



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
COMPUTER SCIENCE AND ENGINEERING

Ciclo 38

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

DIGITAL TWINS ECOSYSTEMS FOR INTELLIGENT APPLICATIONS IN
HEALTHCARE

Presentata da: Samuele Burattini

Coordinatore Dottorato

Paola Salomoni

Supervisore

Alessandro Ricci

Co-supervisore

Mirko Viroli

Esame finale anno 2026

Dottorato di Ricerca in
Computer Science and Engineering

Ciclo XXXVIII

Settore Concorsuale: 09/H1
Settore Scientifico Disciplinare: ING-INF/05

Digital Twins Ecosystems for Intelligent Applications in Healthcare

Candidato:
Samuele Burattini

Coordinatrice Dottorato:
Prof.ssa Paola Salomoni

Supervisore:
Prof. Alessandro Ricci

Co-supervisori:
Prof. Mirko Viroli
Dott. Ing. Angelo Croatti

Abstract

Italiano

La digitalizzazione della sanità è vista come una evoluzione necessaria per migliorare efficienza e qualità dei sistemi sanitari, aumentando la tracciabilità e supportando il processo decisionale. In questo contesto, il concetto di Digital Twin (DT) si è affermato come un’astrazione per modellare e rappresentare asset fisici, costruendone una replica software continuamente aggiornata tramite dati raccolti dal mondo reale.

Tuttavia, domini complessi come quello sanitario sono caratterizzati da asset eterogenei le cui relazioni sono cruciali per fornire una rappresentazione completa e accurata del contesto fisico. Questa tesi esplora le sfide ingegneristiche legate allo sviluppo di *Ecosistemi di Digital Twins (DTE)* che integrano DT eterogenei connessi tra loro da relazioni dinamiche. Il concetto di DTE è proposto come layer di astrazione a supporto dello sviluppo di applicazioni intelligenti in grado di sfruttarne le capacità per acquisire una visione globale del mondo fisico e automatizzare processi. Per affrontare tali sfide, il lavoro introduce soluzioni architetturali per lo sviluppo di DT focalizzate su modularità e interoperabilità. L’approccio è completato dall’introduzione di descrizioni (semantiche) per DT, volte a supportare sia la gestione operativa che l’integrazione nei DTE. Ispirandosi alla natura basata su hypermedia del Web e al Web of Things, è stato sviluppato un prototipo di *Hypermedia Web of Digital Twins (HWoDT)* per l’implementazione di DTE. Infine, viene esplorato l’uso dei Sistemi Multi-Agente (MAS) come mezzo per realizzare applicazioni intelligenti e affidabili nei DTE, chiarendo la relazione con i DT e contribuendo all’ingegneria dei MAS basati sul modello Belief-Desire-Intention. L’approccio proposto è validato attraverso l’analisi e lo sviluppo di scenari applicativi nel dominio sanitario.

Parole Chiave: Digital Twin, Internet of Things, Web of Things, Cyber-Physical Systems, Artificial Intelligence, Multi-Agent Systems, Software Engineering

English

The digitalization of healthcare is envisioned as a necessary development to improve the efficiency and quality of healthcare systems worldwide by improving traceability and supporting decision-making. In this context, the concept of Digital Twin (DT) has emerged as a powerful abstraction to model and represent physical assets by creating a software replica which is continuously updated with data from the physical world.

Complex domains such as healthcare, however, are characterized by a variety of heterogeneous assets whose interrelations are crucial to provide a complete and accurate representation of the physical context. This thesis explores the engineering challenges in the development of *Digital Twin Ecosystems (DTEs)* integrating heterogeneous DTs connected by dynamic relationships. The work further propose the DTE as a layer of abstraction to support the development of intelligent applications which can exploit its capabilities to acquire a comprehensive view of the physical world and automate processes. To address these challenges, this work proposes architectural and engineering solutions for the development of DTs focusing on modularity and interoperability. This approach is complemented by the introduction of (semantic) descriptions for DTs that can support both the operational management and integration challenges in DTEs. Through an approach inspired by the hypermedia nature of the Web and the Web of Things a prototype for a *Hypermedia Web of Digital Twins (HWO DT)* which can be used to implement heterogeneous DTEs is developed. Finally, the thesis investigates the use of Multi-Agent Systems (MAS) as a way to implement trustworthy intelligent applications in DTE, defining their relationship with DTs and contributing to the engineering of MAS based on the Belief-Desire-Intention model. The proposed approach is validated through the analysis and development of application scenarios in the healthcare domain.

Keywords: Digital Twin, Internet of Things, Web of Things, Cyber-Physical Systems, Artificial Intelligence, Multi-Agent Systems, Software Engineering

*The most profound technologies are those that disappear.
They weave themselves into the fabric of everyday life
until they are indistinguishable from it.*

– Mark Weiser

Acknowledgements

Coming at the end of this PhD journey, I want to express my deepest gratitude to all those who have supported and inspired me throughout this experience.

