *Exp B* and *Exp C*; or experiments *Exp D*, *Exp E* and *Exp F*) can be *horizontally federated*, whereas we can call this hierarchical composition *vertical federation*. The combination of the two can be considered *hybrid federation* (Marah & Challenger, 2025).

Figure 6.13 shows an example timeline of how the interaction between experiments (and their managers) may happen. Notice that there is no requirement that all experiments start and end at the same time.

The topmost EM's behaviour is dependent on how *Exp A-C* and *Exp D-F* relate with each other. All previously presented solutions apply here. In a similar vain, we can deconstruct the behaviour of the port into its smallest components, ensuring this hierarchical conceptual architecture for its TEs.

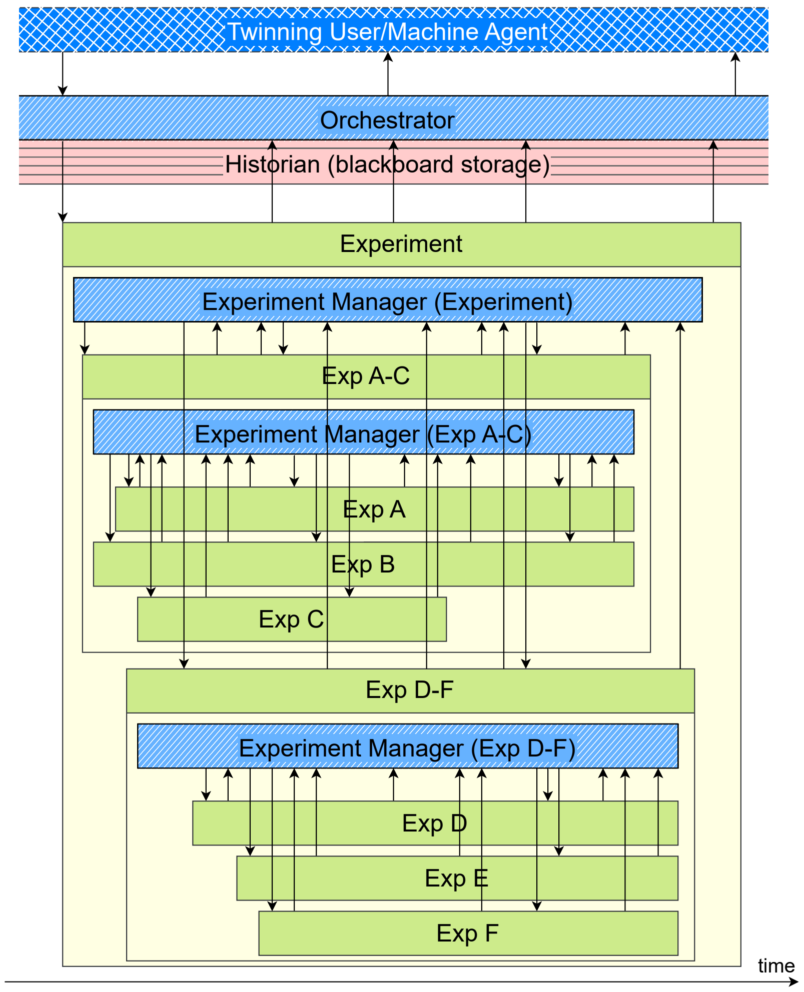

Twinning User/Machine Agent
Orchestrator
Historian (blackboard storage)
Experiment
Experiment Manager (Experiment)
Exp A-C
Experiment Manager (Exp A-C)
Exp A
Exp B
Exp C
Exp D-F
Experiment Manager (Exp D-F)
Exp D
Exp E
Exp F
time

Figure 6.13: Example timeline for hierarchically combining TEs.

6.2 White Box vs. Black Box Combinations

The presented architecture and its combinations and variations are still purely conceptual. They are meant to help define the required components and parts in a Twinning System, but in practice, it will be inefficient and impractical to work with.

Whenever the exact contents of the AOs or the TOs are known, we can apply a white box combination. An example is visualized in Figure 6.14. There is an AO with two sensors (*Sensor A* and *Sensor B*) and one actuator (*Actuator 1*). Another TE (with a different PoI) has an AO with *Sensor B*, *Sensor C*, *Actuator 2* (notice the duplicate use of *Sensor B*). All sensors and actuators are on the same physical device, yet because of our conceptual separation into multiple TEs (due to multiple PoIs), they are conceptually separated. In practice, we want to have a single AO that contains all sensors and all actuators. A white box combination merges both experiments into a TE with a larger AO (*i.e.*, the union of all sensors and actuators).

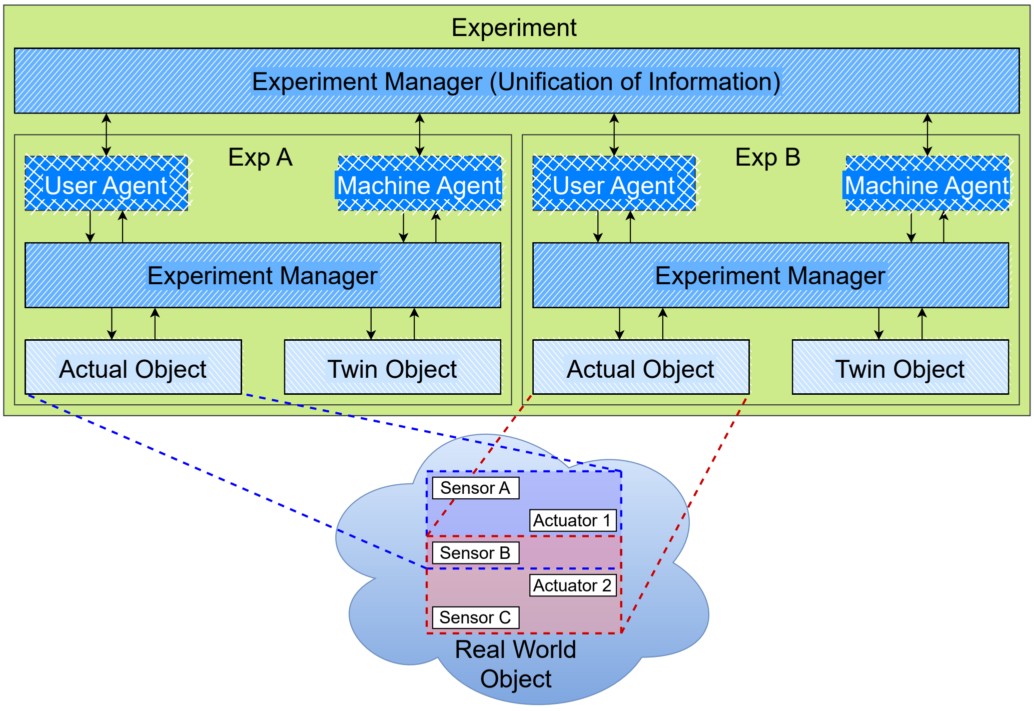


Figure 6.14: Combining two White Boxes through merging the contents.

For instance, for the port of Antwerp, there might be one TO focusing on the left bank and another that focuses on the right bank. From a simulation point of view, it will be much more efficient to merge these into a single simulation.

Similarly, Twinning systems with multiple life-cycle stages (as shown in Figure 6.10) will likely be optimised, such that the TO consists of a state machine that selects the correct life-cycle stage.

If the AOs or TOs are obtained via a third party (API, FMU, *etc.*), it is likely not possible to do combinations on this level. They are commonly black boxes here, due to IP protection. Because the exact details are not known, these boxes cannot be combined. Hence, co-simulation techniques can be used to try and merge these black boxes. This is shown in Figure 6.15. The same TEs as for the white box example are used, only this time, we have no access to the individual components, only their interfaces. Merging them requires the black boxes to co-exist, albeit orchestrated by another component.

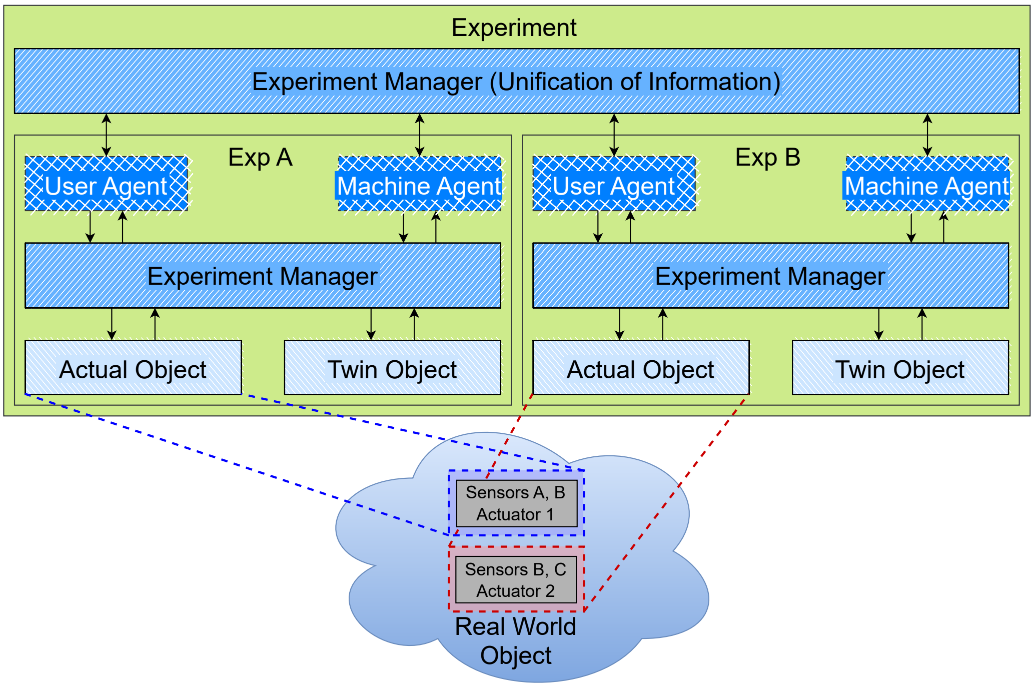


Figure 6.15: Combining two Black Boxes through orchestration.

In general, it is likely that all EMs will be merged or interleaved into a single EM. This optimisation step needs to happen before the actual deployment can be realised, as it will (often) reduce the workload and the required technologies.

6.3 Conclusion

Research Question 5 stipulates *"How can we combine multiple DTs into a larger system?"* Following the notion of a TE as a first-class entity in our conceptual architecture, this chapter answers this question by focusing on combining multiple TEs. It makes use of the architecture presented in Chapter 4 and goes a step further by not only focusing on a single goal (from stage Ⓐ), but also showing how we can construct a Twinning System when multiple goals are selected. Different TEs to be combined may correspond to different requirements, goals and PoIs; may correspond to different components in an architecture; may represent a system at different levels of detail, abstraction, and fidelity; may be used at a type or aggregate level, or at instance level; . . . These different cases were explored and for each a conceptual architecture was proposed. This combination of TEs will especially be considered further in the TurtleBot use-case from Chapter 7.

"Never tell me the odds."

– Han Solo (Harrison Ford),
Star Wars: Episode V - The Empire Strikes Back

Chapter 7 Representative Use-Case: Automated Guided Vehicle

Automated Guided Vehicles (AGVs) are often used in an industrial context as a replacement for the assembly line (Custodio & Machado, 2020). They are self-driving vehicles that are used to move materials, tools and sub-assemblies from one location to another, without taking up as much space (Wu & Ge, 2019).

A Line Following Robot (LFR) is a simple version of an Automated Guided Vehicle (AGV) with the sole purpose of following a line on the ground. In some contexts, this line might be an invisible electromagnetic strip, or a virtual path in an internal GPS. Because of their simplicity, LFRs are often used for Science, Technology, Education, (Arts,) Mathematics (STE(A)M) educational purposes. Some examples include (but are not limited to)[1]:

- LEGO Mindstorms/Inventor:
  https://education.lego.com/en-us/lessons/mindstorms-ev3/line-detection/
- Edison Robot:
  https://meetedison.com/
- Dekimo Challenge:
  https://www.dekimo.com/nl/challenge/
- BlueBot 4-in-1:
  https://www.homesciencetools.com/product/bluebot-4-in-1-educational-robotics-kit/?aff=SB1

This chapter will apply the stages presented in the previous chapters on two practical use-cases: a simple LEGO LFR, and a more industrial case of a TurtleBot[2]. Both can be seen as an approximation for actual AGVs. The LEGO LFR is meant to be a proof-of-concept for the overall Twinning idea. Its end goal is to construct a Digital Shadow (cfr. Kritzinger et al. (2018)'s definition) that is able to accurately follow a line drawn on the ground.

The TurtleBot use-case focuses more on how multiple TEs could be combined in order to yield a valid Twinning System, using the logic presented in Chapter 6. There is an

[1]All urls were last accessed April 22nd 2024.
[2]https://www.turtlebot.com/, accessed July 15th 2024.

additional focus on how to use the *Historian* as a blackboard knowledge base for the system, allowing it to become actual DT (cfr. Kritzinger et al. (2018)).

7.1 LEGO Line Following Robot

As a proof-of-concept for Twinning, we need to design and realize a LFR whose goal it is to follow a line on the ground as closely as possible whilst moving swiftly, economically and safely. In the digital world, a simulated copy (*i.e.,* a TO) is created, acting on the same input from the environment as the real system. The real and the simulated robot should follow the exact same trajectory. Through comparison of the two trajectories, inconsistencies in the operation of the robot can be detected. Ideally, the position (and heading) of the AO and the TO should be sufficiently close at all points in time, independent of the line to follow. Figure 7.1 shows a PM that can be used to construct such a system.

Note that this is a representative workflow for this system, but may evolve over time. Additionally, the developed framework and architecture may change as well. Any implementation of the FTG+PM must support this kind of *evolution* (Meyers & Vangheluwe, 2011).

Furthermore, this workflow will result in a deployed system and a behaviour trace, on which some analysis may be done, potentially yielding insights to be used as input in the development of a new version of this robot.

7.1.1 System Design

The *System Design* and (implicit) *Requirement Elicitation* activity returns to the Feature Trees from Chapter 3. We identify the need for *Data Recording*, *Consistency Monitoring*, *Dashboarding*, and *Live 2D Animation*. While we can select many other goals, this chapter will mainly focus on building a Twinning System that includes but these four goals.

7.1.2 System Decomposition

Next, the (hierarchical) *System Decomposition* activity focuses on the conceptual system architecture (see Chapter 4). Our conceptual system architecture will look a bit like the one shown in Figure 7.2. The AO drives around in its environment, and a TO is mimicking that behaviour. Optionally, the TO can be given additional control signals to synchronize better with the AO. The User Agent is a dashboard that incorporates a plot for the robots, as well as a video stream of the current execution. This way, a user can identify the consistencies between the AO and the TO.

Additionally, both the AO and the TO are decomposed into three parts: some plant equations, a control algorithm, and a design sketch. We will first focus on the AO, but build the models and the logic such that the TO can use them as well.

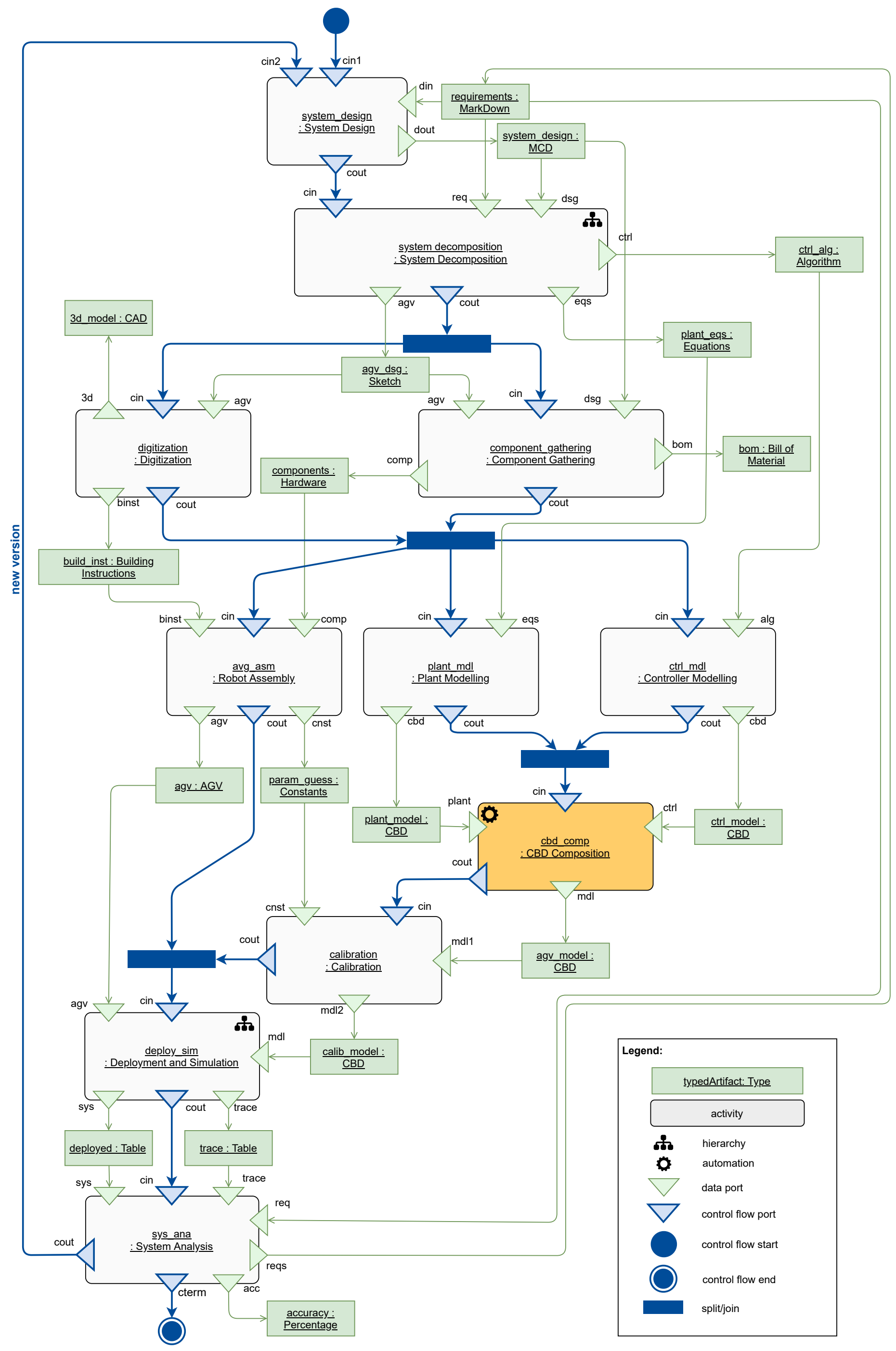


Figure 7.1: Workflow for the LFR use-case.

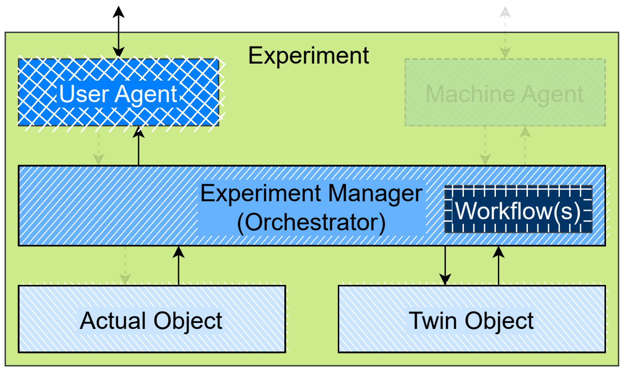


Figure 7.2: Conceptual architecture for the AGV use-case.

7.1.3 Component Gathering

We used the (retired) *LEGO Mindstorms EV3 Core Set* (313131, https://www.lego.com/nl-be/product/lego-mindstorms-ev3-31313) to construct our LFR. While the hardware lacks in precision, its variability and simplicity of construction and adaptation made it a good choice for the purposes of this research. Other literary works have a similar reasoning to use LEGO for their use-cases (Karaduman et al., 2023; Lugaresi et al., 2020; Muñoz et al., 2021).

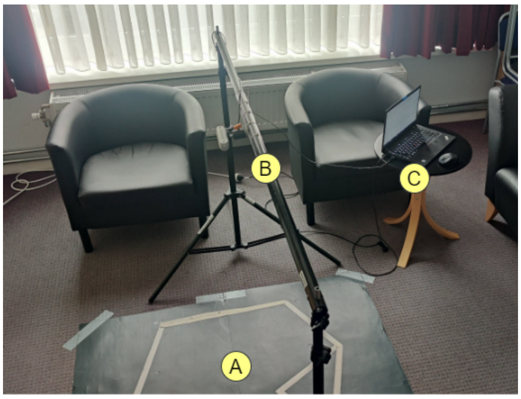
Besides internal odometry, it is important to have an idea of the exact position of the robot. For this, a testing harness has been built. Above the line to follow, a camera is mounted to objectively measure the exact position of the robot. The camera is mounted such that the camera's field of view is able to capture the entire driving range. Care must be taken when mounting the camera to a rig, as the USB-C port for data communication and power is located at the same side as the tripod connector.

In Figure 7.3a, the experimentation setup is shown. This was built in a living room due to the Covid lockdown. Ⓐ is the floor on which the robot drives around, following a white line. Ⓑ is an H-bar, commonly used to attach photography backdrops. Figure 7.3b gives a close-up of the camera mount. Finally, Ⓒ is the laptop on which the control software runs.

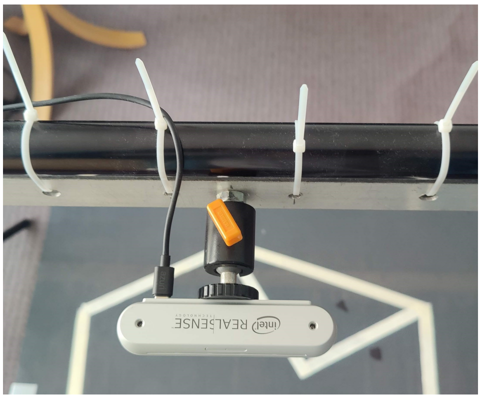
7.1.4 Digitization

To ensure repeatable experiments, a 3D version of this experiment setup was also constructed in Unity3D. For each version of the robot, a CAD model was created and imported into this digital world. Figure 7.4 shows the Unity3D environment running on the server in room M.G.330, at the University of Antwerp. Figures 7.5 and 7.6 show a screenshot of the Unity3D environment itself.

Because LEGO was used to construct this robot, the CAD model was constructed with



(a) Components in the setup.



(b) LFR topdown camera close-up.

Figure 7.3: LFR experimentation setup.

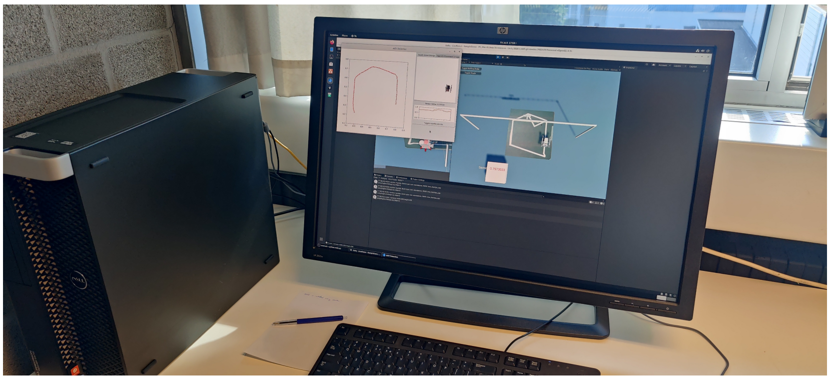


Figure 7.4: Unity3D version of the robot, running on the server in room M.G.330 at the University of Antwerp.

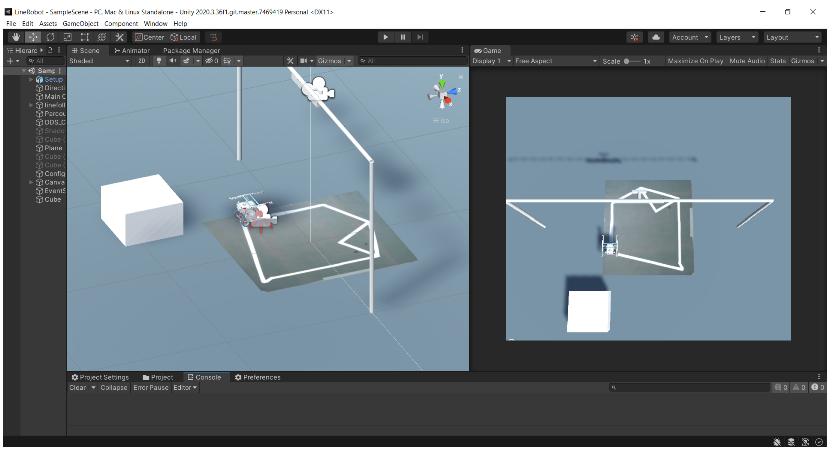

LineRobot - SampleScene - PC, Mac & Linux Standalone - Unity 2020.3.36f1.git.master.7469419 Personal <DX11>
File Edit Assets GameObject Component Window Help
Scene
Animator
Package Manager
Game
Display 1
Free Aspect
Scale
Maximize On Play Mute Audio Stats Gizmos
Account
Layers
Layout
Project Settings
Project
Console
Preferences
Clear Collapse Error Pause Editor

Figure 7.5: Unity3D version of the robot.

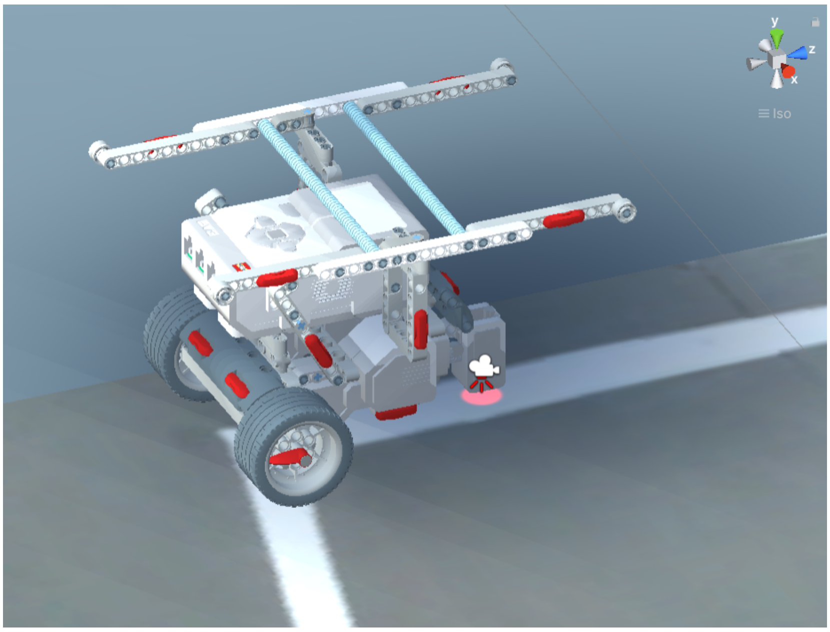


Figure 7.6: Closeup of the Unity3D robot.

LeoCAD[3], which integrates with LDraw[4] for creating LEGO-esque building instructions for these robots.

7.1.5 Robot Assembly

Given the bricks from the *LEGO Mindstorms EV3 Core Set* (313131), together with the bricks from the *LEGO Mindstorms EV3 Education Expansion Set* (45560); and the created building instructions, the actual robot can be built.

7.1.6 Plant Modeling

Wheeled mobile robots are commonly classified as either *omnidirectional* (also known as *omniwheels*, *mecanum wheels*, or *Swedish wheels*), or *nonholonomic* (Lynch & Park, 2017). The main difference between the two is that omniwheels allow for sideways movement, whereas nonholonomic wheels are the more conventional wheels that can be found on cars or bikes.

For the purposes of this research, we will be focusing on a wheeled, nonholonomic *differential-drive* robot (Rajamani, 2011), as shown in Figure 7.7.

The differential-drive behaviour emerges from having two equal wheels of radius r that are driven by their own motors. Conceptually, both wheels are aligned on the same axis,

[3]https://www.leocad.org/
[4]https://ldraw.org/

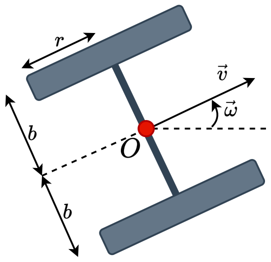


Figure 7.7: Free-body diagram of a differential-drive robot.

at a distance of $2b$ apart (with b half the axle length). The position $O = (x, y)$ is located at the centerpoint between the wheels (*i.e.,* the distance from O to any of the wheels is b). To prevent the robot from tipping over, there can be some additional caster wheels, ball casters or low-friction sliders. Its linear velocity is identified with v and the heading with ω. Thus both wheels can rotate at a different angular velocity, respectively ψ_L and ψ_R for the left and right wheels. To rotate the robot clockwise, (*i.e.,* $\omega < 0$), we should ensure $\psi_R < \psi_L$. Similarly, to rotate the robot anticlockwise, (*i.e.,* $\omega > 0$), we should ensure $\psi_R > \psi_L$.

7.1.6.1 Simple Plant

Given $\langle \dot{v}, \dot{\omega} \rangle$, it is mathematically possible to obtain $\langle \dot{\psi_L}, \dot{\psi_R} \rangle$. We use Newton's notation for time-based derivatives:

$$\dot{v} = \frac{d}{dt}(v)$$

For the sake of simplicity, the *pure rolling condition* is assumed (Torres et al., 2014). That is, all the torque provided to the robot is used to provide motion to the robot. In other words, the robot does not skid, nor slide. A simple derivation yields:

$$\dot{\psi_L} = \frac{1}{r}(\dot{v} - b \cdot \dot{\omega}) \qquad \dot{\psi_R} = \frac{1}{r}(\dot{v} + b \cdot \dot{\omega}) \tag{7.1}$$

Odometry (also called *dead reckoning* in traditional aviation and nautical navigation (Lucas, 2000)) is the process of estimating the state $q = [x, y, \omega]^T$ from the wheel motions, essentially integrating the effect of the wheel velocities (Lynch & Park, 2017). We obtain:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} r/2 \cdot \cos\omega & r/2 \cdot \cos\omega \\ r/2 \cdot \sin\omega & r/2 \cdot \sin\omega \\ -r/(2b) & r/(2b) \end{bmatrix} \cdot \begin{bmatrix} \dot{\psi_L} \\ \dot{\psi_R} \end{bmatrix} \tag{7.2}$$

These equations were used and transformed into CBD models, but for the sake of completeness, the next section will discuss a higher fidelity model.

7.1.6.2 Offsets and Center of Mass

Commonly, O is the important point on the robot, from where all computations should happen, but when the robot is a rigid body, any point on the robot can be used as a point of reference. This is shown in Figure 7.8. Here, the point located at offset (x_T, y_T) in the robot's frame (T) has an equivalent offset (x_R, y_R) in the global reference frame (R).

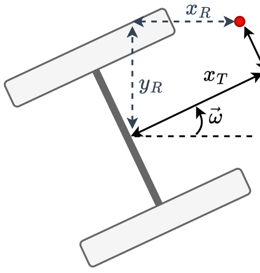


Figure 7.8: Using a different point on the robot.

Because the robot's Center of Mass (CoM) might not be located at O, we will expand Figure 7.7 to Figure 7.9. Here, we assume a single ball caster at location C. The distance between O and CoM is d, and e is the distance between CoM and C.

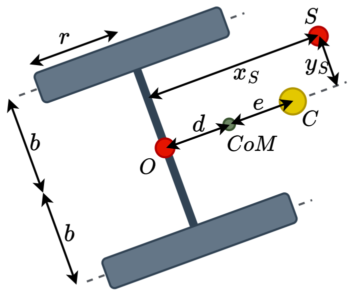


Figure 7.9: Free-body diagram of the robot, with respect to the robot's mass and a ball caster.

A sensor is located on position S. Because we use a single sensor, we can assume $y_S = 0$, but when using multiple sensors, multiple S_i points can be identified. From virtual experiments, it was observed that the closer S is to O, the harder it will be to control the LFR. Logically, this makes sense: S represents a future point that the robot will be at, so controlling values based on S allows you to impact a future state of the robot. A caveat is that we cannot set x_S too large, as it will act too soon on the robot's behaviour.

Torres et al., 2014 defines the kinematics under the pure rolling condition as:

$$\begin{cases} \dot{\psi_R} \cdot r & = \dot{x_c} c_\omega + \dot{y_c} s_\omega + b\dot{\omega} \\ \dot{\psi_L} \cdot r & = \dot{x_c} c_\omega + \dot{y_c} s_\omega - b\dot{\omega} \\ 0 & = \dot{y_c} c_\omega - \dot{x_c} s_\omega - d\dot{\omega} \end{cases} \tag{7.3}$$

Where (x_c, y_c) is the position of the CoM, $c_\omega = \cos(\omega)$, and $s_\omega = \sin(\omega)$. Therefore, we get:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} r/2 \cdot \cos\omega & r/2 \cdot \cos\omega & -d \cdot \sin\omega \\ r/2 \cdot \sin\omega & r/2 \cdot \sin\omega & d \cdot \cos\omega \\ -r/(2b) & r/(2b) & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{\psi_L} \\ \dot{\psi_R} \\ \dot{\omega} \end{bmatrix} \tag{7.4}$$

Ideally, $\dot{\omega}$ is computed first (given that the other calculations need it).

7.1.6.3 Limitations

This section has identified certain limitations within the given context, but they will be summarized below.

- It is assumed that a *frictionless* ball caster is used to prevent the robot from tipping over. In reality, this will still introduce a small amount of friction that can influence the given formulas.
- It is assumed that the robot is a rigid body, without shifting mass. On top of that, it is assumed that the CoM is aligned in the middle of the wheels (*i.e.*, the robot is symmetric).
- The pure rolling condition is assumed in all computations. In reality, there is a linear and lateral force acting on the wheels when the robot is rotating (Sloane, 2018). Torres et al., 2014 solves this by using a Coulomb friction model, but more complex approaches like Pacejka's Magic Formula (Pacejka, 1966) are commonly used in industrial contexts.
- *Overspeeding* is the physical phenomenon that appears when engines rotate at 100% power or more for a certain amount of time. The motors will start to overheat, tear or break. Typical AGVs prevent overspeeding by reducing the maximal allowed velocity for ψ_L and ψ_R.

When comparing the presented model with the actual system behaviour, an offset due to friction can (and will) be measured. A DT can here help to counteract the influence of the friction and still ensure a valid LFR behaviour.

7.1.7 Controller Modelling

Numerous control algorithms exists for controlling a LFR. Their purpose is to yield $\langle \dot{v}, \dot{\omega} \rangle$, based on the sensor input(s). As a result, they are highly dependent on the number of sensors that are being used. Within the current use-case, we will use a single sensor, measuring the colour of the underlying surface under the robot. Many alternative approaches exist. Some use multiple sensors (Erbay et al., 2024; Torres et al., 2014), others a slightly tilted camera (Sloane, 2018).

We will assume a white line on a dark background[5]: *i.e.,* if the sensor measures a *white* value, we are on the line; if it measures a *black* value, we are not[6]. We identify a threshold T to separate between "black" and "white". Notice that the line will have some *feathering* at the edges where a *grey* value is measured. Hence, we can set two thresholds: T_1 where "black" becomes "grey", and T_2 where "grey" becomes "white". The measured darkness B will be directly proportional to the distance between the sensor and the line y_e. We get $B = K_c \cdot y_e$, with K_c a constant.

We will assume the robot drives in *curvilinear space* (Brannon, 2004). This implies the path of the robot always follows a circular path with radius R and curvature κ ($= 1/R$). When $R = \pm\infty$ (*i.e.,* when $\kappa = 0$), the path is straight. When $\kappa < 0$, the path requires a clockwise motion, and when $\kappa > 0$, the path requires an anticlockwise motion. A small radius requires a sharp turn and a large radius a gentle turn. As a result estimated curvature $\tilde{\kappa}$ is a good identification of $\dot{\omega}$. This is visualized in Figure 7.10.

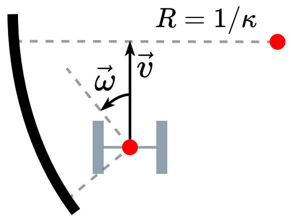


Figure 7.10: Curvilinear space, adapted from Sloane (2018).

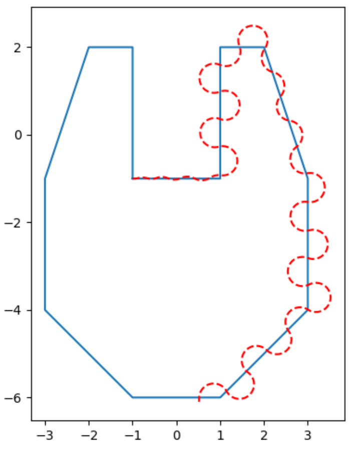
The upcoming sections will describe a set of (non-exhaustive) control algorithms that can be applied to identify $\dot{\omega}$ for the robot. Next, subsection 7.1.7.8 will focus on controlling the velocity of the robot. Finally, subsection 7.1.7.9 will discuss how the robot can recover from losing the line.

7.1.7.1 Bang-Bang Controller

Also called a *Zig-Zag* controller, assumes a closed path without intersections. The robot rotates left[7] with a constant angular velocity Ω. When the sensor identifies black (by using threshold T), the robot rotates right, with velocity Ω.

The variable K_c will be embedded in the choice of Ω. When the path is to be followed in clockwise motion, the robot will *zig-zag* over the outer edge. When anticlockwise motion is preferred, the inner edge will be used. Figure 7.11a shows an example implementation for this kind of controller. The full blue line is the path to follow, and the striped red line is the followed path. The code for this example is based on Sloane (2018).

[5]The same logic can be made for the opposite scenario.
[6]Under the assumption that the sensor is well calibrated.
[7]The same logic can be applied when swapping left and right.


(a) Bang-Bang controller.

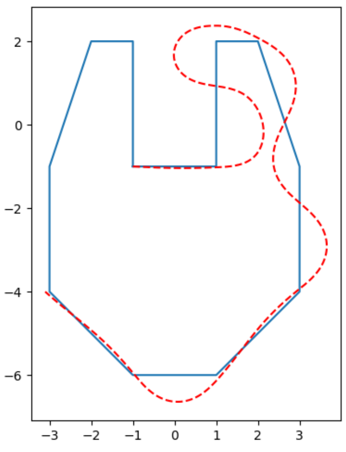

(b) Proportional controller.

Figure 7.11: Bang-Bang and Proportional controller for following a closed path.

7.1.7.2 Proportional Controller

Because B is directly proportional to y_e, it would make sense to set $\dot{\omega}$ higher when the error is larger. We identify constant K_p as a tunable variable (that accommodates for K_c), such that:

$$\dot{\omega} = K_p \cdot B \tag{7.5}$$

Figure 7.11b shows an example implementation for this kind of controller.

Of course, this controller requires K_p to be *tuned* to the most optimal value, here K_p is set to -1.7. It is possible that a better value can be found for following this trajectory.

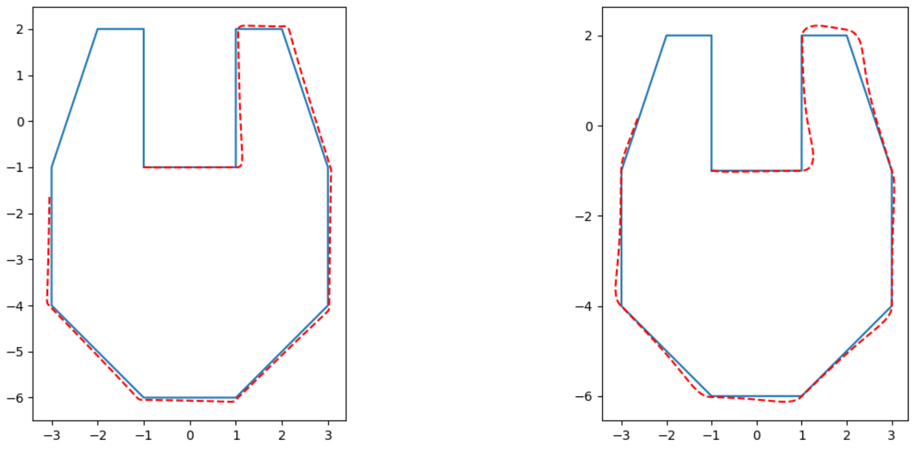
7.1.7.3 Proportional-Differential Controller

A proportional and derivative controller (also known as a PD controller) is a common choice for LFR control. Another tunable constant K_d will provide the influence of the derivative of y_e. This gives

$$\dot{\omega} = K_p \cdot B + K_d \cdot \frac{d}{dt}(B) = K_p \cdot B + K_d \cdot \dot{B} \tag{7.6}$$

In order to get $\dot{B}$, we could numerically differentiate B, which is impacted by the time delay between the previous computation and the current one. This is shown in Figure 7.12a (with $K_p = K_d = -5$). Alternatively, Sloane, 2018 states that the sine of the angle between the heading and the desired heading (ω_e) could also be used (as shown in Figure 7.12b, $K_p = K_d = -5$):

$$\dot{\omega} = K_p \cdot B + K_d \cdot \sin(\omega_e) \tag{7.7}$$



(a) Standard PD algorithm.

(b) Sine-based PD algorithm.

Figure 7.12: Proportional-Differential controller for following a closed path.

7.1.7.4 PID Controller

The Proportional-Integral-Differential (PID) controller is a PD controller that also adds an integral part, using the tunable variable K_i. PID controllers are the most common control loop mechanisms that try to minimize an error (Åström, 2002).

$$\dot{\omega} = K_p \cdot B + K_i \cdot \int B \, dt + K_d \cdot \dot{B} \tag{7.8}$$

The proportional part allows a fast approximation of the setpoint we try to achieve. The derivative part tries to anticipate the future state, but can introduce a lot of errors. The integral tries to counteract the errors from the derivative part, whilst also accounting for the past information about the system. Figure 7.13a shows an example implementation, where $K_p = -5$, $K_i = -1000$, and $K_d = -1$.

Integral Caveats. A downside of using an integrator is the potential *integral windup*. This appears when the limits of the physical system do not coincide with the desired change from the PID controller. The actuator(s) will remain at their limit(s) and the integral will continue to account for the error (Åström, 2002). It is a well-known phenomenon for control manufacturers.

In general, the integral part will have very little impact when applied to LFRs, hence it is commonly skipped in this context (*i.e.*, by setting K_i to 0).

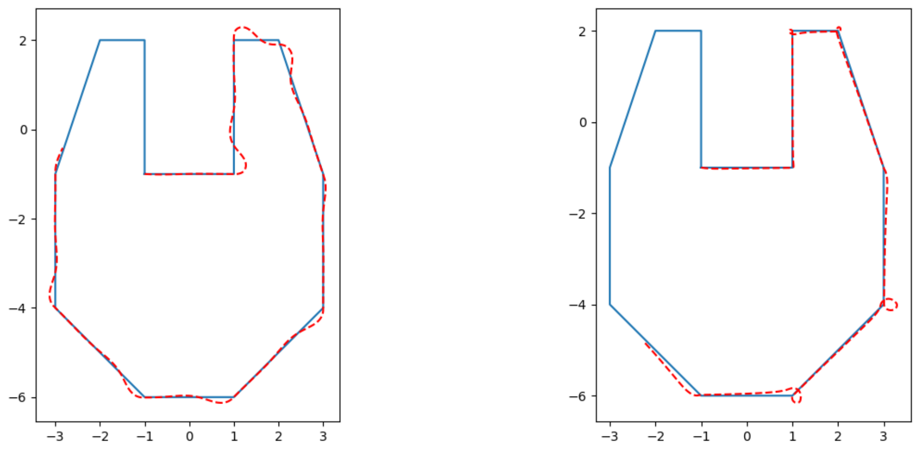

(a) Normal PID controller. (b) Curvature-aware PID controller.

Figure 7.13: Normal PID and Curvature-aware PID controller for following a closed path.

7.1.7.5 Curvature-aware PID Controller

When the curvature of the path is known, the PID controller can use this to more easily find the trajectory.

$$\dot{\omega} = K_p \cdot B + K_i \cdot \int B \, dt + K_d \cdot \dot{B} + \kappa \tag{7.9}$$

$$\dot{\omega} = K_p \cdot B + K_i \cdot \int B \, dt + K_d \cdot \sin(\omega_e) + \kappa \tag{7.10}$$

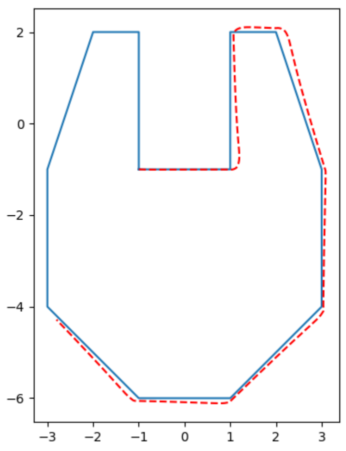
Because of the equivalence between $\dot{\omega}$ and κ, can use the previous value of $\dot{\omega}$ for κ. Any linear difference between the two will be captured by K_p, K_i and K_d. Figure 7.13b shows an example implementation using Equation 7.10, where $K_p = -5$, $K_d = -6.5$, and $K_i = 0$. Optionally, κ can also be weighted using a K_κ parameter.

7.1.7.6 PIDD2 Controller

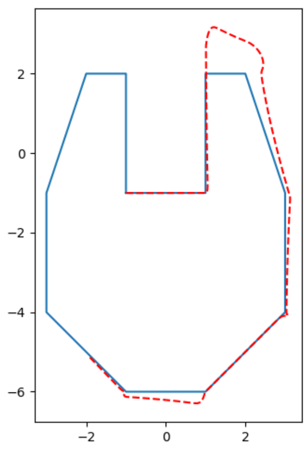
The PIDD2 controller (Kumar & Hote, 2020) is a spectial case of the PID controller that improves the steady-state approximation by including the second-level derivative of the error:

$$\dot{\omega} = K_p \cdot B + K_i \cdot \int B \, dt + K_d \cdot \dot{B} + K_{d2}\ddot{B} \tag{7.11}$$

Mathematically, this accounts for smoothing the inflection points between sequential sections of paths. Figure 7.14a shows an example implementation, where $K_p = -5$, $K_i = 0$, $K_d = -0.5$, and $K_{d2} = -0.05$.



(a) PIDD2 controller.



(b) Micaelli-Samson controller.

Figure 7.14: Special PID controllers for following a closed path.

7.1.7.7 Micaelli-Samson Controller

Micaelli and Samson, 1993 derives an alternative curvature-aware PID controller for unicycle-type and two-steering-wheels mobile robots. However, they define a unicycle as "*a vehicle with two actuated wheels on a common axle and the point M at mid-distance of these wheels*", which actually implies a differential-drive robot. This claim is validated with the kinematic equations, which corresponds to the plant model presented in Section 7.1.6.1.

The closed-loop PID formula that is obtained is

$$\dot{\omega} = vC\left(y_e C\left(g_c \sin\omega_e - K_p \cos\omega_e\right) + \sin\omega_e\left(\kappa\sin\omega_e - K_d\cos\omega_e\cdot\operatorname{sign}(vC)\right) + \kappa\right) \tag{7.12}$$

Where:

$$C = \frac{\cos\omega_e}{1-\kappa y_e}$$

and g_c is the curvature's derivative with respect to the path's curvilinear abscissa. Hence, we can use the approximation $g_c = 0$. Because of how κ is defined, we can also simplify $\operatorname{sign}(vC) = 1$. Equation 7.12 can therefore be simplified as (Sloane, 2018):

$$\dot{\omega} = vC\left(-K_p y_e C\cos\omega_e + \sin\omega_e\left(\kappa\sin\omega_e - K_d\cos\omega_e\right) + \kappa\right) \tag{7.13}$$

Figure 7.14b shows an example implementation, where $K_p = 7.5$ and $K_d = 16.4$. An advantage of this approach, is that the velocity can easily be increased, without needing to change the robot's functionality too much.

7.1.7.8 Velocity Control

The most simplistic control algorithm is by ensuring a constant velocity v (*i.e.*, $\dot{v} = 0$). This has no immediate impact in a simulated context, but when applied to a real LFR, this can result in *understeering* in turns (*i.e.*, skidding of the front wheels).

For the sake of simplicity, we will allow understeering in our system and use a small, but constant v. The interested reader is referred to Sloane, 2018 for identifying specific velocity control algorithms.

7.1.7.9 Error Recovery

When an LFR has lost the line, it can oftentimes be difficult to find the line again. Whenever the line has not been identified for τ time, we can assume the line was lost and error recovery needs to start. We identify two approaches:

- Driving in an outward spiral (*i.e.,* constantly increasing $\dot{\omega}$). This method will cover the full plane, allowing the robot to find the line again. Unfortunately, there is no knowledge of where the LFR returns to the line.
- A continuously increasing sine wave can help return to the closest point on the line, as explained on StackOverflow[8]. Unfortunately, this approach does not work well with sharp turns.

7.1.8 CBD Composition

For the plant, Equation 7.1 was used. The controller was implemented using a simple PD controller (see Section 7.1.7.3). The equations were taken and transformed into CBD models, which is a trivial transformation due to the relationship between ODEs and CBDs. The plant's CBD model can be seen in Figure 7.15, and the controller's CBD is shown in Figure 7.16.

These two CBDs can be coupled together and executed on the LFR itself. However, for the simulation, a closed loop is required. This topmost CBD is shown in Figure 7.17.

It consists of an *Environment* component (identified with the four-leaf-clover icon), which implements a lookup table between the coordinates and the position on the line. The *Control* component (identified with *Ctrl*) sets a new velocity and heading, based on the colour of the surface. Next, there is the *Plant* (identified with *DD*), and finally, the *Odometry* component (identified with *O*) is used to implement Equation 7.2.

7.1.9 Calibration

As mentioned in Section 7.1.7, the controller's parameters need to be tuned for their best execution. Furthermore, the real robot's parameters (*i.e.,* wheel radius r, half the axle length b, and the sensor threshold T) also need to be set.

This activity is commonly the most time-consuming, as it consists of running a plethora of experiments in order to identify the best parameter fit for the robot.

[8]https://stackoverflow.com/questions/40939967/implementing-pid-algorithm-in-line-following-robot/40966558#40966558, accessed on the 24th of April 2024

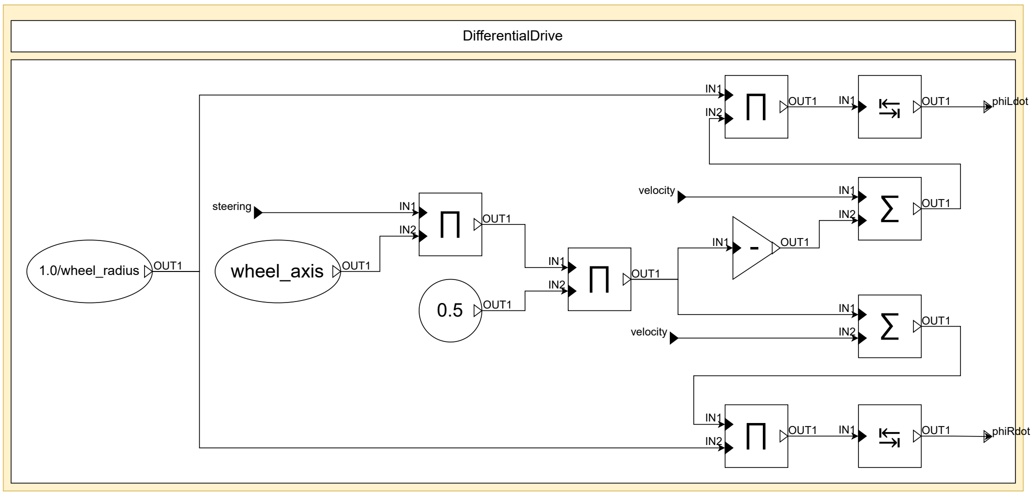


Figure 7.15: CBD plant model for the LFR use-case.

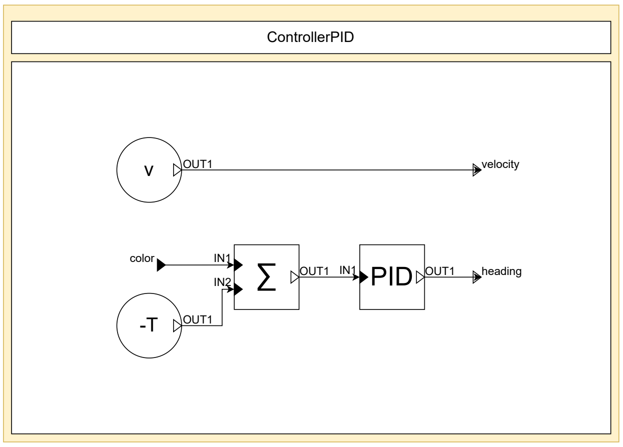


Figure 7.16: CBD controller model for the LFR use-case.

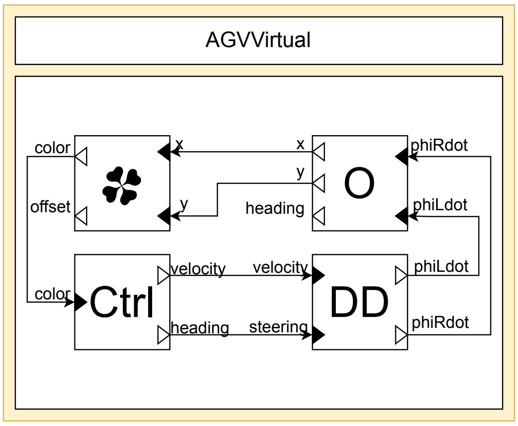


Figure 7.17: CBD model for the TO of the LFR.

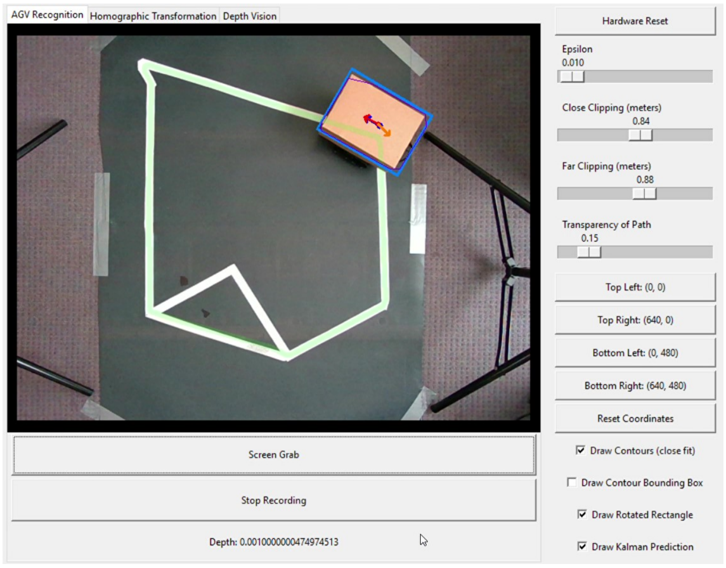
7.1.10 Deployment and Simulation

Internally, the *EV3 Brick* has an *ARM9 TI Sitara AM1x*, running a Linux operating system. Using the micro-SD slot, MicroPython can be used to run more complex software, *i.e.,* a real-time CBD simulator. The servomotors used for steering the AGV are limited to 170 rotations per minute though precision is not ensured when rotating at maximum speed. For safety, the motor speeds were limited (clamped) to 85% of their maximal velocity. The colour sensor is able to measure the amount of reflected red light, in a range from 0 to 100.

The robot communicates through TCP/IP by having an *Edimax EW-7811Un V2* WiFi dongle in the USB port. Note that this dongle is too modern to work out of the box with the *EV3 Brick*, so a new WiFi driver had to be installed onto the brick. A small tutorial on how to do this can be found on http://msdl.uantwerpen.be/people/rparedis/sources/wifi-dongle-EV3.

The camera mounted above the trajectory is an *Intel RealSense D455 Depth Camera* (https://www.intelrealsense.com/depth-camera-d455/) that is able to measure the depth in an ideal range of 0.6 to 6 meters, with an accuracy of less than 2% at 4 meters.

The camera was linked to a computer via USB, such that its information could be used. A Python[9] dashboard was created to accurately follow the position of the robot, as identified from the camera. In Figure 7.18, the dashboard is shown, where the robot is placed under a cardboard box for better image recognition. A deployment diagram is given in Figure 7.19.



Epsilon 0.010

Close Clipping (meters) 0.84

Far Clipping (meters) 0.88

Transparency of Path 0.15

Top Left: (0, 0)

Top Right: (640, 0)

Bottom Left: (0, 480)

Bottom Right: (640, 480)

[✓] Draw Contours (close fit)

[ ] Draw Contour Bounding Box

[✓] Draw Rotated Rectangle

[✓] Draw Kalman Prediction

Depth: 0.0010000000474974513

Figure 7.18: A dashboard for the LFR use-case.

[9]Using Matplotlib, TkInter, and OpenCV.

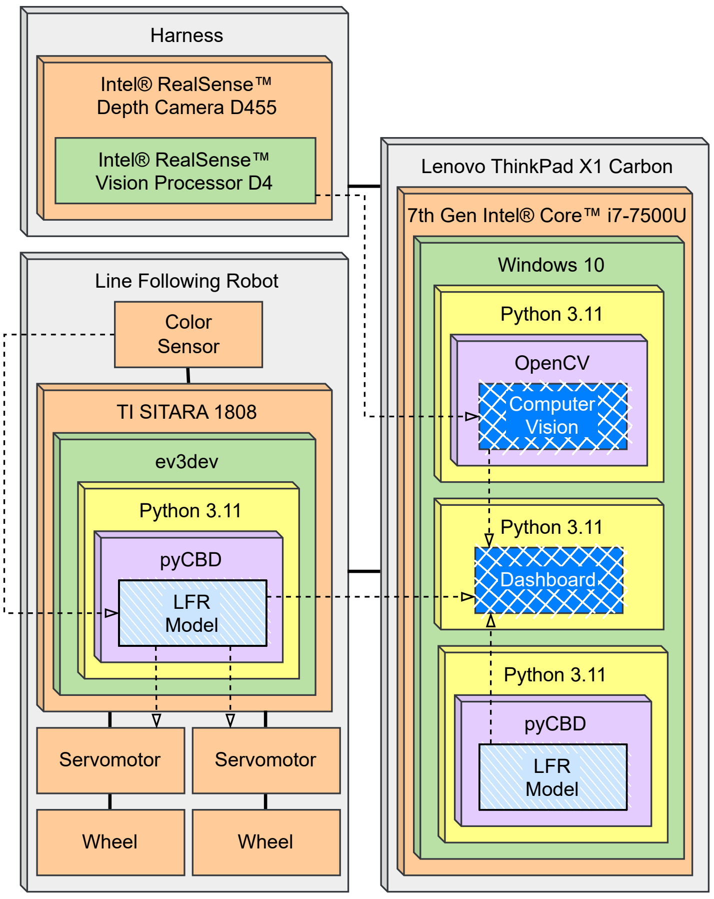


Figure 7.19: Deployment diagram for the LFR use-case.

7.1.11 System Analysis

In Figure 7.20, trajectory data for this system are presented. The blue, full line represents the line to follow (cfr. the experiment setup), the striped orange line identifies the simulation results and the dotted green line represents the AO 's identified position.

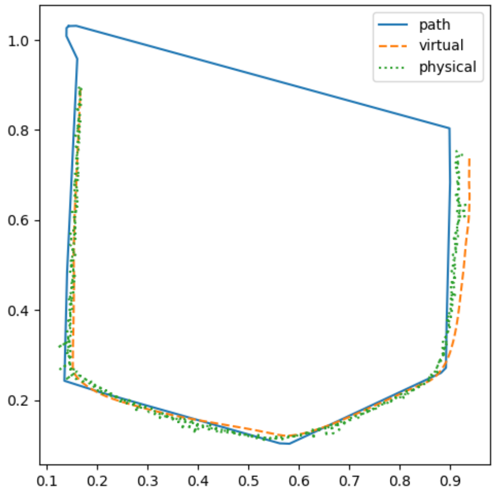


Figure 7.20: Example experiment results for the LFR.

From here, a user can identify the consistency between both the AO and the TO.

7.1.12 Robot Versions

As shown in Figure 7.1, the *System Analysis* activity can yield a termination, based on the accuracy of the resulting AGV. Alternatively, when the LFR is not good enough, a new version can be constructed, hence restarting the entire PM. When this happens, most activities will be shorter or they will reuse past results. For instance, it was identified through Unity3D that the colour sensor should not be placed in between the wheels, but further to the front, such that it can “predict” the movement.

Multiple versions of the LFR have been built, both physically, and digitally in a CAD model. The CAD model allowed a more accurate computation of the weight distribution and thus the identification of the CoM. This section will discuss the three robot versions.

7.1.12.1 Tracked Treads

The first LFR was based on LEGO Mindstorms’ EV3 “TRACK3R” building instructions that originally came with the 313131 set. However, this was only followed until building step 12. From there, the colour sensor was added instead. Figure 7.21 shows what this version of the robot looks like. The building instructions for this robot can be found on https://msdl.uantwerpen.be/cloud/public/randy-lfr-1.

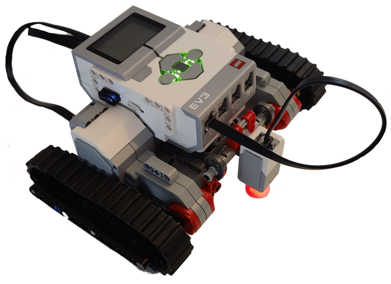
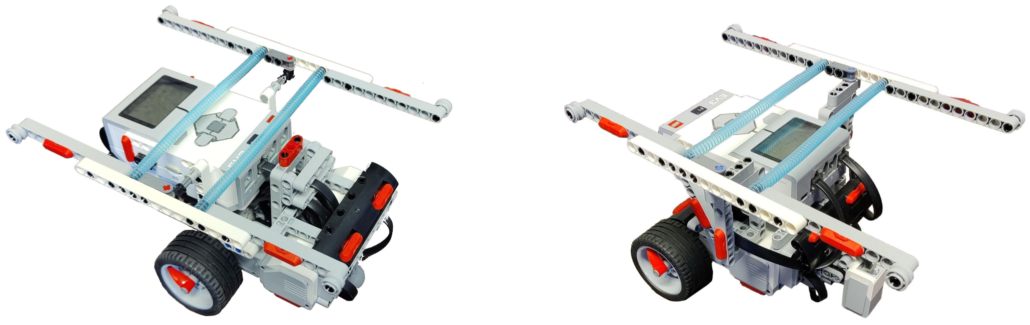


Figure 7.21: LFR with tracked threads.