First and foremost, I want to thank my partner Giada for her constant support, patience and encouragement. Her belief in me has been stronger than mine at times, and she endured all the ups and downs of the PhD life with me, offering love and comfort, but also strong advices that helped to bring me out of my overthinking and procrastination. Thank you for being my companion in this adventure, I hope to continue sharing all the meaningful milestones of our growth.

I am deeply grateful to my family, for their support of my academic pursuits. The love and interest that all of you have shown in my work, even without fully understanding it, has been a source of motivation and pride for me. Thanks to my sister, and especially to my parents, for the gift – not without sacrifices – of allowing me to live and study in Cesena. Giving me that opportunity set me on a path that led me here, almost ten years later. Your support, constant to this day, is never taken for granted and I hope to make you proud.

I want to thank my friends, both the old ones in Ancona who basically know me since forever now, and the ones of the *Spaceteam* who shared the study journey and stuck with me even afterwards. Being able to always count on you for fun times even if I'm not always around is something I really value and will continue to cherish. Special mentions must go to Fra Antici, for knowing on his skin the struggles of being a researcher and helping me to feel less alone in this journey, to Mengo, who involuntarily set me on this path by suggesting to check out this university, to Delu and Fra Dente for being the best group to tackle any project, and to Tommi whose quiet life philosophy always makes him great company to have around. I follow your efforts in making it on your own with *Meravit* with great pride and I hope to see you all succeed in your dreams.

A very special thank goes to all the colleagues and friends I have the pleasure to share my days in Area 4.0 and the Pervasive Software Lab. Your presence in and outside the office made even the most stressful days more bearable and fun even if simply by sharing memes, lunch breaks, coffee or a Friday beer at the Sthop. Special mention for Gianluca for truly being a shining light of positive energy in

the lab since the first day I set foot in it, I deeply cherish our conversations from one end to the other of the lab sitting behind our monitors. Thanks to Martina for sharing the passion of BDI agents (but also plants, cooking, etc.) and allowing me to explore many research ideas together while not losing track of the joys of programming and finally making things work. Thanks also to Angela, Davide, Nicolas, and Mattia for sharing the office gossip and the struggles of the PhD life, and to Matteo, Paolo and Ruslan for sharing this three year adventure from start to end. I hope all of you will achieve the success you deserve in your careers.

All the people mentioned, and many others I have not listed explicitly, have made this personal journey and life experience richer and more meaningful. But the PhD is also a professional journey, and for this reason I want to express my deepest gratitude to all the people whose guidance, mentorship and collaboration have been fundamental in my growth as a researcher.

First, I want to thank my PhD advisor prof. Alessandro Ricci: his inspiring attitude towards research, his dedication to teaching, and his ability to deeply fall in love with new research topics and ideas even after many years are an example for the kind of researcher I aspire to be. Thank you for trusting me from the beginning, for pushing me to think critically about my work, and setting me on a path of exploration and discovery that has shaped my academic career.

I also want to thank my co-advisor Angelo Croatti, who made this PhD possible through the collaboration with AUSL Romagna, my co-advisor prof. Mirko Viroli for supervising my work over the milestones of the PhD journey, and prof. Sara Montagna for being of great support in the challenges of applying research to healthcare settings. Your experience and insights have been helpful in keeping me grounded in potential real-world applications of my research.

Thanks to the reviewers of this thesis, prof. Lukas Esterle and prof. Marco Mamei, for accepting to evaluate my work and for the valuable feedback that helped me improve the quality of this document.

I am deeply grateful to prof. Marco Picone, whose expertise in IoT and passion for Digital Twins have made our collaboration extremely enriching and fruitful. Thank you for dedicating so much time in brainstorming ideas and putting them into practice. Thanks also to the rest of his team in University of Modena and Reggio-Emilia that worked on the WLDT framework, to Stefano Mariani for his contribution in joining agents and Digital Twins in a clear picture, to the team in University of Ferrara for the inspiring work on augmentation, and to the *Twinkle* project team for the collaboration on the Digital Twin Continuum.

I thank prof. Danilo Pianini and Giovanni Ciatto for their unofficial mentorship and constant support in my growth as a researcher. Your insights and feedback on the academic world and research practices have been invaluable in helping me navigate the PhD journey and your efforts in building a supportive research

community in Cesena have made a difference for all of us.

Thanks also to all the students that I had the pleasure to supervise and who helped me to explore research ideas, especially to Andrea Giulianelli for contributing to the Hypermedia Web of Digital Twins ideas and making the prototype a reality, and to Elena Yan for her dedication in exploring explainability techniques for BDI agents.