Instead of focusing on weight distributions, it was assumed that tracked threads would work similarly. Unfortunately, it was quickly discovered that there were a lot of hidden difficulties introduced by using tracked threads (*e.g.*, What is the wheel radius? Where is the centre of rotation? How is slipping and skidding applied to threads? . . .).

7.1.12.2 Frictionless Ball Caster

The robot was altered to use two normal wheels, a frictionless ball caster, and a free-spinning helper wheel. The colour sensor was placed as close as possible to O, and the CoM was designed to be above the wheel axle. It is visualized in Figure 7.22a. The building instructions for this robot can be found on https://msdl.uantwerpen.be/cloud/public/randy-lfr-2.



(a) Version 2, with helper wheel.

(b) Final version.

Figure 7.22: LFR versions with a frictionless ball caster.

7.1.12.3 Final Version

The final version of this robot ensures a colour sensor at the front middle, about 7 centimeters before the wheel axle. This was identified as having a better result during a random experiment in Unity3D. An image of this final robot is shown in Figure 7.22b and the building instructions for this robot can be found on: https://msdl.uantwerpen.be/cloud/public/randy-lfr-3.

7.2 TurtleBot

The TurtleBot[10] is an open-source robotics platform, meant for education and research. It is, in essence, a Roomba vacuum cleaner with additional sensors, but without the vacuuming possibilities. Onboard sensors include (but are not limited to) 2D LiDAR, optical floor tracking sensor, wheel encoders, infrared, and cliff detection. It can be fully controlled through a ROS 2 API. It is pictured in Figure 7.23.

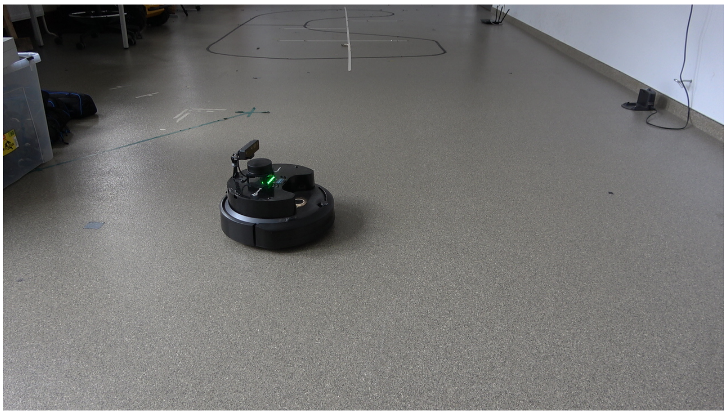


Figure 7.23: The TurtleBot driving around in the lab.

Therefore, it stands to reason that the TurtleBot is a more professional and more precise use-case, compared to the LEGO LFR. To further enhance its practical usability, official digital models of the robot and its behaviour have been built for the Gazebo simulator[11].

Whereas the LEGO LFR was a Digital Shadow, we would like to use this use-case to construct a DT (cfr. Kritzinger et al. (2018)'s definitions). In order to do so, possible combinations of TEs are highlighted, as presented in Chapter 6. Additionally, the concept of a *Historian* (from Chapter 4) will be discussed in much more detail. It is this *Historian* and the higher-level logic that allows the TurtleBot use-case to actually become a DT.

7.2.1 Requirements

When building a DT for this system, we can use the real robot as the AO and the Gazebo simulator as the TO. However, the question of *why* we are building a DT remains. Obstacle avoidance, path planning, odometry, and navigation are out-of-the-box features of the TurtleBot, and therefore the Gazebo simulation as well.

[10]https://www.turtlebot.com/
[11]https://gazebosim.org/home

This is why we have opted to create a *Tracking Simulator*. This requires us to have a *Dashboard* that shows the robot's current position, the projected trajectory and the observed world. If we click somewhere on the map, we will request the TurtleBot to drive to this new location, and the Tracking Simulator should follow.

Additionally, the simulator will also do some *State Estimation* by means of *Dead Reckoning*. This way, if the network between the AO and the TO fails, the TO can still somewhat do *Behaviour Prediction* of what it suspects the AO is doing. In term, this ensures both *Reliability* and *Stability* of the system. This gives us a Digital Shadow (using the definition of Kritzinger et al. (2018)).

In order to be able to consider it a full DT (again, using the definition of Kritzinger et al. (2018)), we need to go a bit further. Even though the TurtleBot itself has built-in path planning and navigation, we will also allow the TO to send the full desired path to the TurtleBot, thus bypassing the extra computations on the robot itself.

Furthermore, we will use this use-case to illustrate how the (blackboard) *Historian* works in the overall architecture. To do this, we will first instruct the robot to discover the full world it is driving in (this world can be stored offline, even if the robot is turned off). Next, when we instruct the robot to go to a new position, we will either:

- search the *Historian* for an existing (*i.e.*, previously computed) path between the current position and the new target; or
- use A* in the discovered world to find the best path from its current position to its new target.

Meanwhile, the robot keeps updating its world, in case obstacles are added or removed.

7.2.2 Conceptual Architecture

Based on the previous explanation, we can actually identify three main TEs (each with their own PoIs) for this use-case, as shown in Figure 7.24.

1. A Tracking Simulator (TO) will continuously be updated with the position and future path from the AO. Additionally, it will apply some dead reckoning in order to ensure stability when the connection drops. We will call this TE *Exp T* (with the "T" from "Tracking").
2. A dashboard is displaying the positions from both the AO and the TO, as well as their predicted paths. The user can click on a new position on the map to make the robot drive towards this new target. We will call this TE *Exp E* (with the "E" from "Exploration").
3. Instead of exploring the world, we may need to find a suitable navigation path in the known world. This path can be stored, such that later experiments can reuse the same path without the need for complex pathfinding. Note that this does not necessarily need to yield the shortest path. We will call this TE *Exp P* (with the "P" from "Pathfinding").

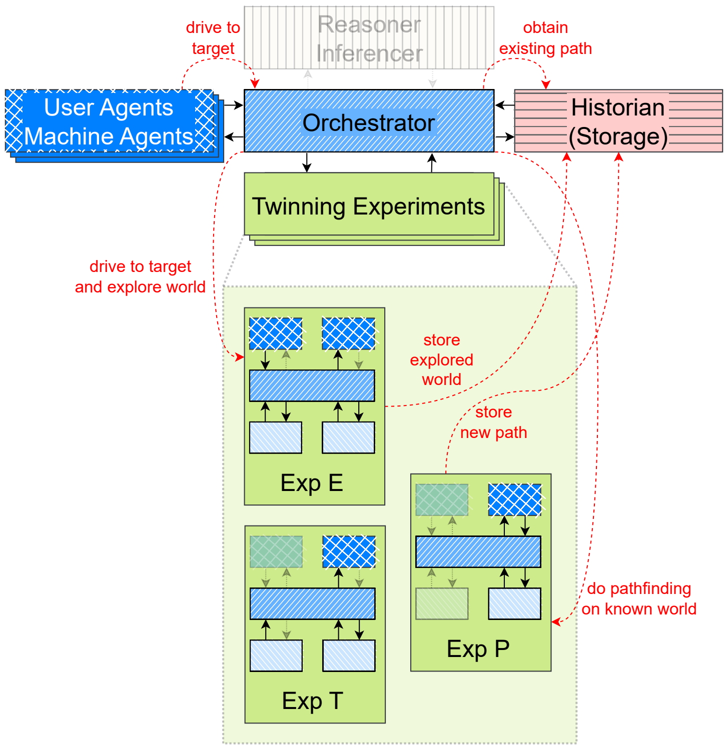


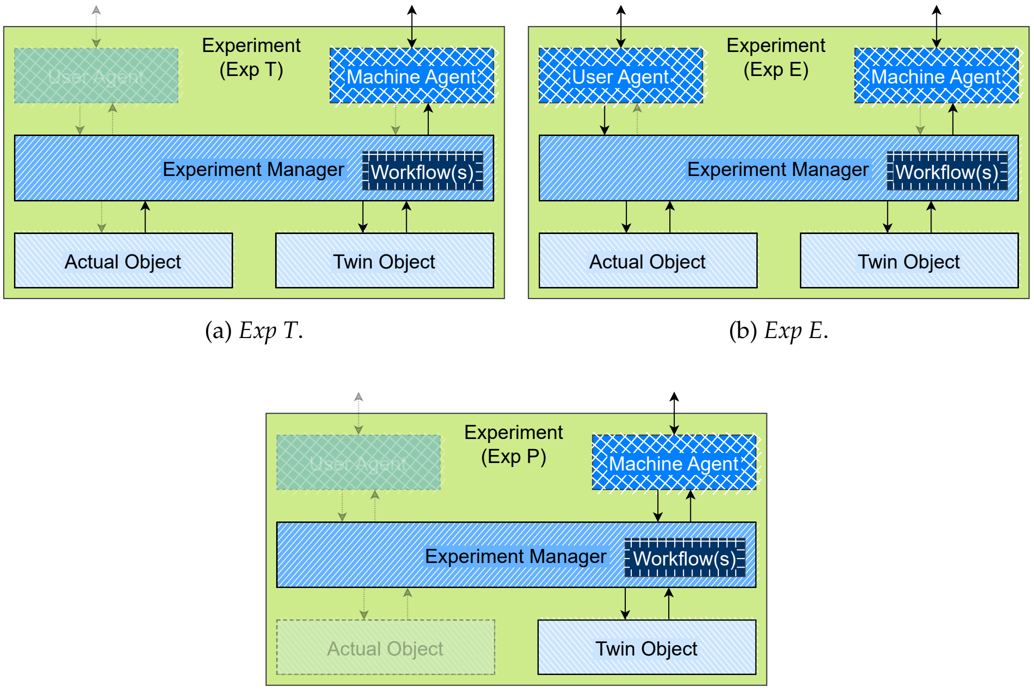
Figure 7.24: TEs for the TurtleBot use-case.

Exp T runs indefinitely, independent of the other experiments. Here, the TO continuously tracks the AO, and applies some dead reckoning for reliability/stability. Its conceptual architecture can be represented with Figure 7.25a.

A top-level User Agent sends a new target request to the *Orchestrator*, which has a choice to make. If the world is not fully explored yet, the request is forwarded to *Exp E*, such that the TurtleBot starts driving towards the new position. In the meantime, the *Historian* is also updated with the observed world information. In essence, for *Exp E*, we can identify the AO as the world in which its sensor (*i.e.*, the TurtleBot) moves around freely. The TO is a virtualization of the world map, persisted through multiple executions. Its conceptual architecture can be represented with Figure 7.25b.

Alternatively, the *Orchestrator* may decide to query the *Historian* for a path between the current location, the desired target, and its knowledge of the world. If no such a path is found, a new instance of *Exp P* is launched. The experiment will run some pathfinding and store the identified path in the *Historian*. The *Orchestrator* then ensures that the TurtleBot drives according to this path. The conceptual architecture of *Exp P* can be represented with Figure 7.25c. Note that this logic follows the one described in Cederbladh et al. (2023): a question is queried (Q), and a potential answer (A) will be returned. If no such answer exists, a new experiment (E) is launched to answer Q.

This entire logic can also be identified from the timeline view of this architecture. This can be seen Figure 7.26.



(a) *Exp T*.

(b) *Exp E*.

(c) *Exp P*.

Figure 7.25: Conceptual architecture for the TurtleBot's TEs.

7.2.3 Formalisms and Models

We can identify a few models and formalisms that we can use to build this system. Firstly, we need a Tracking Simulator, that updates its current position (x, y) and predicted path P, based on its input. Additionally, we assume the velocity v is known. When it does not receive an input, we can compute the current location with:

$$\dot{\omega} = \arctan\left(\frac{y - P_{0y}}{x - P_{0x}}\right) \tag{7.14}$$

$$\dot{x} = \cos(\omega) \cdot v \cdot \Delta t \tag{7.15}$$

$$\dot{y} = \sin(\omega) \cdot v \cdot \Delta t \tag{7.16}$$

Where ω is the heading, (P_{0x}, P_{0y}) the first waypoint in P and Δt the time since the last update. When the robot is close enough (*i.e.*, the distance D is less than threshold D_{min}) to $P_0 = (P_{0x}, P_{0y})$, we assume that this waypoint is reached, and we remove it from P. We can encode this logic into a TFSA with embedded ODEs, as shown in Figure 7.27. The dotted lines indicate different processes that happen at the same time. The first "block" identifies the internal state variables. The zero-crossing symbol ($\searrow$) identifies that this transition should happen *when* D becomes less than D_{min}. In Paredis, Denil, and Vangheluwe (2021), we discussed how such a model could be simulated through DEVS.

For *Exp E* and *Exp P*, we need a grid-based map. The TurtleBot itself uses a LiDAR sensor and runs a Simultaneous Localization and Mapping (SLAM) algorithm for identifying

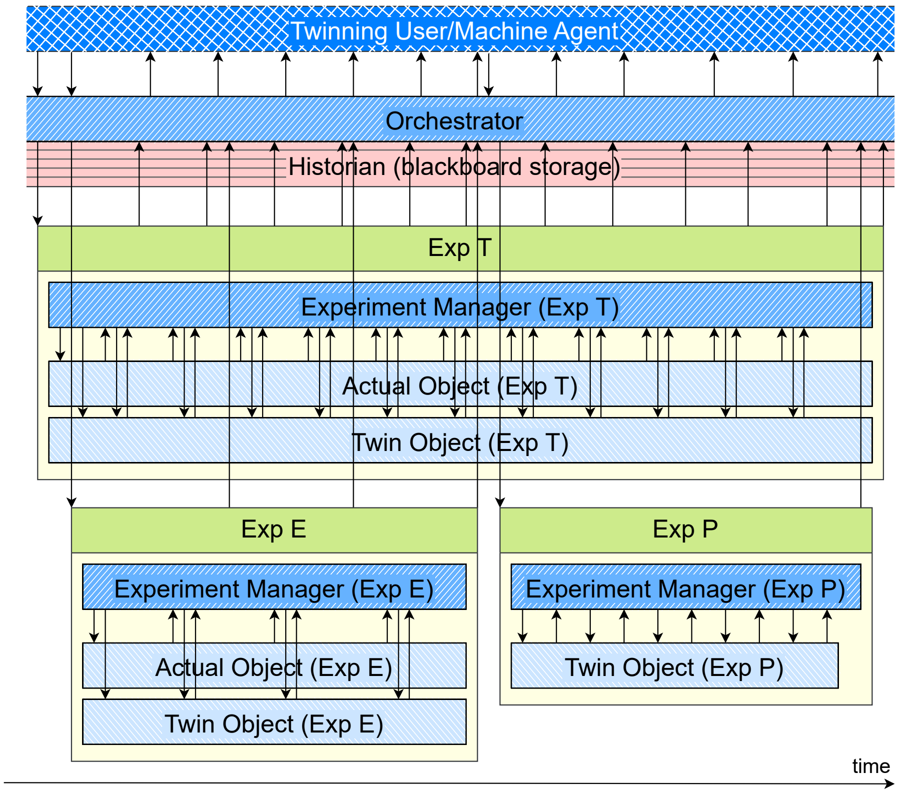

Twinning User/Machine Agent
Orchestrator
Historian (blackboard storage)
Exp T
Experiment Manager (Exp T)
Actual Object (Exp T)
Twin Object (Exp T)
Exp E
Experiment Manager (Exp E)
Actual Object (Exp E)
Twin Object (Exp E)
Exp P
Experiment Manager (Exp P)
Twin Object (Exp P)
time

Figure 7.26: Example timeline for the TurtleBot TEs.

the world it is driving in. This results in a grayscale image, where white represents accessible regions and black obstacles. Gray is used to identify uncertainty. An example map is given in Figure 7.28.

This map is given additional metadata, such as the origin (*i.e.*, the location of the bottom-left corner in world space) and the resolution (in meters/pixels). Because the robot moves around, the origin will also change and needs to be taken into account. We can use a matrix to represent this data in *Exp E* and *Exp P*. The latter can then make use of an algorithm like A* to find the best path in this map.

7.2.4 Deployment

Before we can start constructing this system, we must identify how all these components need to be merged together. Luckily, Chapter 6 describes the possibilities for merging these TEs.

Exp E and *Exp T* can share the same AO, as this represents the real TurtleBot driving in its world. The main difference is that *Exp T* looks at the movement of the robot, and *Exp E* at the construction of the world. Therefore, they have different, yet similar PoIs. We can combine the AOs as discussed in Section 6.1.1 and Figure 6.1.

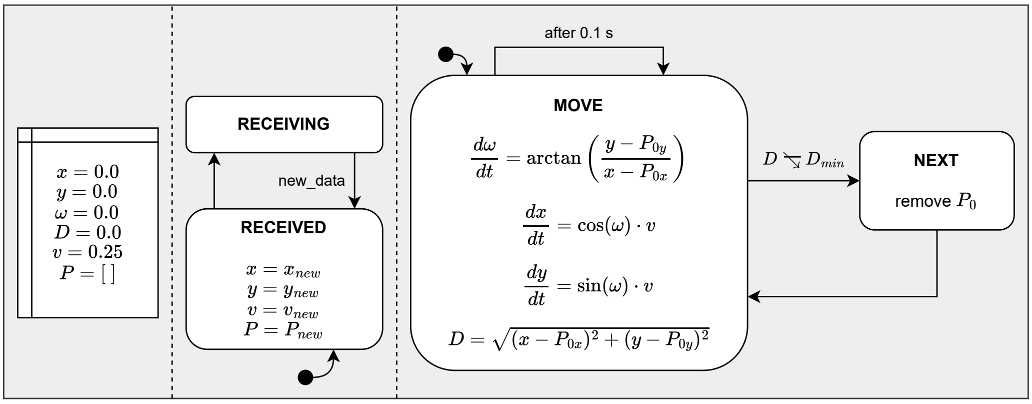

$x = 0.0$
$y = 0.0$
$\omega = 0.0$
$D = 0.0$
$v = 0.25$
$P = [\,]$
RECEIVING
new_data
RECEIVED
$x = x_{new}$
$y = y_{new}$
$v = v_{new}$
$P = P_{new}$
after 0.1 s
MOVE
$\frac{d\omega}{dt} = \arctan\left(\frac{y - P_{0y}}{x - P_{0x}}\right)$
$\frac{dx}{dt} = \cos(\omega) \cdot v$
$\frac{dy}{dt} = \sin(\omega) \cdot v$
$D = \sqrt{(x - P_{0x})^2 + (y - P_{0y})^2}$
$D \searrow D_{min}$
NEXT
remove P_0

Figure 7.27: TFSA with embedded ODEs for the TurtleBot's tracking simulator.

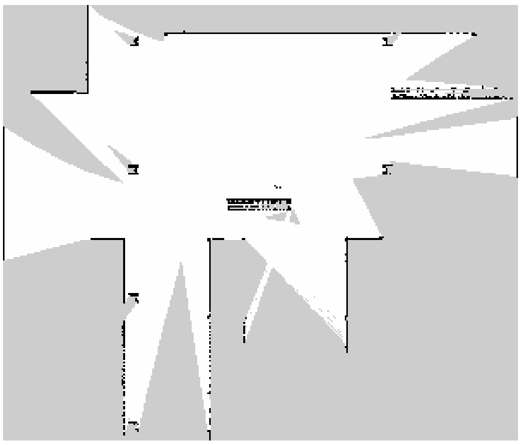


Figure 7.28: Example map for the TurtleBot.

Exp E's TO maintains a map that is also accessible by *Exp P* and the *Historian*. The PoI for *Exp E* is *exploration*, for *Exp P*, this is *pathfinding*. Both PoIs are incredibly similar, and can use the same model for their individual purposes. Hence, we can combine these TOs also as discussed in Section 6.1.1 and Figure 6.1.

Exp T's TO is a completely separate formalism that ensures tracking. Hence, we have multiple formalisms (*i.e.,* a grid and TFSA) that may be combined as discussed in Section 6.1.5 and Figure 6.7.

The main logic of the overall system lies in the *Orchestrator* and dashboard, which is either in *exploration* mode, or in *pathfinding* mode. The former closely interacts with *Exp E*, and the latter with *Exp P*.

Figure 7.29 shows a deployment diagram for this use-case. Note that this is not the only possible way of deploying this system. For instance, each process could have been deployed on a different machine. Also notice that there is no AO in this system. This is because the AO is purely a conceptual interface to the real system and therefore cannot be represented here.

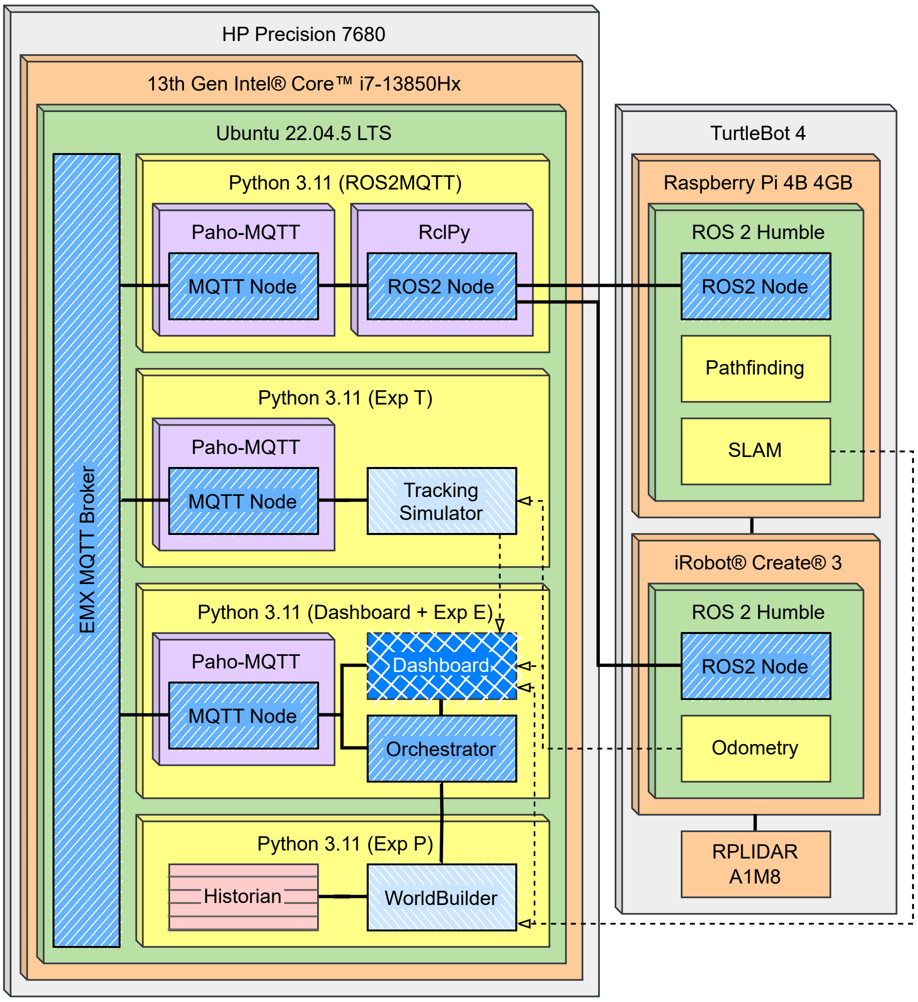


Figure 7.29: Deployment diagram for the TurtleBot.

The top-level User Agent (*i.e.*, the dashboard) was created using Python[12]. A button allows the user to switch between *exploration* and *pathfinding* mode.

The map of the world is persisted over multiple executions and can be accessed as a file through shared memory. The dashboard also visualizes this map by accessing it in the same way.

The TurtleBot communicates all its data through ROS 2. For convenience, a bridge was created between ROS 2 and MQTT, allowing all other communication to happen over MQTT. The following topics are used:

- `turtle_real/pose` The position (x, y) of the AO.
- `turtle_real/plan` The predicted plan P of the AO.

[12]With TkInter and Matplotlib

- `turtle_real/speed` The velocity v of the AO.
- `turtle_sim/pose` The position of the TO.
- `turtle_sim/plan` The predicted plan of the TO.
- `new_target` The new target position for the AO to drive towards.
- `new_plan` A full plan for the AO to drive over. Notice that the TurtleBot automatically does path planning, based on the current world knowledge. In order to execute a new plan, the TurtleBot will receive the intermediate waypoints of the path individually, forcing it to follow that exact plan.

The *Historian* keeps track of all trajectories executed when the *Orchestrator* is in *pathfinding* mode. It can be searched in a smart way:

1. Paths can be followed in both directions.
2. We can use any sub-path of all the paths we have stored.
3. The starting position does not have to match an existing waypoint exactly; *i.e.*, we can look in a (small) radius around each waypoint.

Additionally, due to the map being persisted, we can fill the *Historian* with a set of predefined paths by running A* in an offline mode.

7.3 Conclusions

This chapter has showed how to use the workflow presented in the previous chapters (*i.e.*, stage Ⓐ - Ⓓ) on two practical use-cases. First, a simple LEGO LFR (or AGV) was constructed, following a given PM. This PM closely resembles the different stages presented in this thesis. The end goal of the LEGO LFR was to construct a Digital Shadow (using Kritzinger et al. (2018)'s definition) that was able to accurately follow a line drawn on the ground. This use-case acted as a proof-of-concept for the overall Twinning idea.

Secondly, there is the more industrial TurtleBot use-case that mainly focused on how multiple TEs could be combined in order to yield a valid Twinning System, as was discussed in Chapter 6. Additionally, it focused on the *Historian* that was introduced in Chapter 4. It can practically be used for maintaining knowledge about the overall system.

Both use-cases show that Twinning Systems can easily be built using the presented workflow(s) for "toy" systems, as well as more industrial use-cases. Chapter 8 will focus on two other use-cases, in completely different domains. This shows the overall validity of the presented approach.

"Wisdom comes from experience. Experience is often a result of lack of wisdom."

– Terry Pratchett

Chapter 8 Representative Use-Case: Port of Antwerp

Starting from 2022, AnSyMo partook in the COOCK project called "*Smart Port 2025: improving and accelerating the operational efficiency of a harbour ecosystem through the application of intelligent technologies*". Here, there was a focus on nautical chain optimization of the tugboats and pilots behind the locks. This project was a collaboration between imec-UAntwerpen, TPR – University of Antwerp, AnSyMo – University of Antwerp, and the Port of Antwerp-Bruges.

The results of this project are mainly used in-house at the port, in conjunction with their APICS system. Port of Antwerp-Bruges was able to give us precise insights in the full nautical chain and the exact behaviour of the Senior Fleet Controller. This information yielded the DEVS and Modelica the assignments of the Modelling of Software-Intensive Systems (MoSIS) course in 2022-2023. The full specification of the original assignments can be found on the MoSIS course webpage:

DEVS: http://msdl.uantwerpen.be/people/hv/teaching/MoSIS/202223/assignments/DEVS

Modelica: http://msdl.uantwerpen.be/people/hv/teaching/MoSIS/202223/assignments/Modelica

This chapter will expand these assignments to use-cases in the Twinning domain. First, an expansion of the DEVS assignment will be discussed, followed by the expansion of the Modelica assignment.

For the DEVS use-case, we focus on the highly common combination of anomaly detection and fault isolation (through fault injection). We allow vessels to travel through a simulated port and identify what could have happened when the dock occupancy differs between the AO and the TO. This use-case aims to show how multiple goals (from the Feature Trees in Chapter 3) can be used in Twinning.

For the Modelica use-case, we will show that the presented conceptual architecture (see Chapter 4) can be used to identify the best implementation and deployment for a Twinning System. For simplicity, we downscale the context to a 1D movement of a ship. This allows us to construct a small proof-of-concept for *deployment space exploration* in the Twinning domain.

8.1 Port of Antwerp

In this use-case, the main focus will be on "anomaly detection" and how to solve these anomalies, similar to the Witte Dame building in Eindhoven (Verriet, 2019). Yet, to show the broadness of our approach, a larger proof-of-concept was selected as application domain.

We present a highly simplified model of the Port of Antwerp-Bruges in which vessels travel from the North Sea, over the Scheldt river, through canals and locks toward docks, and back. These locks and docks have a finite capacity and rivers and canals have certain sailing constraints.

Figure 8.1 shows a (highly) simplified map of the Scheldt through the city of Antwerp. For the sake of simplicity, we highlight the equivalence to road traffic. In the figure, circles indicate junctions where multiple trajectories merge and diverge. The three large squares (A, B, and C) are the locks and the small diamonds are the docks. The red, dotted lines identify sections of sailing routes on the river. Canals are represented with full red lines. Generally, these are two-way connections, except for the connection from the *loodskotter* source (identified with Ⓚ) and to the sea endpoint (identified with Ⓢ). Next to all river sections and all canals, their length is denoted.

Ships that arrive from Ⓚ are given a target dock to sail to, where they will stay for some time, before departing towards Ⓢ. Each ship type is given its own unique velocity distribution, based on real-world information. The docks can hold at most 50 vessels and the locks are area-constrained (*i.e.,* we will ignore the rectangle packing problem; if the remaining area of a lock is more than or equal to the surface area of a vessel, it is allowed to enter). As an additional constraint, ships cannot overtake each other on canals.

This system is clearly an abstraction of the real-world scenario, with a very small number of ships. The presented map is also an oversimplification of the real world. All distances, nodes and trajectories are an approximation of the real world scenario. This is mainly such that we can more easily explain what is going on inside. We will use this model as the TO for the port. Port of Antwerp-Bruges has provided their internal map from APICS, allowing us to use a more correct and highly detailed map. For the sake of simplicity, this detailed map will not be discussed. The full model is based on the data (*e.g.,* vessel velocity profiles, lock washing time, and distance information) we received from the Port of Antwerp-Bruges.

Unfortunately, we do not have access to the real port in order to verify our approach. Hence, to make this idea more interesting and fit within the context of Twinning, we will use another simulation of this very same system, with slightly different parameters. This way, we can analyse the impact of different properties as well. This second simulation will be considered the AO and the "source of truth" that we want the TO to conform to. In reality the real port should be used instead of a simulation.

8.1.1 Requirements

The main purpose of this use-case is trying to identify the impact of issues due to capacity constraints. If a lock or a dock becomes unavailable, we should allow a change in the

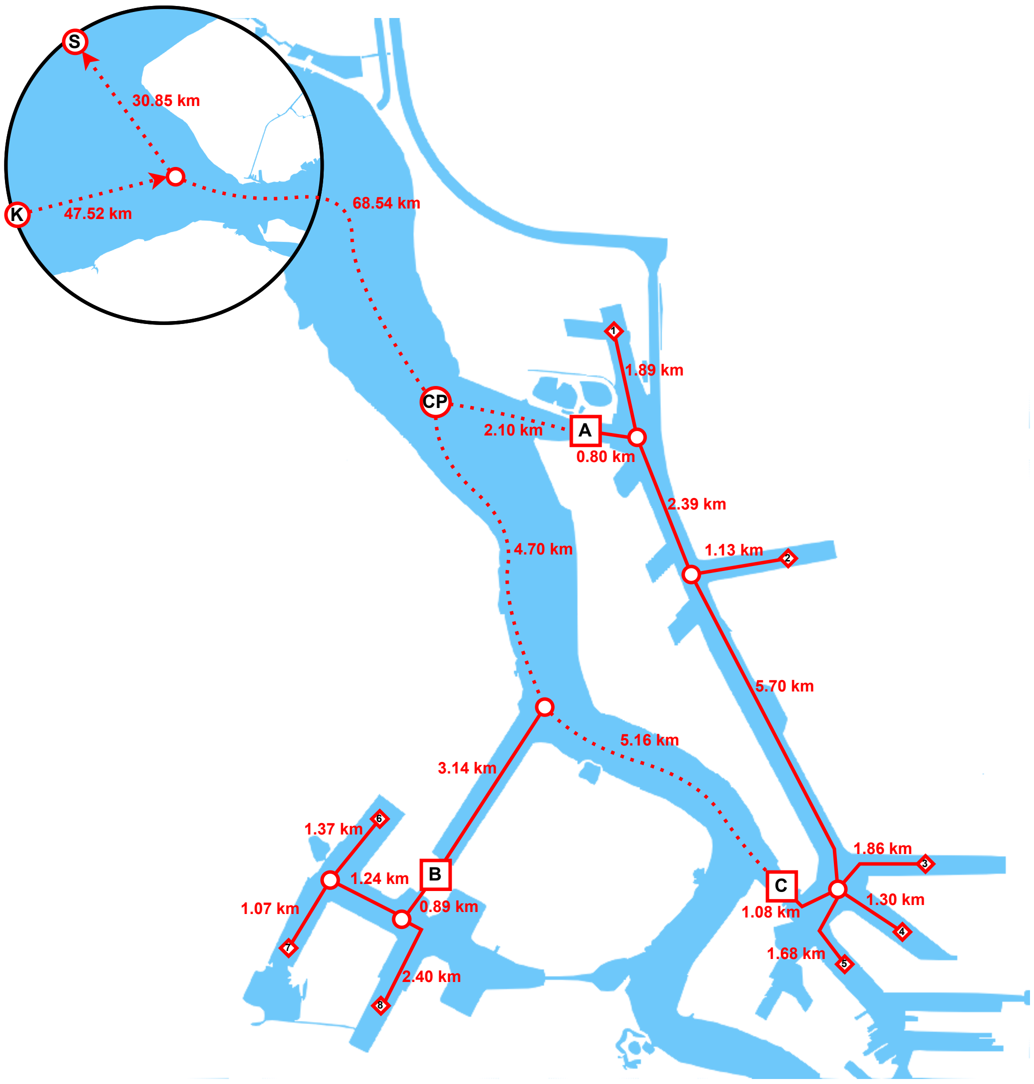


Figure 8.1: Simplified map of the Port of Antwerp.

ships' trajectory, or destination.

In order to identify the requirements, we will refer back to Chapter 3. We would like to do *Anomaly Detection* in this system, such that we can identify issues and hopefully solve them. Ideally, this requires *Fault Diagnosis*, which can be enabled through *Fault Injection*.

Additionally, we would like to visualise the current state of our model in a *Dashboard* (using *2D Animations* on a map). The full trajectory of the vessels should be made visible in this animation. The PoI for this new requirement is the trajectory progress of each vessel. In this dashboard, we also would like to show the occurrences of anomalies.

8.1.2 Conceptual Architecture

Figure 8.2 shows the architecture setup for this example case. The AO and the TO are the models of the port (note that the AO should be the real port, but we do not have access). The *Orchestrator* listens to the dock occupancy values and visualizes those in a dashboard, represented by User Agent.

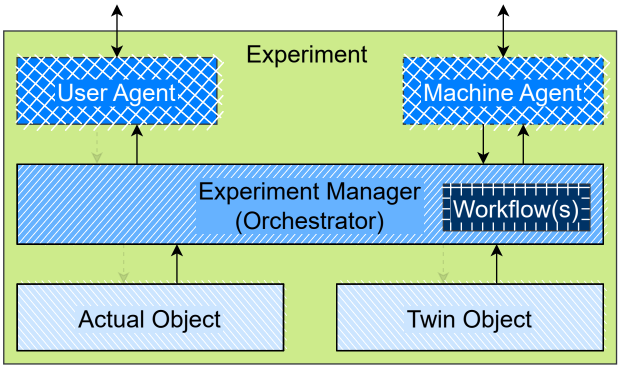


Figure 8.2: Architectural view for the example case.

8.1.3 Formalisms and Models

We can model the port example in DEVS by using PythonPDEVS (Van Tendeloo & Vangheluwe, 2015). This model will be used in both the AO and the TO. We identify the following Atomic DEVS components:

Generator Generates vessels to populate the port with.

Anchorpoint The *loodskotter* named Ⓚ on the map. When a vessel arrives here, it communicates with the Senior Fleet Controller to obtain an exact route for the vessel to sail.

Sea Collector of vessels at the end of the journey, to allow for statistics. Marked with Ⓢ on the map.

ControlTower Acts like the Senior Fleet Controller and identifies which destination in the port each vessel should go to.

WaterwaySingle Single-direction section of river where vessels can overtake each other.

CanalSingle Single-direction section of canals where vessels cannot overtake.

Confluence Acts like a junction in this network. When vessels enter, they are output on the desired output (as per their desired trajectory).

Dock Area at which a limited number of vessels may lie. Each vessel stays here for a while, before starting the journey back.

Lock Mimics the behaviour of a lock, linking the docks to the river: it allows access from the river into the lock while the gates are open if there is enough surface area left. After a while, the gates will close, the lock will start *washing* (*i.e.*, shift up or down) and open its gates on the other side, so the vessel may exit the lock and enter the canals. The same logic applies in the other direction.

For instance, the Confluence may be implemented in PythonPDEVS, as shown in Figure 8.3.

8.1.4 Deployment

Of course, to allow communication between the AO and the TO, some additional components were added on the AO's side to communicate the generation of vessels; and on the TO's side to mimic the exact behaviour. It was decided that this communication happens over MQTT.

When deploying this architecture, we get four main processes: the AO, the TO, the dashboard (User Agent) and the communication process. A deployment diagram is shown in Figure 8.4.

8.1.5 Anomaly Detection

One of the main purposes of this experiment setup is to do *Anomaly Detection*. Here, we define an anomaly as a major difference between the predicted port occupancy and the actual occupancy. When this absolute error exceeds a threshold τ for a minimal duration of θ, we assume an anomaly has happened.

This error can be measured in many ways. Because of the AO also being a DEVS simulation, we can easily use trace alignment. The Needleman-Wunch algorithm (Needleman & Wunsch, 1970) is used in bioinformatics to align protein or nucleotide sequences and compute the similarity between them. Muñoz et al. (2022) recommends using this algorithm with Fréchet-distances to measure the similarity between traces. This approach is highly similar to the Dynamic Time Warping (DTW) algorithm for measuring the similarity between temporal sequences. Gong et al. (2022) uses this to measure the similarity between a person and a robotic arm. A related algorithm, the Smith-Waterman algorithm

```python
class Confluence(AtomicDEVS):
    def __init__(self, name, segments, quays_out):
        super(Confluence, self).__init__(name)
        self.ins = [self.addInPort("in%i" % x) for x in range(segments)]
        self.outs = [self.addOutPort("out%i" % x) for x in range(segments)]

        self.enc = self.addOutPort("enc")

        # reverse the mapping for better lookup
        self.quay_mapping = {}
        for output_idx, quays in quays_out.items():
            for quay_id in quays:
                self.quay_mapping[quay_id] = output_idx

        self.state = {
            "vessels": []
        }

    def timeAdvance(self):
        if len(self.state["vessels"]) > 0:
            return 0.
        return INFINITY

    def extTransition(self, inputs):
        for inp in self.ins:
            if inp in inputs:
                vessel = inputs[inp]
                vessel.source = vessel.trajectory.pop(0)
                self.state["vessels"].append(vessel)
        return self.state

    def outputFnc(self):
        if len(self.state["vessels"]) > 0:
            vessel = self.state["vessels"][0]
            output = self.quay_mapping[vessel.trajectory[0]]
            return {
                self.outs[output]: vessel
            }
        return {}

    def intTransition(self):
        self.state["vessels"].pop(0)
        return self.state
```

Figure 8.3: PythonPDEVS example implementation for a Confluence.

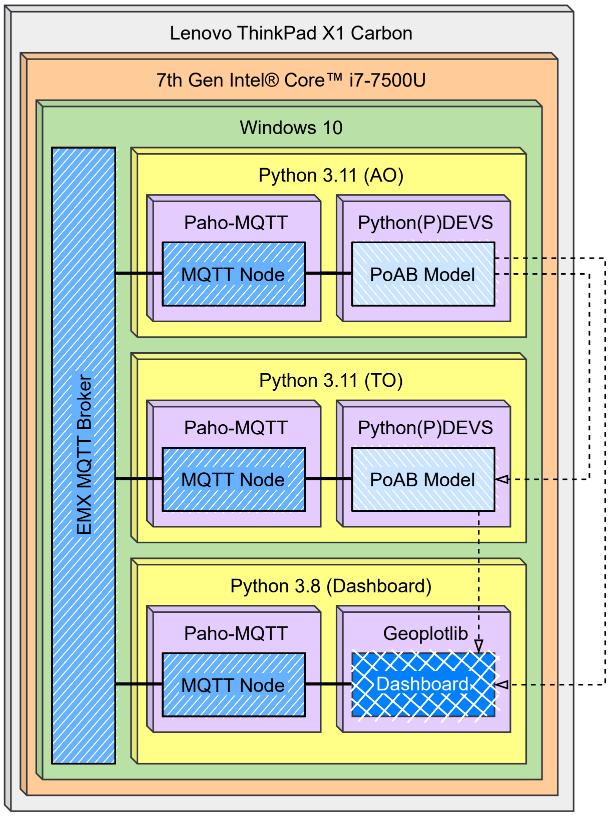


Figure 8.4: Deployment for the Port of Antwerp-Bruges use-case.

(T. F. Smith, Waterman, et al., 1981) could be used to find a local alignment (Zaki et al., 2009). The Levenshtein distance has also been recommended to compare traces (Leroy et al., 2018).

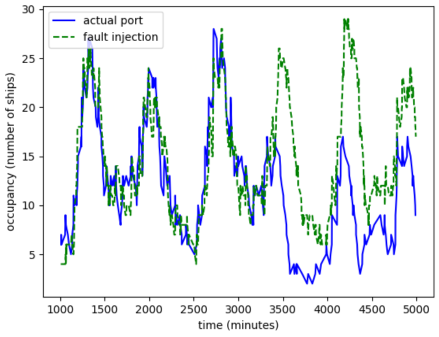
Alternatively, Worden et al. (2020) proposes to measure the similarity between two groups of simulations using the Kullback-Liebler Divergence. A similar approach recommends the Jensen-Shannon Distance (Bojarczuk et al., 2021).

However, we are not looking for a complicated alignment, as we can tweak the traces through DEVS. The main difference will be due to the random number generators – or an anomaly, hence a simple (absolute) point-wise difference can be used as error.

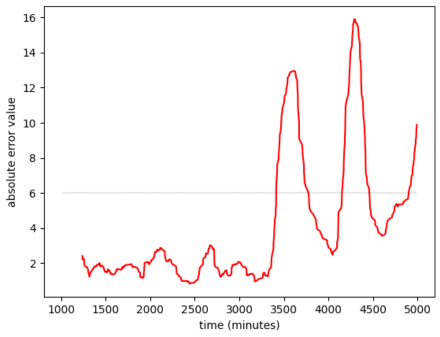
In this system, multiple anomalies may occur: ships might stop entering the port, communication delays could yield inconsistencies, locks or berths may shut down, *etc.*

Suppose the Port of Antwerp-Bruges has a dashboard that keeps track of the current number of vessels that are docked at any time. At the same time, a virtual simulation keeps track of the expected number of vessels docked. This is visualized in Figure 8.5a. The full blue line shows the situation in the AO, whereas the striped green line shows the TO's prediction.

The curves on this plot mostly align, with some small deviations due to unknown ran-



(a) Docking occupancy anomaly trace plot.



(b) Docking anomaly absolute error curve, smoothened by a Savitzky-Golay filter.

Figure 8.5: Docking occupancy anomaly trace and absolute error.

domness. Then, suddenly, somewhere after 3000 minutes, the real port's value suddenly drops below the simulated port's prediction and stays there for quite a while. The dashboard notifies us that an anomaly has occurred. We can verify this event in the absolute error plot, as shown in Figure 8.5b.

The red line is the error curve between the AO and the TO, after being smoothened by a Savitzky-Golay filter. The black line indicates the drop-off point. If the error is above this line for too long, an anomaly is raised. This drop-off is chosen (and calibrated) such that everything above this line is accepted as valid execution.

The anomaly detection algorithm is given in Algorithm 8.9. Notice that this is a naive point-wise comparison that suffices for our use-case, but other algorithms might be better suited, as mentioned earlier.

In order to identify which anomaly has occurred, a set of new, "mutated" TOs can be spawned. These mutations use the same simulation setup, but apply *Fault Injection*. These faults represent some potential identified reasons why the anomaly might have happened. We have identified five distinct cases where an anomaly might occur. We will verify the injected faults both visually and mathematically, using DTW. Note that, instead of DTW, an altered Needleman-Wunch algorithm could be used as well (Muñoz et al., 2022; 2024).

8.1.5.1 Limited Ships

One potential reason for the plot mismatch is a lack of arriving ships. The impact of this situation is shown in Figure 8.6. The beginning of the anomaly lines up with the results from this injected fault, with a similar drop-off rate. Yet, as soon as all vessels have arrived (and spent some time in the port), the simulated port empties, which is visualised with the striped green line reaching 0. This is clearly not the anomaly we observe. When we compare both traces using DTW, a distance of 461 was observed.

Algorithm 8.9 Simple anomaly detection algorithm.

```
$D \leftarrow$ Drop-off value for the error.
$M \leftarrow$ Maximal allowed time above $D$.
$prev \leftarrow -1$                              ▷ Stores when the error exceeds $D$.
while system is running do
    $occ_A \leftarrow$ Occupancy value of the AO.
    $occ_T \leftarrow$ Occupancy value of the TO.
    $T \leftarrow$ Current execution time.
    $error \leftarrow |occ_A - occ_T|$
    if $prev > 0$ and $T - prev > M$ then
        Raise an anomaly.
    end if
    if $error > D$ then
        if $prev < 0$ then
            $prev \leftarrow T$
        end if
    else
        $prev \leftarrow -1$
    end if
end while
```

8.1.5.2 Broken Berendrecht-Zandvliet Lock

A second attempt was made by disabling the usage of one of the locks. For this anomaly, the Berendrecht-Zandvliet Lock was chosen. As can be seen in Figure 8.7, this fault is closer to the actual behaviour in the real port. The observed DTW distance comes to 317, which is significantly lower than the previous scenario, thus yielding a higher possibility that this is the error.

8.1.5.3 Broken Kieldrecht Lock

Given the significant correlation between the traces in the previous attempt, we will analyse the impact of another lock. We will disable the usage of the Kieldrecht Lock, thus cutting off the left bank of the river (within this example). As can be seen in Figure 8.8, both the actual port's situation and the injected simulation are incredibly close, with some deviation that can easily be caused by unaccounted randomness. The DTW distance here is 288, again lower than the previous scenario. We have indeed verified that there is an issue with the locks.

8.1.5.4 Broken Boudewijn-Van Cauwelaert Lock

When we test the third and final lock, the Boudewijn-Van Cauwelaert Lock, we identify a DTW distance of 252. As can be seen in Figure 8.9, both the actual port's situation and the injected simulation are even closer than in the previous scenario.

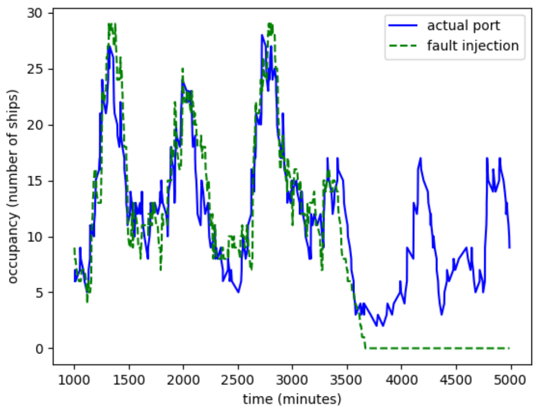


Figure 8.6: Limiting the arrival of ships fault injection.

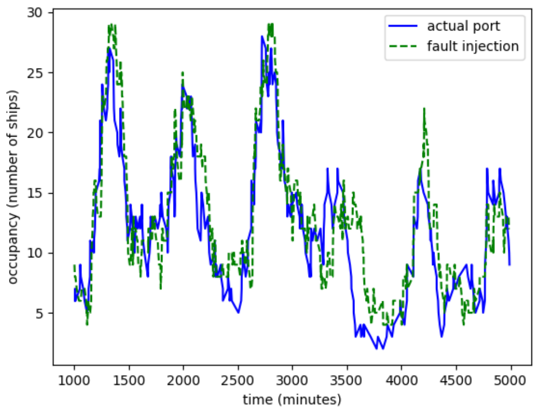


Figure 8.7: Broken Berendrecht-Zandvliet Lock fault injection.

8.1.5.5 Two Broken Locks

Whilst we now can assume the issue is in the locks, we do not know if the issue caused by a single lock. To check this, we will disable both the Berendrecht-Zandvliet Lock, as well as the Kieldrecht Lock. As can be seen in Figure 8.10, around the time of the anomaly, there is a clear mismatch in both plots. Furthermore, the DTW distance has increased to 399. We now know there is likely not an issue with multiple locks.

8.1.5.6 Verification

From the injected faults, we have deduced a hypothesis of where the issue may lie: a lock has broken down. Most likely, this will be the Bouwdewijn-Van Cauwelaert Lock,

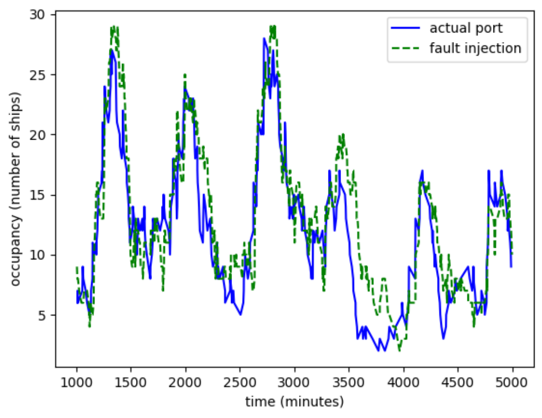


Figure 8.8: Broken Kieldrecht Lock fault injection.

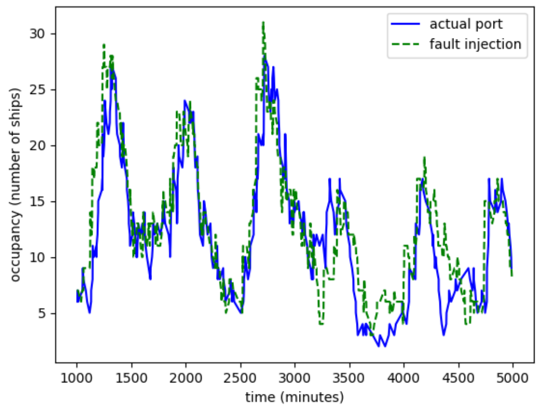


Figure 8.9: Broken Boudewijn-Van Cauwelaert Lock fault injection.

but the Kieldrecht Lock is also a possibility. We would expect that the occupancies of the docks also decreases, as their traffic is related to the locks. Hence, we can narrow down the problem by verifying the occupancy in the docks. Figure 8.11 shows docks 6, 7, and 8 empty after the anomaly has occurred. We can conclude that it is very likely that the Kieldrecht Lock did indeed stop working.

Unfortunately, this issue is not easily fixed, but it is important for the port to be made aware as fast as possible, such that a broken lock can be fixed. In reality, there is another lock on the left bank (the Kallo Lock), making sure that the port can continue to function in the meantime. The simplified example we've used ignores this lock.

Of course, these are way to few experiments to know for certain that this is the exact issue that occurred. In that sense, we cannot truly "verify" this solution. Additionally, it is important to denote that these statistics are not meant to improve the behaviour of the port. It is rather a proof-of-concept that you can use this information to analyse the

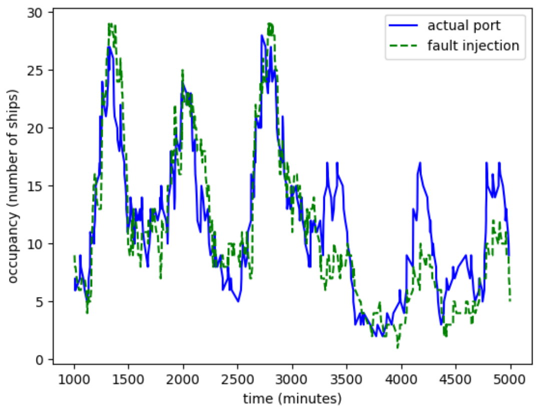


Figure 8.10: Broken Berendrecht-Zandvliet Lock and broken Kieldrecht Lock fault injection.

yielded anomalies.

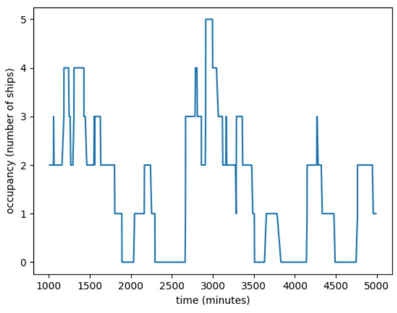
8.2 1D Movement of a Vessel

When building a Twinning System, the selection of the right formalisms (in stage Ⓒ) and the exact technologies and tools (in stage Ⓓ) might be quite ad-hoc. If stage Ⓑ yielded a lot of viable alternatives, it might be unclear which alternative architecture is the most appropriate. Similarly, the choice of a certain technology might have a large impact on the behaviour of the overall Twinning architecture's performance. This will only be known upon deployment. Maybe at one point, a user would also like to change a modelling formalism, or a communication protocol to check its influence on the whole system. Instead of needing to re-deploy and invest in this potential negative change, it would be useful for the user to be able to verify this beforehand. This is where we wanted to answer *Research Question 4*: *"How to quantitatively support deployment choices?"*.

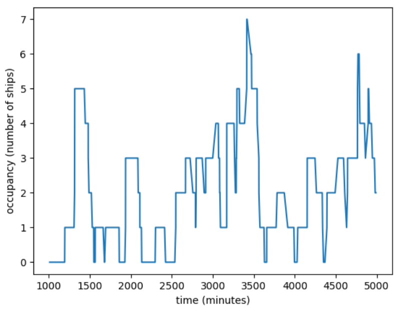
Architecture space exploration and *deployment space exploration* are needed, where multiple solutions for all twin variants can be compared objectively. Doing this using fully realized implementations can easily become prohibitively expensive.

MBSE is meant to support designing (complex) CPS. Following the MBSE paradigm, it stands to reason that we construct a simulatable model of the system's architecture and its deployment. We can easily modify this architecture/deployment model to verify the influence of changes on overall system performance.

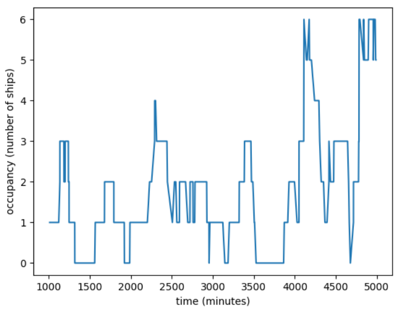
In order to illustrate the proposed modelling and simulation of Twinning architectures, a small example of a ship and its 1D motion is used. This use-case is meant as a proof-of-concept and does not provide an exhaustive identification of all possible variants.



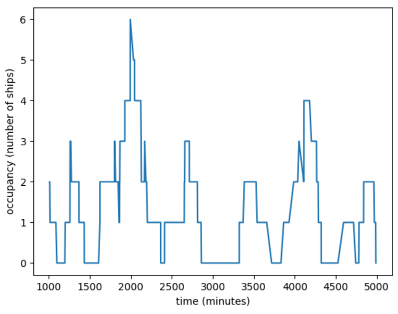
(a) Occupancy of dock 1.



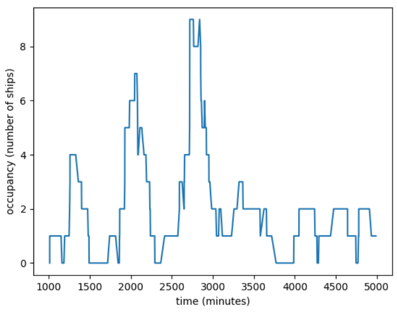
(b) Occupancy of dock 2.



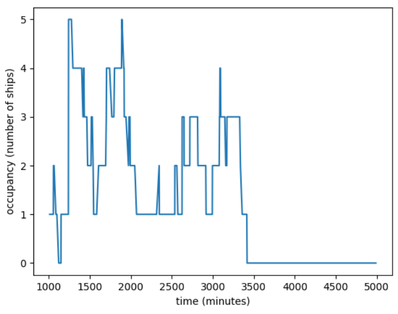
(c) Occupancy of dock 3.



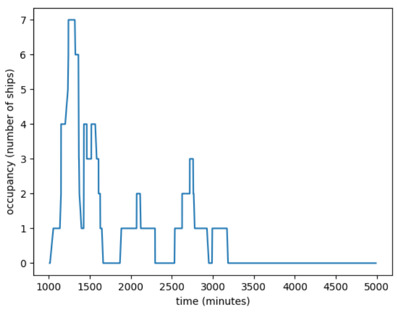
(d) Occupancy of dock 4.



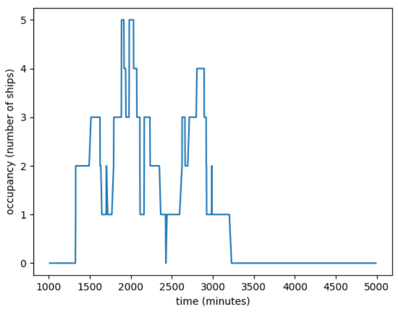
(e) Occupancy of dock 5.



(f) Occupancy of dock 6.



(g) Occupancy of dock 7.



(h) Occupancy of dock 8.

Figure 8.11: Occupancy check of broken Lock fault injection.

8.2.1 Requirements

We consider a ship that only sails in a single direction (*i.e.,* turning is not part of this example nor are pitch and yaw taken into account). We assume that the ship's motion is not influenced by external factors such as currents, wind, tides, *etc.* We know the length L (21.54 m) of the ship, the estimated dry mass m (32,000 kg), and the estimated submerged surface area S (261 m^2) of the vessel (based on earlier parameter estimation experiments). Experiments were carried out in water with a temperature of 15°C (which makes the density of the water ρ approximately 1025 kg/m^3).

Given that we do not have access to a real-world AO, we used a simulation for this, in the form of an FMU. The FMU is a black-box that produces a measured velocity v_{AO} at each point in time. As an external input, the ship receives a time-varying desired target velocity v_t. In this instance, the FMU also produces this v_t at each point in time. This desired target velocity will also be input to the TO.

Instead of merely constructing a simulation of this system, we would like to construct a Twinning System, however, we have very little ideas about which technologies would be best applicable.

From our Feature Trees (see Chapter 3), we select the requirements that are necessary for this use-case. We would like to *monitor* the current (and past) states, supporting *data allocation*. This way we can *visualize* a (historical) plot of the system state, which is *animated* during a *real-time*, *live execution*. When looking at this plot, a human may assess the *consistency* and *reliability* of the constructed twin. Furthermore we can observe whether (1) the velocity of the ship stays within legal bounds (*legal safety*), (2) the ship's engine can produce enough torque to reach the desired velocities (*physical laws*), and (3) passengers do not fall due to excessive acceleration or deceleration (*human safety*). Finally, we may want to *reproduce* such an experiment, also for different ships.

8.2.2 Architecture

The above helped us identify which components are required (and which can be omitted) in the conceptual reference architecture presented earlier in Chapter 4, Figure 4.1.

The black-box model is the AO and our ideal model is simulated in the TO. The AO cannot be controlled, but does provide information (*i.e.,* the v_t) to the TO. A co-simulation orchestrator, interleaving the time-stepping of the AO and TO, is required for correct results. A Machine Agent is not needed, but a simple visualization dashboard is required in the User Agent. These considerations result in the architectural variant shown in Figure 8.12.

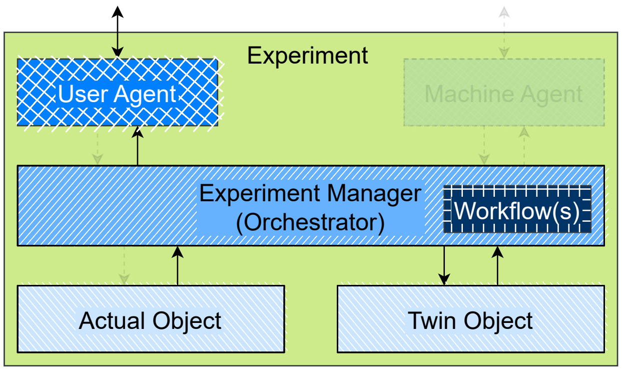


Figure 8.12: Conceptual architecture variant for the ship use-case.

8.2.3 Formalisms and Models

Moving to stage Ⓒ, we can start concretizing this system. In the TO, we use a simple model for physical plant and controller. The plant model consists of the following ODE:

$$\begin{cases} F_R &= \frac{1}{2} \cdot \rho \cdot v_{TO}^2 \cdot S \cdot C_f \\ C_f &= \frac{0.075}{\left(\log_{10}(Re) - 2\right)^2} \\ Re &= v_{TO} \cdot L/k \\ \frac{dv_{TO}}{dt} &= \frac{F_T - F_R}{m} \end{cases} \tag{8.1}$$

F_R is the resistive force the ship experiences when sailing at velocity v. This results in friction value C_f. As introduced earlier, ρ, S, and L are the density of the water, the submerged surface area, and the length of the hull. Re is the Reynolds number for the ship in water, which indicates the nature (laminar or turbulent) of the fluid flow. k is the dynamic viscosity of the water ($1.188 \cdot 10^{-6}\ kg/(m \cdot s)$). F_T is the traction force, as determined by a PID controller which minimizes the difference between the actual velocity v_{TO} and the desired (or target) velocity v_t. The target velocity profile is an external input to the system which in this case is produced by the AO. Our PID controller mimics the controller implemented in the real-world SuS. The PID controller's parameters were estimated during previous experiments. We have modelled the controller equations combined with the plant equations in Modelica. OpenModelica was used to generate an FMU simulation unit.

8.2.4 Deployment

In an ideal scenario, the plot shown in Figure 8.13 should be obtained. It shows the velocity of the real ship v_{AO} (full orange line) and that of its twin v_{TO} (dashed green line) as they try to reach the varying target velocity v_t (dotted blue line), as they are displayed in the dashboard. From the plot, it is clear that, as was to be expected, v_{AO} contains sensor noise. The behaviour seems normal however. If some event were to prevent the

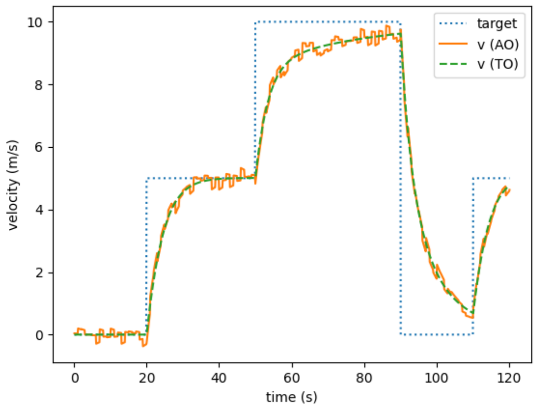


Figure 8.13: Velocity of the real ship (AO) versus that of its twin (TO) as they try to reach the varying target velocity v_t.

velocity of the ship to change as expected, such as being stuck on a sand bank, or the engine failing, this anomaly can easily be identified in the dashboard.

8.2.5 DEVS Model

Note that we have not fully completed stage Ⓓ as we did not discuss actual deployment. Suppose that we don't know which communication protocol to use in order to get the results of Figure 8.13. Instead of testing the system with a large number of technologies, we propose to use MBSE techniques to explore the deployment space. Different deployment architecture will be simulated. As we are simulating Twinning architectures, these will themselves include simulations (of the ship in this case).

According to MPM principles, we we need to select a *most appropriate* formalism to model the architecture. Vangheluwe (2000) shows that the DEVS formalism can be used as a modular assembly language to which a plethora of other existing languages can be mapped, preserving behaviour. Given the heterogeneity of Twinning architectures, and as open modelling and simulation tooling exist, DEVS seems the most appropriate formalism to date.

Hence, DEVS will be used to simulate the exact architecture, using specific technologies. For simplicity, a *FMURunner* atomic component is constructed that can simulate a given FMU for a single time-step (upon receiving an input). The orchestrator can then easily be used for co-simulation between the AO and the TO. The PythonPDEVS code for the orchestrator is shown in Figure 8.14.

There exist DEVS performance models for specific communication protocols and technologies (Burger et al., 2019; Maruyama et al., 2016). However, in many situations this may not be the case. For those instances, we do have access to the exact technologies themselves. We manipulate the DEVS components to make use of the actual technologies, instead of them being an abstraction thereof. This ensures the usage of the actual system

```python
class Orchestrator(AtomicDEVS):
  def __init__(self, name, stepsize=0.1, stoptime=1.0):
    super().__init__(name)
    self.stepsize, self.stoptime = stepsize, stoptime
    self.modules = ["AO", "TO"]
    self.state = {
      "data": { m:[] for m in self.modules },
      "current": 0, "must_output": True,
      "time": { m: 0.0 for m in self.modules }
    }
    self.inputs = { m: self.addInPort("%s-data" % m) for m in self.modules }
    self.outputs = { m: self.addOutPort("%s-action" % m) for m in self.modules }

  def get_cur(self):
    return self.modules[self.state["current"]]

  def timeAdvance(self):
    if self.state["must_output"]:
      return 0.0
    return INFINITY

  def extTransition(self, inputs):
    if self.inputs[self.get_cur()] in inputs:
      cur = self.get_cur()
      data = inputs[self.inputs[cur]][0]
      if round(data[0] - self.state["time"][cur], 6) >= self.stepsize:
        self.state["current"] = (self.state["current"] + 1) % len(self.modules)
      self.state["must_output"] = True
      self.state["time"][cur] = data[0]
      self.state["data"][cur].append(data)
    return self.state

  def outputFnc(self):
    if self.get_cur() == self.modules[0]:
      return {
        self.outputs[self.modules[0]]: ["doStep"]
      }
    elif self.get_cur() == self.modules[1]:
      last, prev = None, None
      for time, value in self.state["data"][self.modules[0]]:
        if time == self.state["time"][self.get_cur()]:
          last = time, value
          break
        elif time > self.state["time"][self.get_cur()]:
          last = time, value
          if prev is not None:
            last = lerp(prev, last, time)
          break
        prev = time, value

      return {
        self.outputs[self.modules[1]]: [last]
      }
    return {}

  def intTransition(self):
    self.state["must_output"] = False
    return self.state
```

Figure 8.14: Orchestrator in PythonPDEVS.

components, including their hidden complexities and unknown influences to the environment. This makes later conversion of the simulation into a realization easier. A similar approach was used in Denil et al. (2017) to explore design and deployment alternatives for a car power window. Syriani and Vangheluwe (2013) applied this approach to evaluate implementations of graph transformation schedules. This is a common approach for calibrating simple black-box systems.

We know that a communication protocol introduces network delays, but does not alter the data that is transferred. Running multiple experiments with the actual system in place will allow us to construct a distribution for the expected time delays, which can subsequently be used in a pure simulation context.

8.2.6 Analysis of Alternatives

Before actually building a working DEVS model of the presented architecture, it is imperative that the conceptual architecture is concretised. But, to do so, we would need to know some additional information about the components. For instance, which communication protocol, such as polling or publish-subscribe, best fits our needs. Or, alternatively, when a communication protocol is chosen, which technology is most appropriate? When experimenting with different technologies, we use the traces in Figure 8.13 to verify correct behaviour.

8.2.6.1 Different Publish-Subscribe Implementations

As a first attempt in finding the most ideal implementation for this system, we will try a publish-subscribe approach. However, given that there exist multiple possibilities to realise this, we need to compare multiple publish-subscribe alternatives. Note that this is by no means a general analysis of these technologies, but rather a use-case-specific comparison on which one works the best *in this case*.

We compare the latency of ROS 2[1], MQTT, and OPC UA's publish-subscribe system (running on top of TCP/IP). All these technologies will use Figure 8.15 as a concrete system architecture.

Figure 8.16 shows a bar plot of the latency obtained after running the publish-subscribe experiments on the same system, using three different technologies. The plot shows the average latency after 10 runs. The simulation was executed using PythonPDEVS, on a Lenovo Thinkpad X1 Carbon with a Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz processor with 16GB of memory available. The Operating System was Windows 10 Pro.

Out of the three options, MQTT appears to be best suited for this task, followed by ROS 2 and OPC UA respectively.

[1]https://www.ros.org/

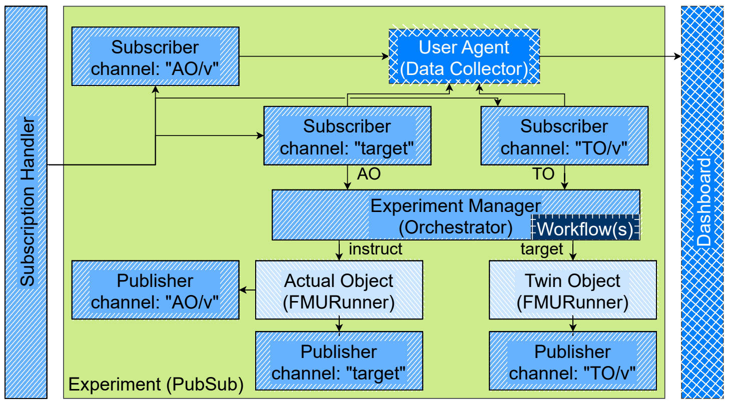

Subscription Handler
Subscriber channel: "AO/v"
User Agent (Data Collector)
Subscriber channel: "target"
Subscriber channel: "TO/v"
AO
TO
Experiment Manager (Orchestrator)
Workflow(s)
instruct
target
Publisher channel: "AO/v"
Actual Object (FMURunner)
Twin Object (FMURunner)
Publisher channel: "target"
Publisher channel: "TO/v"
Experiment (PubSub)
Dashboard

Figure 8.15: Publish Subscribe DEVS model (MQTT).

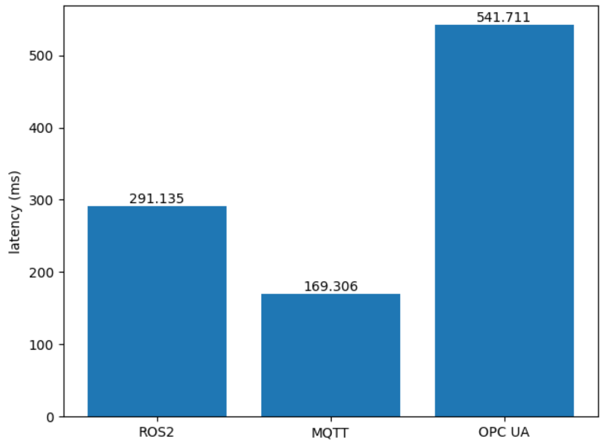


Figure 8.16: Average latency of different publish-subscribe technologies.

8.2.6.2 Different Communication Protocol Tests

The publish-subscribe approach is not the only possible solution. Maybe we would like to use a different technology, or maybe we would like the Twinning System to work *online*. This means that we will analyse the system *during runtime*, such that we can detect anomalies in the ship's velocity as soon as they occur. As a result, we need a communication protocol that is able to process information as-fast-as-possible, with minimal network delays.

As an example, we will compare a publish-subscribe setup, using MQTT[2], with a polling setup, using OPC UA[3], running on top of TCP/IP. A shared memory implementation is used as a baseline implementation. In Figure 8.17, a deployment diagram for the MQTT setup is shown.

[2]https://mqtt.org/
[3]https://opcfoundation.org/about/opc-technologies/opc-ua/

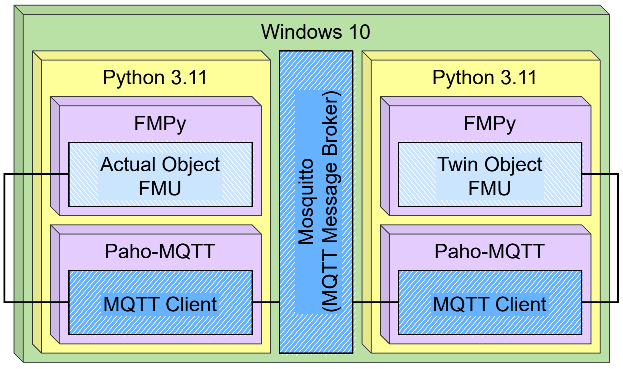


Figure 8.17: Deployment Diagram for MQTT experiment setup.

For each of the alternatives, a DEVS simulation model of the deployed architecture is constructed and evaluated. Figure 8.15 shows the publish-subscribe alternative, Figure 8.18 the polling alternative, and Figure 8.19 the shared memory alternative DEVS coupled model.

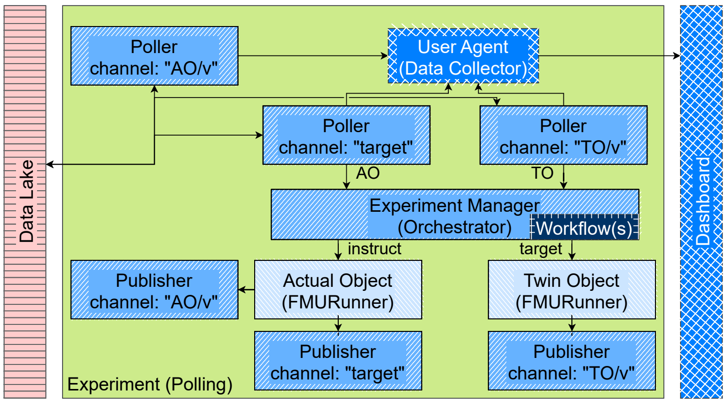

Data Lake
Poller channel: "AO/v"
User Agent (Data Collector)
Poller channel: "target"
Poller channel: "TO/v"
AO
TO
Experiment Manager (Orchestrator)
Workflow(s)
instruct
target
Publisher channel: "AO/v"
Actual Object (FMURunner)
Twin Object (FMURunner)
Publisher channel: "target"
Publisher channel: "TO/v"
Experiment (Polling)
Dashboard

Figure 8.18: Polling DEVS model (OPC UA on top of TCP/IP).

The *Dashboard* is a process that introspects the state of the *Data Collector* and plots the data every x time.

In Figure 8.19, the *Requester*s send a request message to access a variable in the *Memory* component. The latest value of the variable will be returned. In Figure 8.15, the *Publisher*s publish data on a communication channel and a *Subscription Handler* process applies a real-time interrupt in the *Subscriber*s of the same channel(s). Figure 8.18 is similar, but combines the variable request from Figure 8.19 and the external process from Figure 8.15.

Notice that we chose to let the orchestrator communicate directly with both AO and TO. An alternative would be to let this communication also happen via a resource-constrained communication channel.

Figure 8.20 shows a bar plot of the latency performance metric obtained after running the three different TEs on the same system, using the different communication technologies.

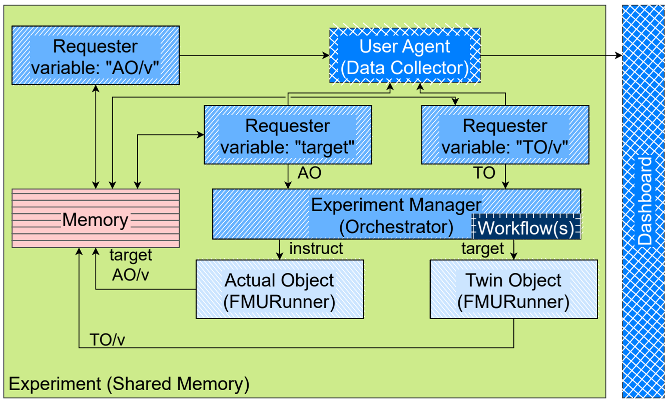


Figure 8.19: Shared Memory DEVS model.

Again, the plot shows the average latency after 10 runs. The simulation was executed using PythonPDEVS, on a Lenovo Thinkpad X1 Carbon with a Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz processor with 16GB of memory available. The Operating System was Windows 10 Pro.

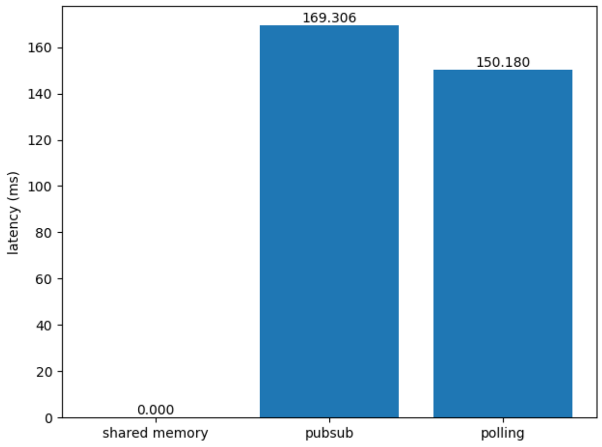


Figure 8.20: Average latency of different communication protocols.

As can be seen, using shared memory will be the most efficient approach, followed by polling. Publish-subscribe is thus the slowest of the three.

8.3 Conclusions

This chapter focuses on two specific use-cases in the nautical domain. They are a result of the *COOCK SmartPort 2025* project, emphasising the practical need for Twinning in this domain. They are not mere “toy” examples, as in Chapter 7, but are highly relevant as they are rooted in practical application. Both use-cases follow the workflow presented in Chapter 1 and discussed in detail in the sequential chapters.

Firstly, we focus on the idea of anomaly detection within a Twinned port context. Ships travel through this system and we keep track of the occupancy of the docks. Suddenly, we notice a discrepancy between the AO and the TO. The provided example can easily be extended to much more detailed systems, but already shows the importance of having similar techniques available. Next, we have shown how fault injection can be used in conjunction with anomaly detection to identify the specific reasons behind the anomaly that occurred. This use-case clearly pictures how multiple goals can be used to benefit the overall system.

Typically, deployment has the largest impact on the cost for constructing a Twinning System. Ideally, this can be quantified, as posed by *Research Question 4*. This brings us to the second use-case presented here: the simple 1D motion of a ship. It proposes a MBSE approach to *deployment space exploration*. In particular, architectural and technological alternatives are explicitly modelled using the DEVS formalism. Simulation of these models allows for quantitative evaluation and comparison of these alternatives. An example is the choice between ROS 2 and MQTT for communication between components. The simulation model may even be used as a basis for the ultimate realization of the Twinning System.

These use-cases focus on a completely different domain (and scale), compared to Chapter 7, thus allowing us to generalize the presented approach.

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."

– Douglas Adams,
The Long Dark Tea-Time of the Soul

Chapter 9
Conclusions and Future Work

The Twinning Paradigm appears everywhere, in a plethora of domains, and applied to any number of research questions. Combined with the ever-increasing size of complex CPS, the growth of AI, Big Data, and IoT; the Twinning Paradigm is an undoubtable pillar of Industry 4.0.

There is a lot of variability in the exact purposes for which TEs are built, as well as their internal architectures – either conceptual or realised. Additionally, the vast number of formalisms, languages, algorithms, technologies, and tools to choose from when creating such a system yields a large product family of possible solutions for any one of the problems that the Twinning Paradigm aims to solve.

9.1 Conclusions

There exist many different usages and applications of DTs in the literature, most of them are built quite ad-hoc with a lot of implicit (design) choices. The Twinning Paradigm is increasingly seen as a solution enabler for a host of problems in engineering and science. The variety of problems leads to many different solutions. Hence, a vast product family of twinning experiments appears. To tackle this variability, this thesis proposes a four-stage workflow, which will be applicable for engineering Twinning Systems. In each stage, variability may appear and choices have to be made by the Twinning System engineer.

Ⓐ **– Problem Space** We can construct a large feature tree based on existing sets of goals (Dalibor et al., 2022; Paredis et al., 2024). Based on these choices, a lot of implementation details are already made explicit. Kang and Lee (2013) identifies the individual requirements as *Goals*, *Usage Contexts*, and *Quality Assurance*. These are key purposes for which a Twinning System might be built. Additionally, there are the specific PoIs (Qamar & Paredis, 2012) that need to be considered. Chapter 3 delved deeper into these requirements and provided a non-exhaustive Feature Tree for them. This Feature Tree provides an answer to ***RQ1*** (*What are the most common reasons to build DTs?*).

Ⓑ **– Conceptual Architectures** A lot of conceptual architectures already exist. Some focus on the connectivity (Kritzinger et al., 2018), others emphasize on the system flow (Eramo et al., 2021), whereas others mainly focus on the internal components of the TO itself (Bolender et al., 2021). Chapter 4 showed a conceptual architecture with presence conditions that is used for all applications in the full scope of the Twinning Paradigm. Specific features could be enabled or disabled to yield a vast number of possibilities, yet only a handful are practically used. This provides an answer to ***RQ2*** (*Given the large number of existing DTs in the literature, can we unify?*): we can unify at a high, conceptual level, but need to be aware of the level of variability in solutions. The chapter also shows the different kinds of AOs and TOs that exist. The chapter revolves around TEs as first-class entities in the architecture. As the use of Twinning increases, the need to *combine* TEs arises. Hence, Chapter 6 showed how not only a single goal can be converted into a conceptual architecture, but multiple goals as well. It describes possible combinations for TEs, in turn answering ***RQ5*** (*How can we combine multiple DTs into a larger system?*).

Ⓒ – **Formalisms and Models** This stage mainly identifies the application of MBSE in this overall approach. MPM (Mosterman & Vangheluwe, 2004) can be used to reduce the scope of possible modelling languages.

Ⓓ – **Deployment** A large explosion of possibilities happens here, as there are many tools, frameworks and technologies that exist for creating DTs already. Chapter 5 provided an outline of these technologies, based on a small-scale analysis of the literature. Ideally, one would like to do some *deployment space exploration* and find the best possible set of solutions for deploying a Twinning System, *i.e.*, ***RQ4*** (*How to quantitatively support deployment choices?*). A proof-of-concept for this was discussed in Chapter 8.

This full timeline aims to answer ***RQ3*** (*What is the relationship between specific DT requirements, the system architecture, the used models, and the eventual deployment?*). Ideally, the decision procedure at every stage of the presented workflow can be automated or guided for a better construction of DTs. The hypothesis is that this thesis provides a foundation for doing this.

The power of this workflow and its usages is shown in Chapters 7 and 8 with a couple of example, yet representative, use-cases. As a whole, each use-case has focused on a different aspect that may be important when constructing a Twinning System: the LFR showed a generic workflow (in the form of an FTG+PM); the TurtleBot focused on the combination of TEs and the use of the *Historian*; the Port of Antwerp-Bruges use-case identified the need for combining multiple goals; and the 1D ship identified deployment space exploration. By using multiple domains, we can assume that the presented approach in this work is generalizable to the full scope of the Twinning Paradigm.

Over the entire discussion of these four stages, ***RQ3*** has been kept in mind, and the use-cases (Chapters 7 and 8) practically show how this over-arching relationship can be maintained as well.

9.2 Limitations and Threats to Validity

Most of this research is based on the academic literature. It is assumed there exist a lot of industrial cases that could have valuable insights into this framework, however, they might not have been published about (due to IP protection *etc.*). Chapter 6 has mainly been the result of high-level discussions, and are not compared to the literature, nor to individual proofs-of-context. As discussed in Section 9.4, this is still open research.

The presented proofs-of-context show a practical implementation of the individual components of the presented framework. By selecting these to be far apart, we postulate that this framework is applicable to the wide range in-between, however this is not proven, nor can it be proven. In reality, the literature should be (re-)analysed within the context of this framework to gain more certainty to this statement.

9.3 In Preparation

There are some papers currently in preparation. There is no guarantee that these will be published, but it is important to remark that these journal papers are only made possible because of the extensive research in this thesis.

The *COOCK SmartPort 2025* project with Port of Antwerp-Bruges has its results and research outlined already in a format for a journal paper. This paper is submitted to the journal of Transportation Engineering.

A combination of Chapter 6 and the TurtleBot use-case presented in the second half of Chapter 7 is being marked up for a journal, as the practical combination of TEs would benefit the scientific community. Having a practical and verified way of combining TEs will allow academia to focus on single TEs, thus leaving more room for scientific breakthroughs on a low level.

9.4 Future Work

This thesis has used research from the DT domain and has hypothesised it could be applied to the full spectrum of possibilities in the Twinning Paradigm. However, this hypothesis has not yet been verified. Hence, for a full understanding of the unification potential (*i.e., **RQ2***), the literature needs to be re-studied with this work in mind.

Furthermore, in Chapter 4, a non-exhaustive list of seven entities is presented to categorize the AO and the TO. This allows us to construct a matrix stating which combinations exist and which do not. It would be highly interesting to identify why certain combinations do (not) make sense. It would also be good to find situations where we can reap massive benefits from ATs; potentially through analog computers.

Chapter 4 introduces the top-level conceptual architecture for managing TEs, and Chapter 8 goes further into detail by applying it to a practical example. Yet, only the *Historian* is discussed. Having a detailed research on both the *Orchestrator*, as well as the *Reasoner* is still required.

While this thesis shows a workflow for constructing Twinning Systems, there is a gap in linking each stage of the process together (*i.e., **RQ3***). Ideally, a Knowledge Graph is constructed that allows a (partial) automation or guidance of (the decision procedure in) the workflow.

Chapter 6 discusses a high-level conceptual idea of how the presented architectures can be combined (*i.e.,* ***RQ5***). These combinations have not been verified against the full scope of the literature, nor by creating a detailed proof-of-concept. The TurtleBot use-case (Chapter 7) makes a first attempt at practically combining TEs, but this research still needs to be expanded on.

However, most of the future work of this thesis comes from the overly simplistic use-case scenarios in Chapters 7 and 8. In the future, the presented methodology can be applied to different application domains: robotics, production processes, waste water treatment, *etc.* Hence, more complicated (and real-world) use-cases need to be used to show and verify the validity of the provided workflow. There is a need for collaborating with companies to start focusing on actual real-world scenarios, as opposed to the academic cases presented in this thesis. Especially the combination of TEs (see also Chapter 6) needs to be verified and tested against all literature cases, as opposed to anecdotal examples.

Finally, given the possibilities of combining TEs, one situation has not been discussed in this thesis. This is the idea of Higher-Order Twinning Hierarchies (HOTH). HOTH focuses on building Twinning Systems of Twinning Systems; *i.e.,* investigating the usefulness (and practicality) of having a DT of a DT (DT-DT); or maybe an AT of an AT (AT-AT); or anything in-between. Note that this differs from the vertical federation identified in Chapter 6, but rather focuses on Twinning a fully deployed Twinning System as a whole. This idea might help the Twinning Paradigm to exist in a sustainable environment.

Chapter 10
Anecdotes

During one of the many meetings with prof. Hans Vangheluwe, we discussed the idea that the twin of a system does not always exist in the virtual world, *"like those images where Napoleon is moving pieces on a map in a war room"*. This birthed the idea of an Analog Twin (AT) (where the TO also exists in the physical world) and consequentially the *Twinning Paradigm* as a whole.

Additionally, that single quote sparked a well of interesting historical research into that topic, without any specific academic contributions. This appendix summarizes that research, as well as describing other anecdotes that were encountered whilst researching the origins of DTs.

10.1 Sand Tables

Etymologically, the word "abacus" comes from the ancient Greek word ἄβαξ (pronounced */ávaks/*[1], meaning "board" or "slab") (Fernandes, 2014), which might come from the Semitic אבק (pronounced */avak/*[2], meaning "dust"). Other etymologies have also been suggested (D. E. Smith, 1925).

During the middle ages in Western Europe, both *abax* and *abacus* were used to refer to a counting device. Nowadays, we commonly associate this with a counting board, but originally, it seems to have referred to a table covered with sand or fine dust (D. E. Smith, 1925).

The ancient Greek, Romans, Babylonians, and ancient Egyptians used this *Sand Table* (also known as a *Dust Abacus*) to learn how to write, draw, and do mathematics (specifically for geometry). A picture on the famous Darius Vase (created around 400-500 B.C., located in the Museum at Naples) is supposed to depict a Sand Table being used[3]. Medieval sources from Europe also mention the abax being used to help understand geometry and astrology. Surprisingly, multiple sources from that time indicate that Sand Tables were covered with *green* dust or sand (D. E. Smith, 1925).

[1] According to https://www.greek2ipa.com/.
[2] According to Google Translate for Hebrew.
[3] Even though there is some speculation about this.

At some point during the antiquity, armies started to use the Sand Table to draw maps of their battlefields and they used pebbles to represent their troops. If we consider the battlefield with troops as the SuS, we can identify the Sand Table as an abstracted model. The action on the battlefield influences the changes on the Sand Table, and from there, the generals can plan their next course of action. Within this context, we can consider an Sand Table as an AT.

The Sand Table has been used for military purposes throughout history, albeit using another name. *War Table*, *Map Table*, *War Room*, *Plotter*, *Tactical Map*, *Wargaming*, or *War-Planning Table* are all synonyms.

Sadly, no sources were found to indicate that Napoleon actually used a Sand Table, but it is extremely likely given his quote *"There is no man more pusillanimous than I when I am planning a campaign."* Additionally, detailed maps and tabletop map models were commonly used during that time (as hinted at in some sources of the American Civil War).

A *Sand Table Excercise* is still a common term in the military and pictures of Sand Tables can be found from the First and Second World War. The Battle of Britain Bunker, a control room near London, used numbered blocks on a "map table" in the Plotting Room[4] to indicate the positions of planes.

Modern versions of Sand Tables have embraced virtualization (Wisher, 2001), and especially AR[5]. Board games (like *Risk*, *Stratego*, *Battleships*, chess and checkers) are likely influenced by Sand Tables. Movies, series, and video games also easily show a Sand Table when depicting wars.

But even outside the context of war, Sand Tables are also still being used. Either in their original context (to learn how to draw and write), or in a more creative context as is the case for Antwerp Zoo. There, children are taught about deforestation and its impact on the tamarin population using a miniature model of the rainforest and decommissioned fire-hoses that represent roads or highways.

In short, the abax is the most common twin that is still being used to this day.

10.2 Apollo 13

On April 13th, 1970, James Arthur Lovell Jr. spoke the famous words "*Houston, we've had a problem here*". The crew of the Apollo 13 was 330,000 km away from Earth as an explosion in the oxygen tanks critically damaged their main engine: an issue that seemed almost impossible to solve from Earth. Fortunately, the National Aeronautics and Space Administration (NASA) was quick on the draw. The 15 training simulators for astronauts were recalibrated to match the oxygen tank explosion and resulting drift of the capsule. This way, experienced astronauts and engineers were able to come up with a solution for the issue and bring home the crew of the spacecraft. Many researchers (including NASA themselves) consider this feat the first DT[6] (Allen, 2021; Boschert & Rosen, 2016;

[4]https://battleofbritainbunker.co.uk/virtual-resources/

[5]https://en.wikipedia.org/wiki/Augmented_Reality_Sandtable

[6]Called "Virtual Digital Fleet Leader" at the time.

Ferguson, 2020).

In actuality, this twin was not really digital, but rather *electromechanical*. The training simulators were used to *physically* twin the spacecraft (even though some components might have been digital, the twin itself was not). This is therefore an example of an AT instead.

10.3 Alan Alda meets Alan Alda 2.0

The term *Digital Twin* in its current context dates back to 2002 (Grieves & Vickers, 2017). However, this was not the first usage of that term. That is attributed to the media and entertainment business, as can be seen in Figure 10.1.

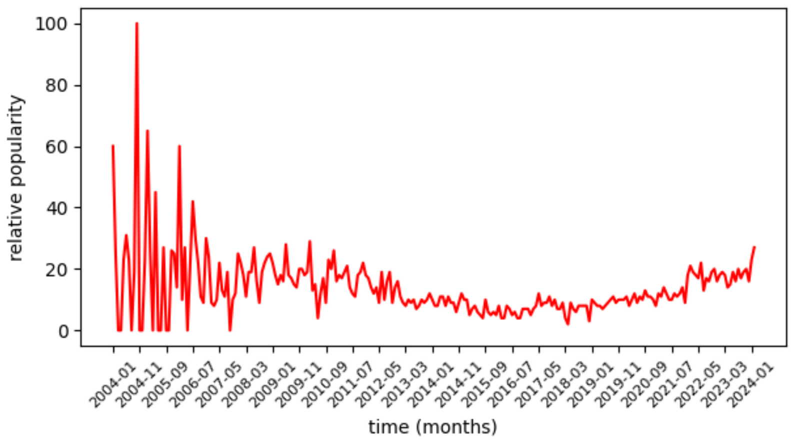


Figure 10.1: Worldwide Google Trends for the *Digital Twin* term in the *Media and Entertainment* category, as obtained on February 7th 2024.

Hodgins (1998) describes how physics simulations can be used to mimic human motion within the 3D (movie) animation industry. At that point in time, this was a relatively novel concept. A sidebar of that story, written by Alden M. Hayashi, was used as marketing for Scientific American Frontiers, season 8, episode 4: *"The Art of Science"*.

In the episode, host and Emmy-award-winning actor Alan Alda wonders what it would take for 3D animation (as used in Hollywood) to be indistinguishable from reality, with the technology that was available at the time. The goal of the project was to make a *"digital facsimile of* [Alan Alda's] *head that can talk – doing and saying things that the real* [actor] *has never done or said."*

Viewpoint DataLabs made a 3D scan of Alan Alda's face, which was given to animation studio Lamb & Company (nowadays called Lamb.com)[7] to provide facial expressions. Next, ATR Research[8] sampled the actor's voice and combined it with Text to Speech. This

[7]https://lamb.com/
[8]https://www.atr.jp/

was then merged with the animation at Lamb & Company to get a 1-minute conversation between Alan Alda and *Alan Alda 2.0*.

The project, code-named *Alan 2.0*, called the animated head a *Digital Twin* in the sidebar and during the episode.

10.4 Industrial Revolutions

We are currently at the forefront of the fourth major Industrial Revolution, also referred to as Industry 4.0. Yet, in order to understand why DTs form the major backbone for Industry 4.0, this section briefly summarizes the previous Industrial Revolutions as well.

10.4.1 First Industrial Revolution – Mechanization

English economic historian Arnold Toynbee popularized the term *Industrial Revolution* by referring to the British economic development from 1760 to 1840 (The Editors of Encyclopaedia Britannica, 2024b).

The drive force behind the First Industrial Revolution is the invention of a brand new energy source: steam power. Thomas Savery invented the *Miner's Friend* in 1698 (Jenkins, 1936), which eventually lead to the steam engine (generally accredited to James Watt, but numerous other inventors also had a hand in its conception).

This new energy source, together with a new general understanding of machinery and mechanization set the stage for a plethora of new machines. Most notably in the textile industry: Richard Arkwright's *spinning jenny* (1769) inspired Edmund Cartwright to invent the infamous *power loom* in 1784 (The Editors of Encyclopaedia Britannica, 2024a).

In their turn, the power loom, spinning jenny, and similar inventions allowed industry to skyrocket. New inventions for the iron industry (*e.g.,* automated bellows) allowed for more heat and therefore more iron production.

Steam-powered inventions filled the industrial landscape, solving increasingly larger and more complicated tasks. Engineers from Great Britain started inventing train locomotives and in 1835, *l'Eléphant* (the Elephant), *la Flèche* (the Arrow), and the *Stephenson* were the first passenger trains to travel the European mainland between Brussels and Mechelen (Canon Van Vlaanderen, 2024).

10.4.2 Second Industrial Revolution – Industrialization

While the concept of electricity has been around since the antiquity, it was mostly unobtainable until Alessandro Volta invented the first "battery" in 1800 (Routledge, 1881). Whereas most engineers still focused on steam power during that time, a new generation of inventors gradually switched to electrical power.

The introduction of the Bessemer process in 1856 (Bessemer, 1856), as well as other chemical processes allowed the iron industry to get yet another large update: the manufacturing of steel, which in turn also helped the large-scale manufacturing of steam trains and railroads.

Thomas Edison stole and bought a lot of new inventions, such that he could patent them. He acquired the idea of the light bulb in 1879 and was able to distribute these, thanks to "his" Direct Current lightning system (Kennedy, 2016). A year later, in 1870, a Cincinnati slaughterhouse used the very first automated production line, an idea that Henry Ford would reuse in 1913 for the first assembly lines (Nibert, 2011). Factories started to appear, which enabled mass-production and caused a ginormous economical influx, resulting in the *Roaring Twenties*.

10.4.3 Third Industrial Revolution – Digitization

Up to this point, inventions were purely mechanical, analog devices, facilitated by mathematics, chemistry, and the laws of physics. As a result, large-scale computations were slow and prone to many errors. But all of that changed with Turing et al. (1936) and Shannon (1938). Whereas Turing focused on automation and countability, Shannon worked on logical circuits and information theory. Their combined works formed the basis for current-day computers.

Turing was involved in the creation of the Automatic Computing Engine (ACE) in 1945, after having cracked the Enigma code. Meanwhile, the Americas were building the Electronic Numerical Integrator and Computer (ENIAC), generally considered the first computer.

The end of the war showed the power of nuclear fission, which paved the way to yet another new power source. On December 20th, 1951, the first electricity was generated using nuclear power (Hearst Magazines, 1952). This gradually caused the world to start building nuclear power plants to harness even more electricity.

1969 brought the inventions of the ARPANET (predecessor of the internet) and the PLC. In the 1980s, recorded music switched from cassettes and vinyl to CDs. Additionally, more people started using fax over telex. The world had fully entered the digital era.

10.4.4 Fourth Industrial Revolution – Robotization

Multiple countries have started to focus on their industrial productivity. In France and Italy, this is called the *Factory of the Future* (Davies, 2015). The UK focuses its Catapult centers within the context of *Made Smarter* (Davies, 2015; Yan & Li, 2020). In China, there is *Made in China 2025* (Tao & Zhang, 2017), as well as *Intelligent Manufacturing* (Yan & Li, 2020). The US focuses mainly on *Industrial Internet* (Yan & Li, 2020) and *Digital Engineering* (Mohamed et al., 2019; Rivera et al., 2022). Japan has its *Society 5.0* and Canada its *Digital Government* (Rivera et al., 2022). Famously, in Germany the term *Industrie 4.0* appeared around 2015 (Davies, 2015; Hankel & Rexroth, 2015).

This global shift in mentality is generally considered *Industry 4.0*, a term that mostly overlaps with similar movements and therefore has been adapted worldwide. While there is no clear new source of energy, we are globally shifting towards the usage of Green Energy (*e.g.*, windmills, solar panels, ...), which allows for a better and cleaner overall industry. Together with IoT, DTs and newer AI techniques, it is predicted that the economy marked will increase once more.

10.4.5 A Note on Industry 5.0

An Industrial Revolution can be defined as "*a rapid major change in an economy marked by the general introduction of power-driven machinery or by an important change in the prevailing types and methods of use of such machines*" according to Merriam Webster[9].

This definition provides a clear line throughout the previous sections: new power sources, developments, and inventions caused an increase in industry, positively impacting the global Gross Domestic Product (GDP), as can be seen in Figure 10.2.

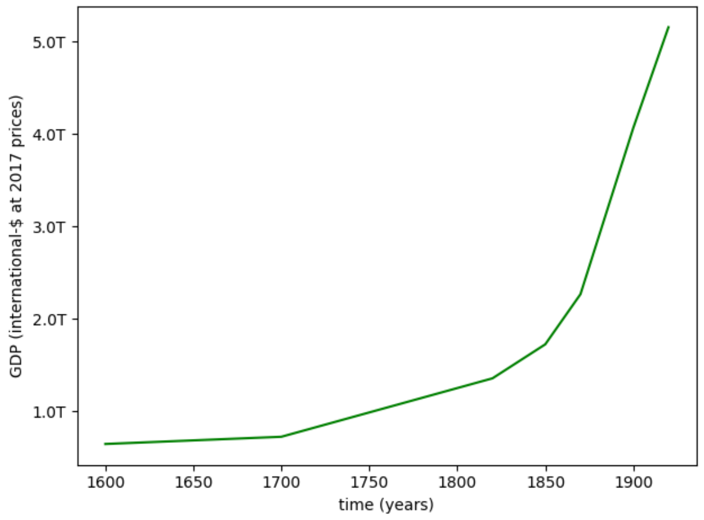
Around 1700 (Figure 10.2a), we can clearly see a steady impact of the First Industrial Revolution. Second Industrial Revolution also benefited from the momentum of the First, but it still introduced a new power source (*e.g.*, electricity), new machines (*e.g.*, trains, light bulbs, production lines, ...) and an additional increase in GDP (visible around 1850).

The Third Industrial Revolution profited from a plethora of wartime inventions, with its GDP increase visible around 1960 (Figure 10.2b). Finally, around 1990, the GDP increases even more, indicating the beginning of the Fourth Industrial Revolution.

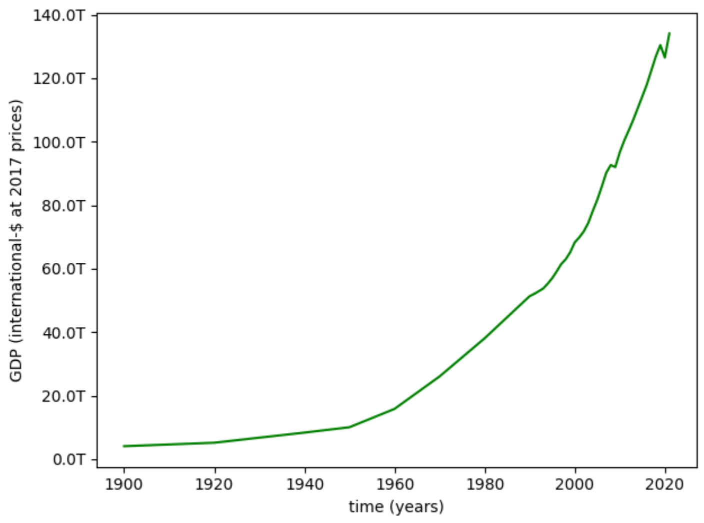
Industry 5.0 is the odd one out. It is seen as the step beyond Industry 4.0 (Yao et al., 2022), with a main focus on society, sustainability and human-robot interactions (*e.g.*, cobots). In 2021, the European Union proposed Industry 5.0 to *complement* Industry 4.0, with a main emphasis to take humans into account in this digitization craze. Yet, on its own it is still lacking the notion of an actual revolution. It should more accurately be described as Industry 4.1, or Industry 4.5. Raja Santhi and Muthuswamy (2023) argues to use *Industry 4.0S* as a name instead (*i.e.*, Industry 4.0 + Sustainability). Both Industry 4.0 and Industry 5.0 complement each other and should be seen as a whole when considering an industrial revolution.

Following the trends set out by the past, C. Zhang et al. (2023) describes that Industry 5.0 should better be seen as a "*future revolution*", as opposed to one happening now. Chourasia et al. (2022) even discusses Industry 6.0 as an interesting avenue for researchers to start exploring. And, whereas it is good to look towards the future, this concept might still be too premature to consider.

[9] https://www.merriam-webster.com/dictionary/industrial%20revolution



(a) Global GDP between 1600 and 1920.



(b) Global GDP between 1900 and 2021.

Figure 10.2: Global GDP between 1600 and 2021, based on data from Bolt and van Zanden (2020) and The World Bank (2023), with major data processing by Our World in Data (2023).

Chapter 11 pyCBD

In order to allow modelling using CBDs, a custom CBD simulator (`pyCBD`) was constructed, written in Python. The architecture and parallel execution of this simulator was based on the PythonPDEVS framework (Van Tendeloo & Vangheluwe, 2015).

Because CBDs are a graphical formalism, a https://www.draw.io (Draw.IO)[1] library was constructed for `pyCBD` and a corresponding model transformation tool was created. This way, models can be constructed in Draw.IO and then transformed to equivalent Python code. For instance, the CBD model shown in Figure 11.1a will yield the code shown in Figure 11.1c (excluding the `bounce` function). As a whole, the expected behaviour is shown in Figure 11.1b.

11.1 Simulator Features

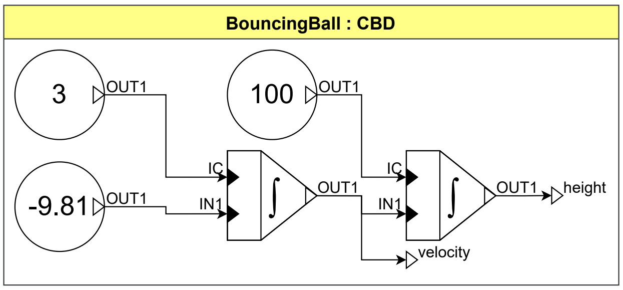
The `pyCBD` simulator has a lot of interesting features and Quality of Life aspects. An interested reader is referred to the detailed documentation of this framework, but some interesting aspects are summarized below.

Standard Library The most common CBD blocks are included in a standard library. Most of these are "atomic" (*i.e.,* they provide a single mathematical function), such that composition is easy. This ensures that users do not need to re-create the same blocks over and over again.

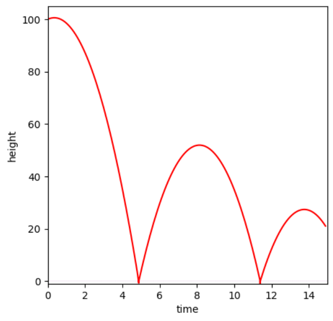
Additionally, a backwards Euler `IntegratorBlock` and `DerivatorBlock` is included, as well as the commonly used `AddOneBlock`. These last three are coupled CBD blocks, made up of the other mathematical functions. Chapter 11 shows a detailed list of all the blocks that can be found in this library.

Hierarchical Composition and Flattening Because a CBD model encapsulates a single mathematical function, it can easily embedded in another CBD model, thus yielding hierarchy in the model structure. Figure 11.2a shows the top-level of a simple

[1]Formerly known as https://www.diagrams.net



(a) CBD model of the Bouncing Ball, using Draw.IO.



(b) Plot of the "bounce".

```python
class BouncingBall(CBD):
  def __init__(self, k=0.7, v0=3, y0=100):
    super().__init__("BouncingBall", output_ports=["height", "velocity"])
    self.k = k   # The elasticity of the bounce

    self.addBlock(ConstantBlock("g", -9.81))
    self.addBlock(ConstantBlock("v0", v0))
    self.addBlock(ConstantBlock("y0", y0))
    self.addBlock(IntegratorBlock("v"))
    self.addBlock(IntegratorBlock("y"))

    self.addConnection("g", "v")
    self.addConnection("v", "y")
    self.addConnection("v", "velocity")
    self.addConnection("y", "height")
    self.addConnection("v0", "v", input_port_name="IC")
    self.addConnection("y0", "y", input_port_name="IC")

  def bounce(self):
    v_pre = self.getSignalHistory("velocity")[-1].value
    v_new = -v_pre * self.k
    self.getBlockByName("v0").setValue(v_new)
    self.getBlockByName("y0").setValue(0.0)
```

(c) Generated Python code for the Bouncing Ball model.

Figure 11.1: BouncingBall example CBD model.

`EvenNumberGenerator`. The `AddOneBlock` (*i.e.,* the component marked with "+1") is in itself another CBD block that has the structure shown in Figure 11.2b. The Draw.IO to CBD exporter allows these compositions by defining multiple custom blocks (*i.e.,* defining both Figure 11.2a and Figure 11.2b), or by the "syntactic sugar" Quality of Life approach shown in Figure 11.2c.

The hierarchical blocks can be trivially flattened as shown in Figure 11.2d. The small dashed circles[2] indicate the ports that were "dissolved" during flattening. In order to maintain the same behaviour, an internal lookup table is used to keep references to these ports.

Scaled Real-Time Simulation Besides the standard *as-fast-as-possible* simulation, it is possible to run the simulator in (scaled) real-time. This is interesting if we want to use a dashboard, or plot the change over time in a longer simulation. This feature was fully based on PythonPDEVS.

Tracing Another feature based on PythonPDEVS is tracing. Sometimes it is useful to log the behaviour of the model. Arguably, for CBDs it might be more useful to have a visual debugger than a tracer.

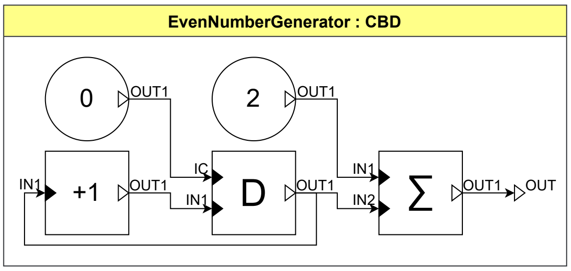
Publish-Subscribe Architecture The CBD simulator has a Publish-Subscribe architecture that allows users to hook custom functions on events. For instance, it is possible to change the value of a `ConstantBlock` after every iteration, or signal a user that a simulation has ended.

Dashboarding Just like in PythonPDEVS, `TkInter`[3] can be used to construct complex dashboards. These can make use of tracers, or the Publish-Subscribe hooks that are included.

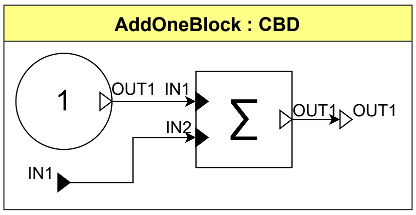
Live Plotting Framework An additional module in the CBD framework appears to work well with PythonPDEVS, which is a Live Plotting Framework. By using the hooks and some wrappers around `matplotlib`[4], `seaborn`[5] and `bokeh`[6], (scaled) real-time simulations can visualize their internal signals during runtime.

Adaptive Step-Size Also known as *varying step-size*, *adaptive step-size* varies the time interval between computations. Normally, a fixed Δt is used as the time delay between two sequential computations. All values in-between are assumed using interpolation. This method is called the *fixed step-size* approach. Using a high Δt, a lot of precision is lost, but the computational power required is minimal. A low Δt maintains the precision,

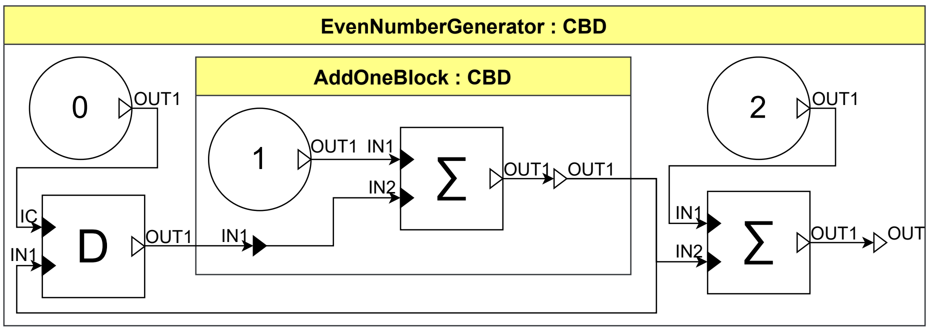
[2]Which are not part of the Draw.IO library.
[3]https://docs.python.org/3/library/tkinter.html
[4]https://matplotlib.org/
[5]https://seaborn.pydata.org/
[6]https://docs.bokeh.org/en/latest/index.html



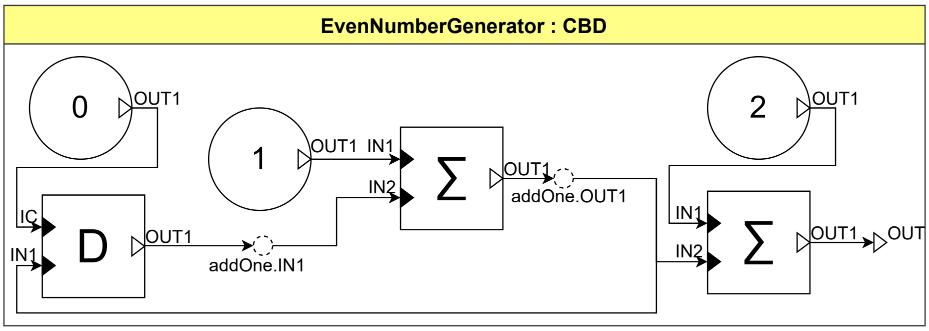
(a) CBD model of an `EvenNumberGenerator`, using Draw.IO.



(b) Contents of the `AddOneBlock` model, using Draw.IO.



(c) Quality of Life version of the `EvenNumberGen` model, using Draw.IO.



(d) Flattened version of the `EvenNumberGenerator`, using Draw.IO.

Figure 11.2: `EvenNumberGenerator` example CBD model.

but requires a lot of (potentially unnecessary) computations. A slowly evolving function is therefore ideally computed with a high Δt, and a highly fluctuating function needs a low Δt. Sometimes, a function is both highly fluctuating in certain domains and slowly evolving in another domain. This requires us to *vary* the Δt through time[7].

By default, a *fixed step-size* is used (where the user sets a Δt). This results in a fixed-rate clock to be added to the model during its simulation. However, the clock's h value can be altered to change the time in-between computations, based on the signal values of a certain computation.

A Runge-Kutta preprocessor is included that allows the full family of Runge-Kutta algorithms (including, but not limited to) RKF45 (Fehlberg, 1969), by using their Butcher Tables (Butcher, 1964).

Hybrid Simulations The simulator can be easily embedded into DEVS, as was shown in Paredis, Denil, and Vangheluwe (2021). Other embeddings should work as well (but are not tested, or verified).

State-Event Location The core feature of hybrid simulations is called State-Event Location. This is because certain state changes might happen when a signal falls below a certain value. For instance, in the bouncing ball example, we want to know exactly *when* the ball hits the floor, not *if* it has gone through the floor between the current and the previous iteration.

To accomplish this task, a simple linear translation can be combined with a root-finding algorithm in order to know the exact point in time of when the state-event has occurred. The CBD simulator will iteratively jump forwards and backwards through time (thus solving the root-finding algorithm) until it has found the exact point in time when this happened. If this moment is found, a potential manipulation may happen to the model (*e.g.*, adding new blocks, or reversing the velocity) before the simulation is restarted.

Exports Because of the equivalence between ODEs and CBDs, it stands to reason that CBD models can be transformed into systems of ODEs. The generic algorithm for this was discussed in Gomes et al. (2020). An export module is included in the framework to allow models to be transformed into LaTeX math equations. Either the full set of equations can be generated, or a (simplified) minimized set can be yielded. For educational purposes, all intermediary simplification steps can also be returned.

Additionally, it is also possible to export to FMU.

11.2 Visual Syntax in DrawIO

The standard CBD library that is included in the CBD simulator framework, including all visual block representations, was made with Draw.IO. All blocks are shown in Table 11.1.

[7] A varying Δt is often called h in the mathematical literature on this topic.

Table 11.1: Standard library for the CBD framework, including the graphical notations, created in Draw.IO.

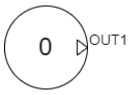
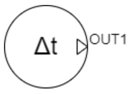
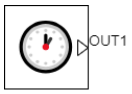
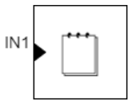
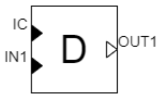
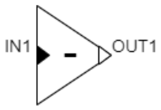
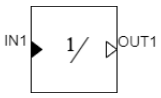
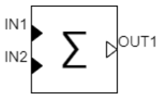
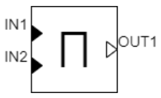
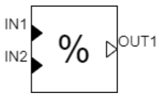
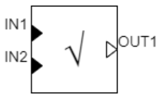
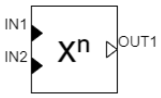
Block Representation	Name	Description
	`ConstantBlock`	Constructed with a value v. Outputs v.
	`DeltaTBlock`	Outputs the current Δt.
	`TimeBlock`	Outputs the current time.
	`LoggingBlock`	Prints a message to the console with a certain priority, based on the received input.
	`DelayBlock`	Delays the input for one iteration. In the first iteration, the value of `IC` is output.
	`NegatorBlock`	Outputs `-IN1`.
	`InverterBlock`	Outputs `1/IN1`.
	`AdderBlock`	Constructed with n inputs. Outputs the sum of all inputs.
	`ProductBlock`	Constructed with n inputs. Outputs the product of all inputs.
	`ModuloBlock`	Outputs `IN1 mod IN2`.
	`RootBlock`	Outputs $\sqrt[IN2]{IN1}$.
	`PowerBlock`	Outputs $IN1^{IN2}$.

Table 11.2: Standard library for the CBD framework (cont.).

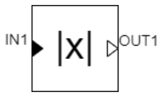
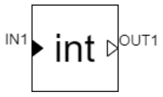
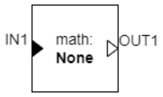
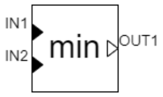
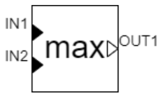
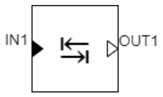
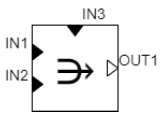
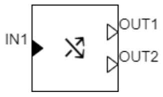
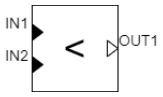
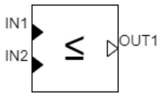
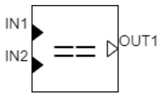
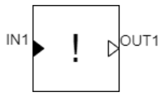
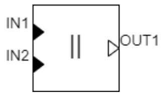
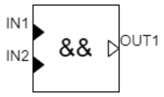
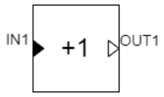
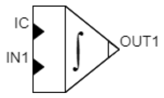
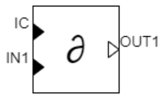
Block Representation	**Name**	**Description**
	`AbsBlock`	Outputs the absolute value of its input.
	`IntBlock`	Outputs the (floored) integer value of its input.
	`GenericBlock`	Allows a user to set a mathematical function that will be executed on the input.
	`MinBlock`	Outputs the smallest value of its inputs.
	`MaxBlock`	Outputs the largest value of its inputs.
	`ClampBlock`	Clamps the input between a minimum and a maximum.
	`BlendBlock`	"Blends" both the inputs, *i.e.,* it performs a linear interpolation between `IN1` and `IN2`, using `IN3` to weight them. This function is known as the "`mix`" function in most shader languages. It outputs $(IN3 \cdot IN1) + ((1 - IN3) \cdot IN2)$.
	`SplitBlock`	Constructed with n outputs. Produces the input on all outputs.
	`LessThanBlock`	Outputs 1 if $IN1 < IN2$, otherwise 0.
	`LessThan-OrEqualsBlock`	Outputs 1 if $IN1 \leq IN2$, otherwise 0.
	`EqualsBlock`	Outputs 1 if $IN1 = IN2$, otherwise 0.

Table 11.2: Standard library for the CBD framework (cont.).

Block Representation	**Name**	**Description**
	`NotBlock`	Outputs 0 if the input is "truthy", otherwise 1.
	`OrBlock`	Constructed with n inputs. Outputs 1 if one of the inputs is "truthy", otherwise 0.
	`AndBlock`	Constructed with n inputs. Outputs 1 if all of the inputs are "truthy", otherwise 0.
	`AddOneBlock`	Hierarchical CBD. Outputs the input increased by one.
	`IntegratorBlock`	Hierarchical CBD. Applies the backwards Euler integration function. Outputs `IC` during the first iteration.
	`DerivatorBlock`	Hierarchical CBD. Applies the backwards Euler derivation function. Outputs `IC` during the first iteration.

Acronyms

AAS Asset Administration Shell 12, 13, 30, 31, 45, 51, 58
ACE Automatic Computing Engine 139
AGV Automated Guided Vehicle iv, ix, xxvi, 81, 84, 89, 97, 99, 108
AI Artificial Intelligence iii, vii, xiii, 2, 13, 31, 35, 52, 53, 131, 140
AnSyMo Antwerp Systems and software Modelling xiii, xvi, 8, 109
AO Actual Object iii, vii, xxvi, xxviii, 5, 10, 11, 27–32, 34–37, 39–49, 56, 63–67, 69, 73, 74, 78, 79, 82, 99, 101–103, 105–110, 112, 113, 115, 116, 122–124, 128, 130, 132, 133
API Application Programming Interface 40, 79, 101
AR Augmented Reality 18, 29, 52, 57, 136
AT Analog Twin 10, 133–137

CBD Causal-Block Diagram xii, xvi, xxvii, xxviii, 16, 18, 19, 52, 54, 87, 95–97, 143–150
CoM Center of Mass 88, 89, 99, 100
CPS Cyber-Physical System iii, vii, 1, 2, 13, 23, 29, 35, 46, 120, 131

DDS Data Distribution Service iv, viii, 23, 24
DEVS Discrete EVent system Specification iv, v, viii, ix, xii, xiii, xxviii, 15, 16, 18, 21, 22, 54, 59, 69, 104, 109, 112, 113, 115, 124, 126–130, 147
Draw.IO https://www.draw.io 143–148
DT Digital Twin iii, vii, xxv, xxviii, 2–5, 7, 9–14, 23–25, 28, 32, 35–37, 39, 42, 46–49, 51–53, 58, 61, 62, 79, 82, 89, 101, 102, 131–138, 140
DTDL Digital Twin Design Language 54
DTW Dynamic Time Warping 113, 116–118

EM Experiment Manager 40–42, 44, 45, 49, 63–71, 73–76, 79
ENIAC Electronic Numerical Integrator and Computer 139

FIPA Foundation for Intelligent Physical Agents 56
FMI Functional Mock-Up Interface 21, 54
FMU Functional Mock-Up Unit 16, 18, 21, 33, 79, 122–124, 147
FTG Formalism Transformation Graph xxv, 7, 14–16
FTG+PM Formalism Transformation Graph and Process Model xi, 14, 16, 17, 82, 132

GDP Gross Domestic Product xxviii, 140, 141
GIS Geographic Information System 53
GPSS General Purpose Simulation System xiii, 15

HOTH Higher-Order Twinning Hierarchies 134

IoT Internet of Things iii, vii, 4, 23, 52, 53, 131, 140
IP Intellectual Property 33

LFR Line Following Robot iv, ix, xi, xxvi, xxvii, 8, 57, 81–85, 88, 89, 91, 92, 94–101, 108, 132

M&S Modelling & Simulation iii, vii, 1, 3–5, 14, 56
MBSE Model-Based Systems Engineering iv, viii, 1, 7, 14, 27, 58, 120, 124, 130, 132
MoSIS Modelling of Software-Intensive Systems 109
MPM Multi-Paradigm Modelling iii, iv, vii, viii, 1, 2, 7, 9, 13, 14, 18, 54, 58, 124, 132

NASA National Aeronautics and Space Administration 136

ODE Ordinary Differential Equation xxvii, 15, 19, 21, 54, 95, 104, 106, 123, 147

PID Proportional-Integral-Differential 92–94, 123

PLC Programmable Logic Controller 57, 139

PLM Product Lifecycle Management xxv, 9, 36, 37, 53, 73

PM Process Model xxv, 14, 16, 17, 82, 99, 108

PoI Property of Interest iii, iv, vii, viii, xxvi, 5, 8, 26, 27, 37, 45, 61, 63–65, 68–70, 73, 74, 78, 79, 102, 105, 106, 112, 131

RAMI 4.0 Reference Architecture Model for Industry 4.0 2, 12, 24, 39, 45, 58

RKF45 Runge-Kutta-Fehlberg of 4th and 5th order 20, 147

ROS Robot Operating System 24, 56, 101, 107, 126, 130

SLAM Simultaneous Localization and Mapping 104

STE(A)M Science, Technology, Education, (Arts,) Mathematics 81

SuS System under Study iii, vii, 4, 5, 10, 26, 27, 32, 36, 37, 39, 40, 45, 46, 123, 136

TE Twinning Experiment iv, viii, ix, xiv, xxv–xxvii, 5, 37, 39–45, 47, 49, 51, 52, 55, 56, 61–81, 101–105, 108, 128, 131–134

TFSA Timed Finite State Automata xxvii, 15, 16, 104, 106

TO Twin Object iii, vii, xxvi–xxviii, 5, 10–12, 28–36, 39–41, 43–49, 56, 62–67, 69–71, 73, 74, 78, 79, 82, 96, 99, 101–103, 106, 108–110, 112, 113, 115, 116, 122–124, 128, 130, 132, 133, 135

TPR Department of Transport and Regional Economics xiii, xvi, 8, 109

VR Virtual Reality 18, 29, 52, 57

WSN Wireless Sensor Network 56

XR Mixed Reality 18, 29, 52, 57

Bibliography

Ackoff, R. L. (1971). Towards a system of systems concepts. *Management science*, *17*(11), 661–671.

Akroyd, J., Mosbach, S., Bhave, A., & Kraft, M. (2021). Universal digital twin-a dynamic knowledge graph. *Data-Centric Engineering*, *2*, e14.

Ali, S., Arcaini, P., & Arrieta, A. (2025). Foundation Models for the Digital Twins Creation of Cyber-Physical Systems. In T. Margaria & B. Steffen (Eds.), *Leveraging applications of formal methods, verification and validation. application areas* (pp. 9–26). Springer Nature Switzerland.

Allen, B. (2021). Digital Twins and Living Models at NASA [Keynote]. https://ntrs.nasa.gov/citations/20210023699

Alvarez, S. (2019). Tesla cars can now order parts for itself when in need of service repair [Accessed: June 24th 2024]. https://www.teslarati.com/tesla-repairs-service-automatic-pre-order-parts/

Angjeliu, G., Coronelli, D., & Cardani, G. (2020). Development of the simulation model for Digital Twin applications in historical masonry buildings: The integration between numerical and experimental reality. *Computers & Structures*, *238*, 106282. https://doi.org/10.1016/j.compstruc.2020.106282

Antunes, A. (2023). Designing a Digital Twin for Adaptive Serious Games-based Therapy. *Proceedings of the 22nd International Conference on Mobile and Ubiquitous Multimedia*, 574–576.

Arcaute, E., Barthelemy, M., Batty, M., Caldarelli, G., Gershenson, C., Helbing, D., Moreno, Y., Ramasco, J. J., Rozenblat, C., & Sánchez, A. (2021). Future cities: Why digital twins need to take complexity science on board. *ResearchGate*.

Åström, K. J. (2002). Control System Design: Lecture Notes for ME 155A [Accessed: 24th of April 2024]. https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom.html

Åström, K. J., Elmqvist, H., & Mattsson, S. E. (1998). Evolution of Continuous-Time Modeling and Simulation. *Proc. of the 12th European Simulation Multiconference, ESM'98*, 9–18.

Autiosalo, J., Vepsäläinen, J., Viitala, R., & Tammi, K. (2019). A feature-based framework for structuring industrial digital twins. *IEEE access*, *8*, 1193–1208.

Aydt, H., Turner, S. J., Cai, W., & Low, M. Y. H. (2008). Symbiotic simulation systems: An extended definition motivated by symbiosis in biology. *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*, 109–116.

Baalbergen, E. H., de Marchi, J. A., & Klerx, R. (2023). Unleashing the potentials of digital twinning in the production of composite aircraft components. *Journal of Physics: Conference Series*, *2526*(1), 012046.

Badawi, H. F., Laamarti, F., & El Saddik, A. (2021). Devising Digital Twins DNA Paradigm for Modeling ISO-Based City Services. *Sensors*, *21*(4). https://doi.org/10.3390/s21041047

Bader, S., Barnstedt, E., Bedenbender, H., Berres, B., Billmann, M., & Ristin, M. (2022). Details of the asset administration shell-part 1: The exchange of information between partners in the value chain of industrie 4.0 (version 3.0 rc02).

Balachandar, S., & Chinnaiyan, R. (2019). Reliable digital twin for connected footballer. *International Conference on Computer Networks and Communication Technologies: ICCNCT 2018*, 185–191.

Ballouch, Z., Hajji, R., Poux, F., Kharroubi, A., & Billen, R. (2022). A prior level fusion approach for the semantic segmentation of 3d point clouds using deep learning. *Remote Sensing*, *14*(14), 3415. https://doi.org/10.3390/rs14143415

Balta, E. C., Pease, M., Moyne, J., Barton, K., & Tilbury, D. M. (2021). Digital Twin based cyber-attack detection for manufacturing systems. *2021 Winter Simulation Conference*, *2021*.

Banerjee, S., Das, D., Chatterjee, P., & Ghosh, U. (2023). Blockchain-enabled digital twin technology for next-generation transportation systems. *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, 224–229.

Barosan, I., Basmenj, A. A., Chouhan, S. G. R., & Manrique, D. (2020). Development of a Virtual Simulation Environment and a Digital Twin of an Autonomous Driving Truck for a Distribution Center. *European Conference on Software Architecture*, 542–557.

Batty, M. (2021). Multiple models.

Bauer, P., Stevens, B., & Hazeleger, W. (2021). A digital twin of Earth for the green transition. *Nature Climate Change*, *11*(2), 80–83.

Becker, F., Bibow, P., Dalibor, M., Gannouni, A., Hahn, V., Hopmann, C., Jarke, M., Koren, I., Kröger, M., Lipp, J., et al. (2021). A conceptual model for digital shadows in industry and its application. *Conceptual Modeling: 40th International Conference, ER 2021, Virtual Event, October 18–21, 2021, Proceedings 40*, 271–281.

Bellis, S., & Denil, J. (2022). Challenges and possible approaches for sustainable digital twinning. *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 643–648.

Bergero, F., & Kofman, E. (2011). PowerDEVS: a Tool for Hybrid System Modeling and Real-Time Simulation. *Simulation*, *87*, 113–132.

Berti, N., Finco, S., Guidolin, M., & Battini, D. (2023). Towards Human Digital Twins to enhance workers' safety and production system resilience. *IFAC-PapersOnLine*, *56*(2), 11062–11067.

Bessemer, H. (1856). Improvement in the manufacture of iron and steel [Patent].

Bibow, P., Dalibor, M., Hopmann, C., Mainz, B., Rumpe, B., Schmalzing, D., Schmitz, M., & Wortmann, A. (2020). Model-driven development of a digital twin for injection molding. *International Conference on Advanced Information Systems Engineering*, 85–100.

Biesinger, F., Meike, D., Kraß, B., & Weyrich, M. (2018). A Case Study for a Digital Twin of Body-in-White Production Systems General Concept for Automated Updating of Planning Projects in the Digital Factory. *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 19–26. https://doi.org/10.1109/ETFA.2018.8502467

Blew, R. D. (1996). On the definition of ecosystem. *Bulletin of the Ecological Society of America*, *77*(3), 171–173.

Bojarczuk, K., Gucevska, N., Lucas, S., Dvortsova, I., Harman, M., Meijer, E., Sapora, S., George, J., Lomeli, M., & Rojas, R. (2021). Measurement challenges for cyber cyber digital twins: Experiences from the deployment of Facebook's ww simulation system. *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–10.

Bolender, T., Bürvenich, G., Dalibor, M., Rumpe, B., & Wortmann, A. (2021). Self-adaptive manufacturing with digital twins. *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 156–166.

Boletsis, C. (2022). The Gaia System: A tabletop projection mapping system for raising environmental awareness in islands and coastal areas. *Proceedings of the 15th International Conference on PErvasive Technologies Related to Assistive Environments*, 50–54.

Bolt, J., & van Zanden, J. L. (2020). *The Maddison Project: Maddison style estimates of the evolution of the world economy. A new 2020 update* (tech. rep.). University of Groningen.

Bonney, M. S., de Angelis, M., Wagg, D., & Dal Borgo, M. (2021). Digital Twin Operational Platform for Connectivity and Accessibility using Flask Python. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 237–241.

Borland, S., & Vangheluwe, H. (2003). Transforming statecharts to devs. *Proceedings of the 2003 Summer Simulation Conference*, 154–159.

Boschert, S., Heinrich, C., & Rosen, R. (2018). Next generation digital twin.

Boschert, S., & Rosen, R. (2016). Digital Twin – the Simulation Aspect. In *Mechatronic futures* (pp. 59–74). Springer.

Boss, B., Malakuti, S., Lin, S. W., Usländer, T., Clauer, E., Hoffmeister, M., & Stojanovic, L. (2020). Digital twin and asset administration shell concepts and application in the industrial internet and industrie 4.0. *Industrial Internet Consortium: Boston, MA, USA*.

Brannon, R. M. (2004). Curvilinear analysis in a Euclidean space. *University of New Mexico*.

Brockhoff, T., Heithoff, M., Koren, I., Michael, J., Pfeiffer, J., Rumpe, B., Uysal, M. S., Van Der Aalst, W. M. P., & Wortmann, A. (2021). Process prediction with digital twins. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 182–187.

Burger, A., Koziolek, H., Rückert, J., Platenius-Mohr, M., & Stomberg, G. (2019). Bottleneck identification and performance modeling of OPC UA communication models. *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 231–242.

Butcher, J. C. (1964). On Runge-Kutta processes of high order. *Journal of the Australian Mathematical Society*, *4*(2), 179–194. https://doi.org/10.1017/S1446788700023387

Camus, B., Paris, T., Vaubourg, J., Presse, Y., Bourjot, C., Ciarletta, L., & Chevrier, V. (2018). Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware. *SIMULATION*, *94*(12), 1099–1127.

Canon Van Vlaanderen. (2024). De eerste treinrit - Het spoor ontsluit het land [Accessed: 29th of March 2024]. https://www.canonvanvlaanderen.be/events/de-eerste-treinrit/

Carnap, R. (1936). Testability and meaning. *Philosophy of science*, *3*(4), 419–471.

Carreira, P., Amaral, V., & Vangheluwe, H. (2020). Multi-Paradigm Modelling for Cyber-Physical Systems: Foundations. In *Foundations of multi-paradigm modelling for cyber-physical systems* (pp. 1–14). Springer International Publishing. https://doi.org/10.1007/978-3-030-43946-0_1

Cavalieri, S., & Salafia, M. G. (2020). Insights into mapping solutions based on OPC UA information model applied to the industry 4.0 asset administration shell. *Computers*, *9*(2), 28.

Cederbladh, J., Cleophas, L., Kamburjan, E., Lima, L., & Vangheluwe, H. (2023). Symbolic Reasoning for Early Decision-Making in Model-Based Systems Engineering. *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 721–725.

Charette, R. N. (2009). This car runs on code. *IEEE spectrum*, *46*(3), 3.

Choppy, C., Klai, K., & Zidani, H. (2011). Formal verification of UML state diagrams: a petri net based approach. *ACM SIGSOFT Software Engineering Notes*, *36*(1), 1–8.

Chourasia, S., Tyagi, A., Pandey, S. M., Walia, R. S., & Murtaza, Q. (2022). Sustainability of Industry 6.0 in Global Perspective: Benefits and Challenges. *MAPAN*, *37*(2), 443–452. https://doi.org/10.1007/s12647-022-00541-w

Custodio, L., & Machado, R. (2020). Flexible automated warehouse: A literature review and an innovative framework. *The International Journal of Advanced Manufacturing Technology*, *106*, 533–558.

Czarnecki, K. (2004). Overview of generative software development. In J. Banâtre, P. Fradet, J. Giavitto, & O. Michel (Eds.), *Unconventional programming paradigms, international workshop UPP, revised selected and invited papers* (pp. 326–341, Vol. 3566). Springer. https://doi.org/10.1007/11527800_25

Czarnecki, K., Helsen, S., et al. (2003). Classification of model transformation approaches. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, *45*(3), 1–17.

Czarnecki, K., Østerbye, K., & Völter, M. (2002). Generative programming. In J. H. Núñez & A. M. D. Moreira (Eds.), *Object-oriented technology, ECOOP 2002 workshops and posters* (pp. 15–29, Vol. 2548). Springer. https://doi.org/10.1007/3-540-36208-8_2

D'Abreu, M. C., & Wainer, G. A. (2005). M/CD++: Modeling continuous systems using Modelica and DEVS. *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 229–236.

Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., & Wortmann, A. (2022). A Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins. *Journal of Systems and Software*, *193*, 111361. https://doi.org/10.1016/j.jss.2022.111361

Damköhler, F. (2022). Mastering the Nordschleife with hydrogen [Accessed: 31st of May 2024]. https://newsroom.porsche.com/en/2022/innovation/porsche-engineering-simulation-hydrogen-combustion-engines-nuerburgring-nordschleife-29401.html

David, I., Archambault, P., Wolak, Q., Vu, C. V., Lalonde, T., Riaz, K., Syriani, E., & Sahraoui, H. (2023). Digital Twins for Cyber-Biophysical Systems: Challenges and Lessons Learned. *ACM/IEEE 26th International Conference on Model-Driven Engineering Languages and Systems (MODELS). IEEE*, 1–12.

David, I., & Bork, D. (2024). Infonomics of Autonomous Digital Twins. *Advanced Information Systems Engineering - 36th International Conference, CAiSE 2024, Limassol, Cyprus, 2024, Proceedings*.

David, I., Shao, G., Gomes, C., Tilbury, D., & Zarkout, B. (2025). Interoperability of Digital Twins: Challenges, Success Factors, and Future Research Directions. In T. Margaria & B. Steffen (Eds.), *Leveraging applications of formal methods, verification and validation. application areas* (pp. 27–46). Springer Nature Switzerland.

Davies, R. (2015). Industry 4.0: Digitalisation for productivity and growth.

Denil, J., Meulenaere, P. D., Demeyer, S., & Vangheluwe, H. (2017). DEVS for AUTOSAR-based system deployment modeling and simulation. *SIMULATION*, *93*(6), 489–513. https://doi.org/10.1177/0037549716684552

Denil, J., Meyers, B., De Meulenaere, P., & Vangheluwe, H. (2015). Explicit Semantic Adaptation of Hybrid Formalisms for FMI Co-Simulation. *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, 99–106.

Denil, J., Vangheluwe, H., De Meulenaere, P., & Demeyer, S. (2012). Calibration of deployment simulation models: A multi-paradigm modelling approach. *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium*, 1–8.

Denis, H., Paredis, R., Albertins, P., Vangheluwe, H., Farzadmehr, M., Carlan, V., Vanelslander, T., Luong, N.-Q., & Mercelis, S. (2025). Towards Smart Port of the Future: Harnessing Ai and Simulation Models for Nautical Chain Optimization. *Transportation Engineering*.

de Weck, O. L., Roos, D., Magee, C. L., & Vest, C. M. (2011). Life-Cycle Properties of Engineering Systems: The Ilities. In *Engineering Systems: Meeting Human Needs in a Complex Technological World* (pp. 65–96). MIT Press.

Diakité, M., & Traoré, M. K. (2023). Formal Approach to Digital Twin Specification. *2023 Annual Modeling and Simulation Conference (ANNSIM)*, 233–244.

Doellner, J., Merino Cordoba, S., Guzman Navarro, F., Martinez, J., De Dios Lara, J., & Guzman, R. (2023). Towards concepts for climate and energy-oriented digital twins for buildings. *The 28th International ACM Conference on 3D Web Technology*, 1–9. https://doi.org/10.1145/3611314.3616066

Domaneschi, M., Mitoulis, S. A., Cucuzza, R., Villa, V., Di Bari, R., Siva, G., et al. (2023). Restoration of a landmark balanced cantilever bridge considering different resilience and sustainability strategies. In *Compdyn proceedings* (pp. 3339–3356, Vol. 2). National Technical University of Athens.

El Saddik, A. (2018). Digital Twins: The Convergence of Multimedia Technologies. *IEEE MultiMedia*, *25*(2), 87–92. https://doi.org/10.1109/MMUL.2018.023121167

Eramo, R., Bordeleau, F., Combemale, B., van Den Brand, M., Wimmer, M., & Wortmann, A. (2021). Conceptualizing digital twins. *IEEE Software*.

Erbay, O., Doğan, A., & Devecioğlu, E. (2024). Line Follower Robot with PID Control.

European Space Agency. (2021). Working towards a Digital Twin of Earth [Accessed: February 23h 2025]. https://www.esa.int/Applications/Observing_the_Earth/Working_towards_a_Digital_Twin_of_Earth

Fehlberg, E. (1969). Classical fifth-and seventh-order runge-kutta formulas with stepsize control. *Computing*, *4*, 93–106.

Feng, H., Gomes, C., Thule, C., Lausdahl, K., Sandberg, M., & Larsen, P. G. (2021). The Incubator Case Study for Digital Twin Engineering. *arXiv preprint arXiv:2102.10390*.

Feng, X., Wu, J., Wu, Y., Li, J., & Yang, W. (2023). Blockchain and digital twin empowered trustworthy self-healing for edge-AI enabled industrial Internet of Things. *Information Sciences*, *642*, 119169. https://doi.org/10.1016/j.ins.2023.119169

Fergus, P., Chalmers, C., Longmore, S., Wich, S., Warmenhove, C., Swart, J., Ngongwane, T., Burger, A., Ledgard, J., & Meijaard, E. (2023). Empowering Wildlife Guardians: An Equitable Digital Stewardship and Reward System for Biodiversity Conservation Using Deep Learning and 3/4G Camera Traps. *Remote Sensing*, *15*(11), 2730. https://doi.org/10.3390/rs15112730

Ferguson, S. (2020). Apollo 13: The First Digital Twin [Accessed: February 7th 2024]. https://blogs.sw.siemens.com/simcenter/apollo-13-the-first-digital-twin/

Ferko, E., Berardinelli, L., Bucaioni, A., Behnam, M., & Wimmer, M. (2024). Towards Interoperable Digital Twins: Integrating SysML into AAS with Higher-Order Transformations. *3rd International Workshop on Digital Twin Architecture (TwinArch) and Digital Twin Engineering (DTE)*.

Ferko, E., Bucaioni, A., Pelliccione, P., & Behnam, M. (2023). Standardisation in digital twin architectures in manufacturing. *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, 70–81.

Fernandes, L. (2014). The Abacus: A Brief History. https://www.ecb.torontomu.ca/~elf/abacus/history.html

Flammini, F. (2021). Digital twins as run-time predictive models for the resilience of cyber-physical systems: A conceptual framework. *Philosophical Transactions of the Royal Society A*, *379*(2207), 20200369.

Franceschini, R., Challenger, M., Cicchetti, A., Denil, J., & Vangheluwe, H. (2019). Challenges for automation in adaptive abstraction. *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 443–448.

Franceschini, R., Van Mierlo, S., & Vangheluwe, H. (2019). Towards Adaptive Abstraction in Agent Based Simulation. *Winter Simulation Conference (WSC)*, 2725–2736. https://doi.org/10.1109/WSC40007.2019.9004843

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G. S., & Friday, A. (2021). The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, *2*(9), 100340. https://doi.org/10.1016/j.patter.2021.100340

Gallego-García, S., Ren, D., Gallego-García, D., Pérez-García, S., & García-García, M. (2022). Dynamic Innovation Information System (DIIS) for a New Management Age. *Applied Sciences*, *12*(13). https://doi.org/10.3390/app12136592

Gelernter, D. (1993). *Mirror worlds: Or the day software puts the universe in a shoebox... how it will happen and what it will mean*. Oxford University Press.

Giese, H. (2006). *Software Engineering for Software-Intensive Systems: I Introduction* (tech. rep.). University of Paderborn.

Glaessgen, E., & Stargel, D. (2012). The digital twin paradigm for future NASA and US Air Force vehicles. *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, 1818.

Gockel, B., Tudor, A., Brandyberry, M., Penmetsa, R., & Tuegel, E. (2012). Challenges with Structural Life Forecasting Using Realistic Mission Profiles. *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. https://doi.org/10.2514/6.2012-1813

Goldstein, R., Breslav, S., & Khan, A. (2016). DesignDEVS: Reinforcing Theoretical Principles in a Practical and Lightweight Simulation Environment. *Proceedings of the 2016 Spring Simulation Multiconference*, 1–8.

Gomes, C., Denil, J., & Vangheluwe, H. (2020). Causal-Block Diagrams: A family of languages for causal modelling of cyber-physical systems. In *Foundations of multi-paradigm modelling for cyber-physical systems* (pp. 97–125). Springer International Publishing. https://doi.org/10.1007/978-3-030-43946-0_4

Gomes, C., Thule, C., Broman, D., Larsen, P. G., & Vangheluwe, H. (2018). Co-Simulation: A Survey. *ACM Comput. Surv.*, *51*(3).

Gomez-Escalonilla, J., Garijo, D., Valencia, O., & Rivero, I. (2020). Development of efficient high-fidelity solutions for virtual fatigue testing. *ICAF 2019–Structural Integrity in the Age of Additive Manufacturing: Proceedings of the 30th Symposium of* *the International Committee on Aeronautical Fatigue, June 2-7, 2019, Krakow, Poland*, 187–200.

Gong, L., Chen, B., Xu, W., Liu, C., Li, X., Zhao, Z., & Zhao, L. (2022). Motion similarity evaluation between human and a tri-co robot during real-time imitation with a trajectory dynamic time warping model. *Sensors*, *22*(5), 1968.

Gonzalez, M., Salgado, O., Croes, J., Pluymers, B., & Desmet, W. (2019). Model-Based State Estimation for the Diagnosis of Multiple Faults in Non-linear Electro-Mechanical Systems. In A. Fernandez Del Rincon, F. Viadero Rueda, F. Chaari, R. Zimroz, & M. Haddar (Eds.), *Advances in condition monitoring of machinery in non-stationary operations* (pp. 77–89). Springer International Publishing.

González, M., Salgado, O., Croes, J., Pluymers, B., & Desmet, W. (2020). Application of state estimation to the monitoring of multiple components in non-linear electro-mechanical systems. *Applied Acoustics*, *166*, 107371. https://doi.org/10.1016/j.apacoust.2020.107371

González, M., Salgado, O., Hernandez, X., Croes, J., Pluymers, B., & Desmet, W. (2019). Model-based condition monitoring of guiding rails in electro-mechanical systems. *Mechanical Systems and Signal Processing*, *120*, 630–641. https://doi.org/10.1016/j.ymssp.2018.10.044

Graessler, I., & Poehler, A. (2018). Intelligent control of an assembly station by integration of a digital twin for employees into the decentralized control system [4th International Conference on System-Integrated Intelligence: Intelligent, Flexible and Connected Systems in Products and Production]. *Procedia Manufacturing*, *24*, 185–189. https://doi.org/10.1016/j.promfg.2018.06.041

Grieves, M. (2008, September). Back to the Future: Product Lifecycle Management and the Virtualization of Product Information. In *Product realization* (pp. 1–13). Springer US. https://doi.org/10.1007/978-0-387-09482-3_3

Grieves, M. (2014). Digital twin: Manufacturing excellence through virtual factory replication. *White paper*, *1*(2014), 1–7.

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary perspectives on complex systems* (pp. 85–113). Springer.

Grinshpun, G., Cichon, T., Dipika, D., & Rossmann, J. (2016). From Virtual Testbeds to Real Lightweight Robots: Development and deployment of control algorithms for soft robots, with particular reference to industrial peg-in-hole insertion tasks. *Proceedings of ISR 2016: 47st International Symposium on Robotics*, 1–7. https://api.semanticscholar.org/CorpusID:57601202

Gross, D. C. (1999). Report from the fidelity implementation study group. *Fall Simulation Interoperability Workshop Papers*.

Hankel, M., & Rexroth, B. (2015). The reference architectural model industrie 4.0 (RAMI 4.0). *Zvei*, *2*(2), 4–9.

Hassan, A. A., & Aggarwal, G. (2023). Sustainable Manufacturing: Digital Twinning for a Mechanical Assembly Production Line. *2023 IEEE Smart World Congress (SWC)*, 758–763. https://doi.org/10.1109/SWC57546.2023.10449295

He, B., & Bai, K.-J. (2021). Digital twin-based sustainable intelligent manufacturing: A review. *Advances in Manufacturing*, *9*, 1–21.

Hearst Magazines. (1952). Atomic Reactor Makes Electricity. In *Popular mechanics* (p. 105).

Hendy, S. (2015). Professor Nicola Spaldin – cosmic strings. https://www.macdiarmid.ac.nz/news-and-events/news/interface-magazine-archive/professor-nicola-spaldin-cosmic-strings/

Heithoff, M., Hellwig, A., Michael, J., & Rumpe, B. (2023). Digital Twins for Sustainable Software Systems. *Int. Workshop on Green and Sustainable Software (GREENS 2023)*, 19–23.

Ho, A. K. (2023). Understanding the OPC Unified Architecture (OPC UA) Protocol. *Control Automation*.

Hodgins, J. K. (1998). Animation Human Motion. *Scientific American*, *278*, 46–51. http://www.sciam.com/1998/0398issue/0398hodgins.html

Hofmeister, M., Brownbridge, G., Hillman, M., Mosbach, S., Akroyd, J., Lee, K. F., & Kraft, M. (2024). Cross-domain flood risk assessment for smart cities using dynamic knowledge graphs. *Sustainable Cities and Society*, *101*, 105113.

Hong, N. P. C. (2021). *Reproducibility Badging And Definitions: A Recommended Practice Of The National Information Standards Organization* (tech. rep.). University of Edinburgh. National Information Standards Organization (NISO).

Höpfner, A., Poenicke., O., Blobner., C., & Winge., A. Use of a Virtual Twin for Dynamic Storage Space Monitoring in a Port Terminal. In: In *Proceedings of the 2nd international conference on innovative intelligent industrial production and logistics - in4pl*. INSTICC. SciTePress, 2021, 116–122. ISBN: 978-989-758-535-7. https://doi.org/10.5220/0010676800003062

Horizon Grand View Research. (2024). Global Digital Twin Market Size & Outlook, 2023-2030 [Accessed: July 8th 2024]. https://www.grandviewresearch.com/horizon/outlook/digital-twin-market-size/global

Howard, D. A., Ma, Z., Veje, C., Clausen, A., Aaslyng, J. M., & Jørgensen, B. N. (2021). Greenhouse industry 4.0 – digital twin technology for commercial greenhouses. *Energy Informatics*, *4*(2), 1–13.

Industry IoT Consortium. (2022). The Industrial Internet Reference Architecture [Accessed: April 5th 2024]. https://www.iiconsortium.org/IIRA/

Jain, P., Poon, J., Singh, J. P., Spanos, C., Sanders, S. R., & Panda, S. K. (2020). A Digital Twin Approach for Fault Diagnosis in Distributed Photovoltaic Systems. *IEEE Transactions on Power Electronics*, *35*(1), 940–956. https://doi.org/10.1109/TPEL.2019.2911594

Jayed, I., & Carlomagno, M. (2024). Sensory design for enhancing perceptual experience.: 80/20 olfactory training kit. a project for the recovery and rehabilitation of the sense of olfaction. *Convergences-Journal of Research and Arts Education*, *17*(33), 19–30.

Jenkins, R. (1936). Savery, Newcomen and the Early History of the Steam Engine. In *Links in the history of engineering and technology from Tudor times: the collected papers of Rhys Jenkins – Comprising articles in the professional and technical press mainly prior to 1920 and a catalogue of other published work* (pp. 48–72). Newcomen society at the University Press.

Jeong, D.-Y., Baek, M.-S., Lim, T.-B., Kim, Y.-W., Kim, S.-H., Lee, Y.-T., Jung, W.-S., & Lee, I.-B. (2022). Digital twin: Technology evolution stages and implementation layers with technology elements. *Ieee Access*, *10*, 52609–52620.

Jones, D., Snider, C., Nassehi, A., Yon, J., & Hicks, B. (2020). Characterising the Digital Twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, *29*, 36–52. https://doi.org/10.1016/j.cirpj.2020.02.002

Joordens, M., & Jamshidi, M. (2018). On the development of robot fish swarms in virtual reality with digital twins. *2018 13th Annual Conference on System of Systems Engineering (SoSE)*, 411–416.

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

Kang, K. C., & Lee, H. (2013). Variability modeling. In *Systems and software variability management* (pp. 25–42). Springer.

Kannoth, S., Hermann, J., Damm, M., Rübel, P., Rusin, D., Jacobi, M., Mittelsdorf, B., Kuhn, T., & Antonino, P. O. (2021). Enabling SMEs to Industry 4.0 Using the BaSyx Middleware: A Case Study. *Software Architecture*, 277–294. https://doi.org/10.1007/978-3-030-86044-8_19

Kapos, G.-D., Dalakas, V., Nikolaidou, M., & Anagnostopoulos, D. (2014). An Integrated Framework for Automated Simulation of SysML Models Using DEVS. *Simulation*, *90*(6), 717–744.

Karaduman, B., Tezel, B. T., & Challenger, M. (2023). Rational software agents with the BDI reasoning model for Cyber-Physical Systems. *Engineering Applications of Artificial Intelligence*, *123*, 106478. https://doi.org/https://doi.org/10.1016/j.engappai.2023.106478

Karakra, A., Fontanili, F., Lamine, E., Lamothe, J., & Taweel, A. (2018). Pervasive computing integrated discrete event simulation for a hospital digital twin. *2018 IEEE/ACS 15th international conference on computer systems and Applications (AICCSA)*, 1–6.

Kassiakoff, A., Sweet, W. N., Seymour, S. J., & Biemer, S. M. (2010). *Systems Engineering Principles And Practice* (A. P. Sage, Ed.; second). John Wiley & Sons, Inc.

Kennedy, A. (2016). *Edison: Inventing the Modern World*. CreateSpace Independent Publishing Platform.

Kibira, D., Shao, G., & Weiss, B. A. (2021). Buiding a digital twin for robot workcell prognostics and health management. *2021 Winter Simulation Conference (WSC)*, 1–12.

Knibbe, W. J., Afman, L., Boersma, S., Bogaardt, M.-J., Evers, J., van Evert, F., van der Heide, J., Hoving, I., van Mourik, S., de Ridder, D., & de Wit, A. (2022). Digital twins in the green life sciences. *NJAS: Impact in Agricultural and Life Sciences*, *94*(1), 249–279. https://doi.org/10.1080/27685241.2022.2150571

Kosicka, E., Kozłowski, E., & Mazurkiewicz, D. (2017). Intelligent Systems of Forecasting the Failure of Machinery Park and Supporting Fulfilment of Orders of Spare Parts. *Proceedings of Intelligent Systems in Production Engineering and Maintenance – ISPEM 2017*, 54–63. https://api.semanticscholar.org/CorpusID:115479060

Kritzinger, W., Karner, M., Traar, G., Henjes, J., & Sihn, W. (2018). Digital Twin in Manufacturing: A Categorical Literature Review and Classification. *IFAC-PapersOnLine*, *51*(11), 1016–1022.

Kumar, M., & Hote, Y. V. (2020). Robust pidd2 controller design for perturbed load frequency control of an interconnected time-delayed power systems. *IEEE Transactions on Control Systems Technology*, *29*(6), 2662–2669.

Landahl, J., Panarotto, M., Johannesson, H., Isaksson, O., & Lööf, J. (2018). Towards adopting digital twins to support design reuse during platform concept development. *Proceedings of NordDesign 2018*. https://api.semanticscholar.org/CorpusID:58180294

Lee, K., & Kang, K. C. (2010). Usage context as key driver for feature selection. *Software Product Lines: Going Beyond: 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings 14*, 32–46.

Leroy, D., Bousse, E., Megna, A., Combemale, B., & Wimmer, M. (2018). Trace comprehension operators for executable dsls. *Modelling Foundations and Applications: 14th*

European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings 14, 293–310.

Lim, K. Y. H., Zheng, P., & Chen, C.-H. (2020). A state-of-the-art survey of Digital Twin: techniques, engineering product lifecycle management and business innovation perspectives. *Journal of Intelligent Manufacturing*, *31*, 1313–1337.

Llopis, J., Criado, J., Iribarne, L., Muñoz, P., Troya, J., & Vallecillo, A. (2023). Modeling and Synchronizing Digital Twin Environments. *2023 Annual Modeling and Simulation Conference (ANNSIM)*, 245–257. https://doi.ieeecomputersociety.org/

Long-Fox, J. M., Landsman, Z. A., Easter, P. B., Millwater, C. A., & Britt, D. T. (2023). Geomechanical properties of lunar regolith simulants lhs-1 and lms-1. *Advances in Space Research*, *71*(12), 5400–5412.

Loverdos, D., & Sarhosis, V. (2023). Geometrical digital twins of masonry structures for documentation and structural assessment using machine learning. *Engineering Structures*, *275*, 115256. https://doi.org/10.1016/j.engstruct.2022.115256

Lu, J., Zheng, X., Gharaei, A., Kalaboukas, K., & Kiritsis, D. (2020). Cognitive twins for supporting decision-makings of internet of things systems. *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing*, 105–115.

Lucas, G. W. (2000). A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators [Accessed: 24th of April 2024]. http://rossum.sourceforge.net/papers/DiffSteer/

Lúcio, L., Mustafiz, S., Denil, J., Meyers, B., & Vangheluwe, H. (2012). The formalism transformation graph as a guide to model driven engineering. *School Comput. Sci., McGill Univ., Tech. Rep. SOCS-TR2012*, *1*.

Lugaresi, G., Frigerio, N., & Matta, A. (2020). A new learning factory experience exploiting LEGO for teaching manufacturing systems integration. *Procedia Manufacturing*, *45*, 271–276.

Lugaresi, G., & Matta, A. (2018). Real-time simulation in manufacturing systems: Challenges and research directions. *2018 Winter Simulation Conference (WSC)*, 3319–3330.

Lutters, E. (2018). Pilot production environments driven by digital twins. *South African Journal of Industrial Engineering*, *29*(3), 40–53. https://api.semanticscholar.org/CorpusID:115677639

Lutters, E., & Damgrave, R. (2019). The development of pilot production environments based on digital twins and virtual dashboards. *Procedia CIRP*, *84*, 94–99.

Lynch, K. M., & Park, F. C. (2017). *Modern Robotics: Mechanics, Planning and Control*. Cambridge University Press.

Ma, J., Wang, G., Lu, J., Vangheluwe, H., Kiritsis, D., & Yan, Y. (2022). Systematic Literature Review Of MBSE Tool-Chains. *Applied Sciences*, *12*(7), 3431/1–21. https://doi.org/10.3390/app12073431

Madni, A. M., Madni, C. C., & Lucero, S. D. (2019). Leveraging digital twin technology in model-based systems engineering. *Systems*, *7*(1), 7.

Mahmoud, M. M. M., Darwish, R., & Bassiuny, A. M. (2023). Development of a Smart Aquaponic System Based on IoT. *2023 23rd International Conference on Control, Automation and Systems (ICCAS)*, 1592–1597.

Maleki, M., Woodbury, R., Goldstein, R., Breslav, S., & Khan, A. (2015). Designing devs visual interfaces for end-user programmers. *Simulation*, *91*(8), 715–734.

Mandolla, C., Petruzzelli, A. M., Percoco, G., & Urbinati, A. (2019). Building a digital twin for additive manufacturing through the exploitation of blockchain: A case analysis of the aircraft industry. *Computers in Industry*, *109*, 134–152.

Marah, H., & Challenger, M. (2023). MADTwin: A Framework for Multi-agent Digital Twin Development: Smart Warehouse Case Study. *Annals of Mathematics and Artificial Intelligence*. https://doi.org/https://doi.org/10.1007/s10472-023-09872-z

Marah, H., & Challenger, M. (2025). (Re-)Engineering Digital Twins Towards Federation: Vision and Roadmap. In T. Margaria & B. Steffen (Eds.), *Leveraging applications of formal methods, verification and validation. software engineering methodologies* (pp. 60–81). Springer Nature Switzerland.

Marah, H., Paredis, R., Challenger, M., & Vangheluwe, H. (2023). A Multi-Robot Warehouse System: An Exemplar. *Proceedings of the 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 530–538.

Marion, P. (2021). How Boston Dynamics makes Atlas run, flip & vault [Accessed: September 18th 2024]. https://www.therobotreport.com/how-boston-dynamics-makes-atlas-run-flip-vault/

Market Research Future. (2017). Global Digital Twin Market Is Estimated to Grow at a CAGR of 37% from 2017 to 2023 [Accessed: July 8th 2024]. https://www.marketresearchfuture.com/press-release/digital-twin-market

Markwirth, T., Jancke, R., & Sohrmann, C. (2021). Dynamic fault injection into digital twins of safety-critical systems. *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 446–450.

Maruyama, Y., Kato, S., & Azumi, T. (2016). Exploring the performance of ROS2. *Proceedings of the 13th international conference on embedded software*, 1–10.

Maxwell, F. D., & Corn, B. C. (1978). *The determination of measures of software reliability* (tech. rep.). NASA.

Meyers, B., Deshayes, R., Lucio, L., Syriani, E., Vangheluwe, H., & Wimmer, M. (2014). ProMoBox: a framework for generating domain-specific property languages. *Software Language Engineering: 7th International Conference, SLE 2014, Västerås, Sweden, September 15-16, 2014. Proceedings 7*, 1–20.

Meyers, B., & Vangheluwe, H. (2011). A Framework For Evolution Of Modelling Languages. *Science of Computer Programming*, *76*(12), 1223–1246.

Micaelli, A., & Samson, C. (1993). *Trajectory tracking for unicycle-type and two-steering-wheels mobile robots* [Doctoral dissertation, INRIA].

Michael, J. (2023, October). Unlocking Potential: Rocking the Sustainable Future with Digital Twins [Keynote]. https://judithmichael.github.io/assets/pdfs/23.10.01.ModDiT.Keynote.JudithMichael.pdf

Michael, J., & Wortmann, A. (2021). Towards Development Platforms for Digital Twins: A Model-Driven Low-Code Approach. *IFIP International Conference on Advances in Production Management Systems*, 333–341.

Minerva, R., Lee, G. M., & Crespi, N. (2020). Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE*, *108*(10), 1785–1824. https://doi.org/10.1109/JPROC.2020.2998530

Mitchell, M. (2009). *Complexity: A guided tour*. Oxford University Press.

Mittal, R., Eslampanah, R., Lima, L., Vangheluwe, H., & Blouin, D. (2023). Towards an Ontological Framework for Validity Frames. *2023 ACM/IEEE International Confer-*

ence on Model Driven Engineering Languages and Systems Companion (MODELS-C), 801–805.

Modoni, G. E., Caldarola, E. G., Sacco, M., & Terkaj, W. (2019). Synchronizing physical and digital factory: Benefits and technical challenges [12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy]. *Procedia CIRP*, *79*, 472–477. https://doi.org/10.1016/j.procir.2019.02.125

Mohamed, N., Al-Jaroodi, J., & Lazarova-Molnar, S. (2019). Leveraging the Capabilities of Industry 4.0 for Improving Energy Efficiency in Smart Factories. *IEEE Access*, *7*, 18008–18020. https://doi.org/10.1109/ACCESS.2019.2897045

Monteiro, J., Barata, J., Veloso, M., Veloso, L., & Nunes, J. (2023). A scalable digital twin for vertical farming. *Journal of Ambient Intelligence and Humanized Computing*, *14*(10), 13981–13996.

Mosterman, P. J., & Vangheluwe, H. (2004). Computer automated multi-paradigm modeling: An introduction. *Simulation*, *80*(9), 433–450. https://doi.org/10.1177/0037549704050532

Mouflih, C., Gaha, R., Durupt, A., Bosch-Mauchand, M., Martinsen, K., & Eynard, B. (2023). Decision support framework using knowledge based digital twin for sustainable product development and end of life. *Proceedings of the Design Society*, *3*, 1157–1166.

Moya, B., Alfaro, I., Gonzalez, D., Chinesta, F., & Cueto, E. (2020). Physically sound, self-learning digital twins for sloshing fluids. *PLoS One*, *15*(6), e0234569.

Mukherjee, T., & DebRoy, T. (2019). A digital twin for rapid qualification of 3D printed metallic components. *Applied Materials Today*, *14*, 59–65.

Müller, T., Lindemann, B., Jung, T., Jazdi, N., & Weyrich, M. (2021). Enhancing an Intelligent Digital Twin with a Self-organized Reconfiguration Management based on Adaptive Process Models. *CoRR*, *abs/2107.03324*, 786–791. https://arxiv.org/abs/2107.03324

Muñoz, P., Troya, J., & Vallecillo, A. (2021). Using UML and OCL Models to Realize High-Level Digital Twins. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 212–220.

Muñoz, P., Troya, J., & Vallecillo, A. (2024). Towards measuring digital twins fidelity at runtime. *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 507–512.

Muñoz, P., Wimmer, M., Troya, J., & Vallecillo, A. (2022). Using trace alignments for measuring the similarity between a physical and its digital twin. *Proceedings of the 25th international conference on model driven engineering languages and systems: Companion proceedings*, 503–510.

Münzinger, M., Prechtel, N., & Behnisch, M. (2022). Mapping the urban forest in detail: From LiDAR point clouds to 3D tree models. *Urban Forestry &; Urban Greening*, *74*, 127637. https://doi.org/10.1016/j.ufug.2022.127637

Mustafiz, S., Denil, J., Lúcio, L., & Vangheluwe, H. (2012). The FTG+PM Framework For Multi-Paradigm Modelling: An Automotive Case Study. *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, 13–18.

Muzy, A., & Nutaro, J. J. (2005). Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators. *1st Open International Conference on Modeling and Simulation*, 273–279.

Nativi, S., Mazzetti, P., & Craglia, M. (2021). Digital ecosystems for developing digital twins of the earth: The destination earth case. *Remote Sensing*, *13*(11), 2119.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, *48*(3), 443–453.

Negrin, D. A. M., Cleophas, L., & Van Den Brand, M. (2021). Using Ptolemy II as a Framework for Virtual Entity Integration and Orchestration in Digital Twins. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 233–236.

Nezhad, S. N., Paredis, R., Van Acker, B., & Vangheluwe, H. (n.d.). Combining experiments and a Historian for building Twinning systems, applied to a TurtleBot use-case.

Nibert, D. (2011). Origins and Consequences of the Animal Industrial Complex. In S. Best, R. Kahn, A. J. Nocella II, & P. McLaren (Eds.), *The Global Industrial Complex: Systems of Domination* (pp. 197–209). Lexington Books.

Niloofar, P., Francis, D. P., Lazarova-Molnar, S., Vulpe, A., Vochin, M.-C., Suciu, G., Balanescu, M., Anestis, V., & Bartzanas, T. (2021). Data-driven decision support in livestock farming for improved animal health, welfare and greenhouse gas emissions: Overview and challenges. *Computers and Electronics in Agriculture*, *190*, 106406.

Nutaro, J. J. (2015). Adevs [Accessed 10th May.].

Oakes, B., Gomes, C., Larsen, P., Denil, J., Deantoni, J., Cambeiro, J., & Fitzgerald, J. (2023). Examining Model Qualities and Their Impact on Digital Twins. *2023 Annual Modeling and Simulation Conference (ANNSIM)*, 220–232.

Oakes, B. J., Parsai, A., Meyers, B., David, I., Mierlo, S. V., Demeyer, S., Denil, J., Meulenaere, P. D., & Vangheluwe, H. (2021). A digital twin description framework and its mapping to asset administration shell. *Companion Proceedings of the International Conference on Model-Driven Engineering and Software Development*, 1–24.

Object Management Group. (2017). Omg® unified modeling language® (omg uml®) 2.5.1. https://www.omg.org/spec/UML/2.5.1/PDF

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., & Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, *14*(3), 54–62.

Our World in Data. (2023). Economic Growth - all charts [Accessed: March 27th 2024]. https://ourworldindata.org/economic-growth#all-charts

Pacejka, H. B. (1966). *The wheel shimmy phenomenon: A theoretical and experimental investigation with particular reference to the non-linear problem* [Doctoral dissertation, Delft University of Technology].

Padovano, A., Longo, F., Nicoletti, L., & Mirabelli, G. (2018). A Digital Twin based Service Oriented Application for a 4.0 Knowledge Navigation in the Smart Factory [16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018]. *IFAC-PapersOnLine*, *51*(11), 631–636. https://doi.org/10.1016/j.ifacol.2018.08.389

Paredis, R., Denil, J., & Vangheluwe, H. (2021). Specifying and Executing the Combination of Timed Finite State Automata and Causal-Block Diagrams by Mapping onto DEVS. *Proceedings of the 2021 Winter Simulation Conference (WSC)*.

Paredis, R., Exelmans, J., & Vangheluwe, H. (2022). Multi-Paradigm Modelling for Model-Based Systems Engineering: Extending the FTG+PM. *Proceedings of the 2022 Annual Modeling and Simulation Conference (ANNSIM)*, 461–474.

Paredis, R., Gomes, C., & Vangheluwe, H. (2021). Towards a Family of Digital Model / Shadow / Twin Workflows and Architectures. *Proceedings of the 2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2021)*, 174–182.

Paredis, R., Gomes, C., & Vangheluwe, H. (2023). A Family of Digital T Workflows and Architectures: Exploring Two Cases. In A. Smirnov, H. Panetto, & K. Madani (Eds.), *Innovative intelligent industrial production and logistics* (pp. 93–109). Springer Nature Switzerland.

Paredis, R., Van Mierlo, S., & Vangheluwe, H. (2020). Translating Process Interaction World View Models to DEVS: GPSS to (Python(P))DEVS. In K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, & R. Thiesing (Eds.), *Proceedings of the 2020 winter simulation conference* (pp. 2221–2232). Institute of Electrical; Electronics Engineers, Inc.

Paredis, R., & Vangheluwe, H. (2021). Exploring a Digital Shadow Design Workflow by Means of a Line Following Robot Use-Case. *Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM)*, 1–12.

Paredis, R., & Vangheluwe, H. (2022). Towards a Digital Z Framework Based on a Family of Architectures and a Virtual Knowledge Graph. *Companion Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS-C)*, 491–496.

Paredis, R., & Vangheluwe, H. (2024a). Exploring Twinning Variability [Poster]. *Proceedings of the 4th Conference on Machines, Vehicles and Production Technology (CMVPT)*.

Paredis, R., & Vangheluwe, H. (2024b). Modelling and Simulation-Based Evaluation of Twinning Architectures and Their Deployment. *Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, 170–182. https://doi.org/10.5220/0012865300003758

Paredis, R., Vangheluwe, H., & Albertins, P. A. R. (2024). *COOCK project Smart Port 2025 D3.1: "To Twin Or Not To Twin"* (tech. rep.) (ArXiv preprint). University of Antwerp.

Parezys, J., Paredis, R., & Vangheluwe, H. (2023). CLAVS/ODVS: Combining Class/Object Diagrams and DEVS. *Proceedings of the 2023 Winter Simulation Conference (WSC)*, 2591–2602.

Plesser, H. E. (2018). Reproducibility Vs. Replicability: A Brief History Of A Confused Terminology. *Frontiers in neuroinformatics*, *11*, 76.

Popa, C. L., Cotet, C. E., Popescu, D., Solea, M. F., Şaşcîm, S. G., & Dobrescu, T. (2018). Material flow design and simulation for a glass panel recycling installation. *Waste Management & Research*, *36*(7), 653–660.

Poppe, A., Farkas, G., Gaál, L., Hantos, G., Hegedüs, J., & Rencz, M. (2019). Multi-domain modelling of LEDs for supporting virtual prototyping of luminaires. *Energies*, *12*(10), 1909.

Qamar, A., & Paredis, C. (2012). Dependency Modeling And Model Management In Mechatronic Design. *Proceedings of the ASME Design Engineering Technical Conference*, *2*, 1–12. https://doi.org/10.1115/DETC2012-70272

Qin, Y., Wu, X., & Luo, J. (2022). Data-Model Combined Driven Digital Twin of Life-Cycle Rolling Bearing. *IEEE Transactions on Industrial Informatics*, *18*(3), 1530–1540. https://doi.org/10.1109/TII.2021.3089340

Quadrini, W., Cimino, C., Abdel-Aty, T. A., Fumagalli, L., & Rovere, D. (2023). Asset Administration Shell as an interoperable enabler of Industry 4.0 software architectures: a case study. *Procedia Computer Science*, *217*, 1794–1802.

Qudrat-Ullah, H., & Seong, B. S. (2010). How to do structural validity of a system dynamics type simulation model: The case of an energy policy model [Greater China Energy: Special Section with regular papers]. *Energy Policy*, *38*(5), 2216–2224. https://doi.org/10.1016/j.enpol.2009.12.009

R., R., Prahalad, H., Kumar, H. S., & Hoslok, A. (2023). Exploring Digital Twins for Plant Growth Monitoring. *2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, *11*, 1–6. https://doi.org/10.1109/csitss60515.2023.10334087

Raja Santhi, A., & Muthuswamy, P. (2023). Industry 5.0 or industry 4.0S? Introduction to industry 4.0 and a peek into the prospective industry 5.0 technologies. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, *17*(2), 947–979. https://doi.org/10.1007/s12008-023-01217-8

Rajamani, R. (2011). *Vehicle Dynamics and Control*. Springer Science & Business Media.

Rakove, J. N. (1996). Fidelity through History (Or Do It). *Fordham L. Rev.*, *65*, 1587.

Rambow-Hoeschele, K., Nagl, A., Harrison, D. K., Wood, B. M., Bozem, K., Braun, K., & Hoch, P. (2018). Creation of a Digital Business Model Builder. *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 1–7. https://api.semanticscholar.org/CorpusID:52018061

Redelinghuys, A. J. H., Basson, A. H., & Kruger, K. (2019). A six-layer architecture for the digital twin: A manufacturing case study implementation. *Journal of Intelligent Manufacturing*, *31*(6), 1383–1402. https://doi.org/10.1007/s10845-019-01516-6

Reese, V. L., & Dunn, R. (2007). Learning-Style Preferences of a Diverse Freshmen Population in a Large, Private, Metropolitan University by Gender and GPA. *Journal of College Student Retention: Research, Theory & Practice*, *9*(1), 95–112. https://doi.org/10.2190/N836-888L-2311-2374

Rivera, L. F., Jiménez, M., Villegas, N. M., Tamura, G., & Müller, H. A. (2022). The Forging of Autonomic and Cooperating Digital Twins. *IEEE Internet Computing*, *26*(5), 41–49. https://doi.org/10.1109/MIC.2021.3051902

Rosen, R., Fischer, J., & Boschert, S. (2019). Next generation digital twin: An ecosystem for mechatronic systems? *IFAC-papersonline*, *52*(15), 265–270.

Routledge, R. (1881). Physics of the Nineteenth Century – Electricity. In *A Popular History of Science* (pp. 548–594). G. Routledge and sons.

Rožanec, J. M., Lu, J., Rupnik, J., Škrjanc, M., Mladenić, D., Fortuna, B., Zheng, X., & Kiritsis, D. (2021). Actionable cognitive twins for decision making in manufacturing. *International Journal of Production Research*, 1–27.

Rumpe, B. (2021, October). Modelling for and of Digital Twins [Keynote].

Russell, N., Van Der Aalst, W. M. P., & Ter Hofstede, A. H. M. (2016). *Workflow Patterns: The Definitive Guide*. MIT Press.

Santillán Martínez, G. (2019). *Simulation-based Digital Twins of Industrial Process Plants: A Semi-Automatic Implementation Approach* [Doctoral dissertation, Aalto University]. Aalto University.

Sanz, V., Urquia, A., & Dormido, S. (2007). DEVS Specification and Implementation of SIMAN Blocks Using Modelica Language. *Proceedings of the 2007 Winter Simulation Conference*, 2374–2374.

Schäfer, S., Schöttke, D., Kämpfe, T., Ralinovski, K., Tauber, B., & Lehmann, J. (2021). Design and Deployment of Digital Twins for Programmable Logic Controllers in Existing Plants. *2nd International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL)*.

SEBoK Editorial Board. (2023). The Guide to the Systems Engineering Body of Knowledge (SEBoK) (N. Hutchison, Ed.) [Accessed: 11th of April 2024]. www.sebokwiki.org

Segers, B. (2024). "Zelf atomen en moleculen bouwen zonder risico's": leerlingen PITO Stabroek leren fysica en chemie met herwerkte Minecraft-game [Accessed: October 4th 2024]. *VRT Nieuws*. https://www.vrt.be/vrtnws/nl/2024/10/04/zelf-atomen-en-moleculen-bouwen-zonder-risico-s-leerlingen-pi/

Shaikh, R., & Vangheluwe, H. (2011). Transforming uml2.0 class diagrams and statecharts to atomic devs. *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation*, 205–212.

Shannon, C. E. (1938). *A symbolic analysis of relay and switching circuits* [Master's thesis, Massachusetts Institute of Technology (MIT), Dept. of Electrical Engineering].

Sharma, A., Kosasih, E., Zhang, J., Brintrup, A., & Calinescu, A. (2022). Digital Twins: State of the art theory and practice, challenges, and open research questions. *Journal of Industrial Information Integration*, *30*, 100383. https://doi.org/10.1016/j.jii.2022.100383

Sharma, P., Knezevic, D. J., Huynh, P., & Malinowski, G. (2018). Rb-fea based digital twin for structural integrity assessment of offshore structures. *Proceedings of the Offshore Technology Conference*, D031S037R003. https://api.semanticscholar.org/CorpusID:116856513

Shi, G., Gao, D., Song, X., Chai, J., Yang, M., Xie, X., Li, L., & Li, X. (2021). A new communication paradigm: From bit accuracy to semantic fidelity. *arXiv preprint arXiv:2101.12649*.

Shrivastava, C., Berry, T. M., Crone, J., & Defraeye, T. (2020). Digital twins to map the key quality attributes in fresh-produce supply chains. https://doi.org/10.18462/IIR.ICCC.2020.292336

Silber, K., Sagmeister, P., Schiller, C., Williams, J. D., Hone, C. A., & Kappe, C. O. (2023). Accelerating reaction modeling using dynamic flow experiments, part 2: Development of a digital twin. *Reaction Chemistry & Engineering*, *8*(11), 2849–2855.

Singh, M., Fuenmayor, E., Hinchy, E. P., Qiao, Y., Murray, N., & Devine, D. (2021). Digital twin: Origin to future. *Applied System Innovation*, *4*(2), 36.

Singh, R. (1996). International Standard ISO/IEC 12207 software life cycle processes. *Software Process Improvement and Practice*, *2*(1), 35–50.

Sloane, A. (2018). Fast line-following robots [Accessed: 24th of April 2024]. https://www.a1k0n.net/2018/11/13/fast-line-following.html

Smith, D. E. (1925). Mechnaical Aids to Calculation. In *History of Mathematics Vol 2* (pp. 156–178). Ginn And Company.

Smith, T. F., Waterman, M. S., et al. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, *147*(1), 195–197.

Stachowaik, H. (1973). *Allgemeine Modelltheorie*. Springer-Verlag.

Stark, J. (2022). Product lifecycle management (plm). In *Product lifecycle management (volume 1): 21st century paradigm for product realisation* (pp. 1–32). Springer International Publishing. https://doi.org/10.1007/978-3-030-98578-3_1

Steindl, G., Stagl, M., Kasper, L., Kastner, W., & Hofmann, R. (2020). Generic digital twin architecture for industrial energy systems. *Applied Sciences*, *10*(24), 8903.

Steinmetz, C., Rettberg, A., Ribeiro, F. o. G. C., Schroeder, G., & Pereira, C. E. (2018). Internet of Things Ontology for Digital Twin in Cyber Physical Systems. *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, 154–159. https://doi.org/10.1109/SBESC.2018.00030

Syriani, E., & Vangheluwe, H. (2013). A modular timed graph transformation language for simulation-based design. *Software & Systems Modeling*, *12*, 387–414.

Talkhestani, B. A., Jazdi, N., Schloegl, W., & Weyrich, M. (2018). Consistency check to synchronize the digital twin of manufacturing automation based on anchor points [51st CIRP Conference on Manufacturing Systems]. *Procedia CIRP*, *72*, 159–164. https://doi.org/10.1016/j.procir.2018.03.166

Tao, F., Xiao, B., Qi, Q., Cheng, J., & Ji, P. (2022). Digital twin modeling. *Journal of Manufacturing Systems*, *64*, 372–389.

Tao, F., & Zhang, M. (2017). Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing. *IEEE Access*, *5*, 20418–20427.

Tekinerdogan, B., & Verdouw, C. (2020). Systems architecture design pattern catalog for developing digital twins. *Sensors*, *20*(18), 5103.

Tero, A., Takagi, S., Saigusa, T., Ito, K., Bebber, D. P., Fricker, M. D., Yumiki, K., Kobayashi, R., & Nakagaki, T. (2010). Rules for Biologically Inspired Adaptive Network Design. *Science*, *327*(5964), 439–442. https://doi.org/10.1126/science.1177894

The Editors of Encyclopaedia Britannica. (2024a, February). Edmund Cartwright [Accessed: March 29th 2024]. https://www.britannica.com/biography/Edmund-Cartwright

The Editors of Encyclopaedia Britannica. (2024b, February). Industrial Revolution [Accessed: March 25th 2024]. https://www.britannica.com/event/Industrial-Revolution

The Refinery. (2016). The Dangers, and Benefits, of Software Fidelity [Accessed: February 7th 2024]. https://the-refinery.io/blog/the-dangers-and-benefits-of-software-fidelity

The World Bank. (2023). World Development Indicators.

Topuzoglu, T., Köktürk, G., Altun, D. A., Sendemir, A., Cakır, O. A., Tokuc, A., & Ozkaban, F. A. (2019). Finding the Shortest Paths in Izmir Map by Using Slime Molds Images. *2019 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*, 202–206.

Torres, E. O. C., Konduri, S., & Pagilla, P. R. (2014). Study of wheel slip and traction forces in differential drive robots and slip avoidance control strategy. *2014 American Control Conference*, 3231–3236.

Traoré, M. K. (2023). High-Level Architecture for Interoperable Digital Twins. *Preprints*. https://doi.org/10.20944/preprints202310.0659.v1

Turing, A. M., et al. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *58*(345-363), 5.

Um, J., Weyer, S., & Quint, F. (2017). Plug-and-Simulate within Modular Assembly Line enabled by Digital Twins and the use of AutomationML. *IFAC-PapersOnLine*, *50*(1), 15904–15909.

Utzig, S., Kaps, R., Azeem, S. M., & Gerndt, A. (2019). Augmented Reality for Remote Collaboration in Aircraft Maintenance Tasks. *2019 IEEE Aerospace Conference*, 1–10. https://doi.org/10.1109/AERO.2019.8742228

Van Acker, B., Meulenaere, P. D., Vangheluwe, H., & Denil, J. (2024). Validity frame-enabled model-based engineering processes. *Simulation*, *100*(2), 185–226. https://doi.org/10.1177/00375497231205035

Van der Valk, H., Haße, H., Möller, F., Arbter, M., Henning, J.-L., & Otto, B. (2020). A taxonomy of digital twins. *AMCIS*, 1–10.

Van Tendeloo, Y., Paredis, R., & Vangheluwe, H. (2020). An Introduction To Modular Modeling And Simulation With PythonPDEVS And The Building-Block Library

PythonPDEVS-BBL. *Proceedings of the 2020 Winter Simulation Conference (WSC)*, 1152–1166.

Van Tendeloo, Y., Paredis, R., & Vangheluwe, H. (2023). An Introduction to Discrete-Event Modeling and Simulation with DEVS. *Proceedings of the 2023 Winter Simulation Conference (WSC)*, 1531–1545.

Van Tendeloo, Y., & Vangheluwe, H. (2015). PythonPDEVS: a Distributed Parallel DEVS simulator. *Proceedings of the 2015 Spring Simulation Multiconference*, 844–851.

Vangheluwe, H. (2000). DEVS as a Common Denominator for Multi-Formalism Hybrid Systems Modelling. *IEEE International Symposium on Computer-Aided Control System Design*, 129–134.

Verdouw, C. N., & Kruize, J. W. (2017). Digital twins in farm management: illustrations from the FIWARE accelerators SmartAgriFood and Fractals. *Proceedings of the 7th Asian-Australasian Conference on Precision Agriculture Digital, Hamilton, New Zealand*, 16–18.

Verriet, J. (2019). From Virtual Prototype to Digital Twin [Accessed: February 7th 2024]. https://a.storyblok.com/f/74249/x/063ff220fa/s1-verriet-from-virtual-prototype-to-digital-twin.pdf

Wagle, M., Agnihotri, A., Bhangale, P., Patare, A., & Murali, M. (2023). Estimation of State of Charge in Electric Vehicle using the Battery Digital Twin. *2023 3rd International Conference on Intelligent Technologies (CONIT)*, 1–7.

Walravens, G., Muctadir, H. M., & Cleophas, L. (2022). Virtual Soccer Champions: A Case Study on Artifact Reuse in Soccer Robot Digital Twin Construction. *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 463–467. https://doi.org/10.1145/3550356.3561586

Wanasinghe, T. R., Wroblewski, L., Petersen, B. K., Gosine, R. G., James, L. A., De Silva, O., Mann, G. K. I., & Warrian, P. J. (2020). Digital twin for the oil and gas industry: Overview, research trends, opportunities, and challenges. *IEEE access*, *8*, 104175–104197.

Wang, H., Zhou, M., & Liu, B. (2018). Tolerance allocation with simulation-based digital twin for CFRP-metal countersunk bolt joint. *ASME International Mechanical Engineering Congress and Exposition*, *52019*, V002T02A108. https://doi.org/10.1115/IMECE2018-86645

Wärmefjord, K., Söderberg, R., Lindkvist, L., Lindau, B., & Carlson, J. S. (2017). Inspection data to support a digital twin for geometry assurance. *ASME international mechanical engineering congress and exposition*, *58356*, V002T02A101.

Weyns, D., Malek, S., & Andersson, J. (2010). Forms: A formal reference model for self-adaptation. *Proceedings of the 7th international conference on Autonomic computing*, 205–214.

Wisher, R. A. (2001). *The virtual sand table: Intelligent tutoring for field artillery training*. US Army Research Institute for the Behavioral; Social Sciences.

Worden, K., Cross, E. J., Barthorpe, R. J., Wagg, D. J., & Gardner, P. (2020). On digital twins, mirrors, and virtualizations: Frameworks for model verification and validation. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, *6*(3), 030902.

Wu, Y., & Ge, D. (2019). Key technologies of warehousing robot for intelligent logistics. *The First International Symposium on Management and Social Sciences (ISMSS 2019)*, 79–82.

Yacob, F., Semere, D., & Nordgren, E. (2019). Anomaly detection in Skin Model Shapes using machine learning classifiers. *The International Journal of Advanced Manufacturing Technology*, *105*(9), 3677–3689.

Yan, J., & Li, B. (2020). Research hotspots and tendency of intelligent manufacturing. *Chinese Science Bulletin*, *65*(8), 684–694. https://doi.org/10.1360/n972019-00125

Yao, X., Ma, N., Zhang, J., Wang, K., Yang, E., & Faccio, M. (2022). Enhancing wisdom manufacturing as industrial metaverse for industry and society 5.0. *Journal of Intelligent Manufacturing*, *35*(1), 235–255. https://doi.org/10.1007/s10845-022-02027-7

Ye, X., Hong, S. H., Song, W. S., Kim, Y. C., & Zhang, X. (2021). An industry 4.0 asset administration shell-enabled digital solution for robot-based manufacturing systems. *Ieee Access*, *9*, 154448–154459.

Yuan, C., Yeung, L., Zhang, X., & Qiu, S. (2022). Smart-Color: Color-Interactive Device Design Based on Programmable Physical Color-Changing Materials and Motion Capture Technique. *HCI International 2022 – Late Breaking Papers: Ergonomics and Product Design*, 542–551. https://doi.org/10.1007/978-3-031-21704-3_38

Zaki, N., Lazarova-Molnar, S., El-Hajj, W., & Campbell, P. (2009). Protein-protein interaction based on pairwise similarity. *BMC Bioinformatics*, *10*(1). https://doi.org/10.1186/1471-2105-10-150

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of Modeling and Simulation* (3rd). Academic Press.

Zhang, C., Wang, Z., Zhou, G., Chang, F., Ma, D., Jing, Y., Cheng, W., Ding, K., & Zhao, D. (2023). Towards new-generation human-centric smart manufacturing in Industry 5.0: A systematic review. *Advanced Engineering Informatics*, *57*, 102121. https://doi.org/10.1016/j.aei.2023.102121

Zhang, H., Wei, Y., & Li, Z. (2023). Research on the establishment and simulation method of testability model based on digital twin. In X. Yuan & G. Wu (Eds.), *Third international conference on mechanical, electronics, and electrical and automation control (metms 2023)* (127223A, Vol. 12722). SPIE. https://doi.org/10.1117/12.2679714

Zhang, H., Liu, Q., Chen, X., Zhang, D., & Leng, J. (2017). A digital twin-based approach for designing and multi-objective optimization of hollow glass production line. *Ieee Access*, *5*, 26901–26911.

Zhang, Z., Wen, F., Sun, Z., Guo, X., He, T., & Lee, C. (2022). Artificial Intelligence-Enabled Sensing Technologies in the 5G/Internet of Things Era: From Virtual Reality/Augmented Reality to the Digital Twin. *Advanced Intelligent Systems*, *4*(7), 2100228. https://doi.org/10.1002/aisy.202100228