A very special thank goes to the mentors of my research visits abroad. To prof. Simon Mayer for introducing me to the world of research while supervising my master thesis and for always making me feel my opinion as a researcher mattered ever since. Having the chance to work together again in Cesena during my PhD has been a pleasure and an honor. To prof. Rem Collier who hosted me in Dublin making me truly feel at home. Thank you for your enthusiastic support and giving me the chance to live in one of my favorite cities while working together. To prof. Antoine Zimmermann for hosting me in Saint-Étienne, and for the help in understanding knowledge graphs and semantic Web technologies that greatly influenced my research.

Thanks also to all the international collaborators I met along the way, I cannot list you all, but I appreciate greatly that I've been able to connect with such a diverse and talented group of researchers around the world. Special mention to Andrei Ciortea for sharing with me his expertise on hypermedia systems and for leading the work of the Web Agents community group, to Danai Vachtsevanou for the discussions about agent adaptability, to Katharine Beaumont for surprising me with her passion for my own research, and, finally, to prof. Jomi Hübner whom I had the pleasure to work with during his visit in Cesena and whose humble attitude and deep passion for BDI agents has been truly inspiring.

I fundamentally believe that research is a collaborative effort that thrives on diverse perspectives and expertises of the people that are passionate about it, and I consider myself fortunate to have met and worked with so many talented individuals during this journey. This PhD work would not have been possible without you all, and I am deeply thankful for your contributions to my growth as a researcher and as a person.

Contents

Abstract	iii
Introduction	1
I Background and Context	9
1 Healthcare Digitalization	11
1.1 Healthcare 4.0	11
1.2 Hospital Information Systems	15
1.3 Medical Standards for Interoperability	18
1.4 Intelligent Applications in Healthcare	21
1.5 The AUSL Romagna Context	23
1.6 Final Remarks	25
2 Digital Twins	27
2.1 History and Definitions	27
2.2 Reference Architectures and Properties	31
2.3 Enabling Technologies and Platforms	34
2.4 Digital Twins and Artificial Intelligence	37
2.5 Digital Twins in Healthcare	39
2.6 Digital Twin Interoperability	41
2.7 Towards Digital Twin Ecosystems	45
2.8 Final Remarks	48
3 The Semantic Web and The Web of Things	49
3.1 History and Definitions	49
3.2 The Web Architecture and Hypermedia	52
3.3 Semantic Web Technologies	53
3.4 The Web of Things	57
3.5 Final Remarks	61

4	Multi-Agent Systems	63
4.1	History and Definitions	63
4.2	Agent-Oriented Software Engineering	67
4.3	Belief-Desire-Intention Agents	71
4.4	Web-based and Hypermedia MAS	73
4.5	Multi-Agent Systems and Digital Twins	76
4.6	Final Remarks	78
II	Digital Twin Ecosystems	79
5	Engineering Digital Twins	81
5.1	An Abstract Architecture for Digital Twins	81
5.2	Modular Design of Digital Twin Interfaces	86
5.2.1	Physical Interoperability	89
5.2.2	Digital Interoperability	91
5.2.3	Web of Things: enabling DT Interoperability	93
5.3	Managing the Digital Twin Lifecycle	94
5.3.1	A Digital Twin Synchronization Lifecycle	95
5.3.2	Binding the DT with the PA	96
5.3.3	Managing DT and PA Synchronization	97
5.4	Modeling Augmentation Functionalities	102
5.4.1	Extending the DT Model with Augmentation	103
5.4.2	Impact on Shadowing	105
5.4.3	Engineering Augmentation Functions	107
5.5	Extending the White Label Digital Twins Framework	110
5.6	Evaluation of the Proposed Approach	113
5.6.1	Industrial Scenario Description	113
5.6.2	Digital Twin System Architecture	115
5.6.3	Reducing Complexity through Reusable Adapters	117
5.6.4	Lifecycle Awareness and Coordination	122
5.6.5	Performance Evaluation	124
5.6.6	Implementing Augmented Digital Twins	126
5.7	Final Remarks	130
6	Digital Twin Ecosystems	133
6.1	Characterizing Digital Twin Ecosystems	134
6.2	Architectural Patterns	136
6.3	Heterogeneity and Trade-offs	139
6.4	Challenges of Digital Twin Ecosystems	141

7	Interoperable Digital Twin Ecosystems	143
7.1	A Hypermedia Web of Digital Twins	144
7.1.1	A Uniform Interface for Heterogeneous DTs	144
7.1.2	Ecosystem Services in the HWoDT	148
7.2	A Prototype Framework for the HWoDT	150
7.2.1	A WoT-compatible Digital Twin Description	150
7.2.2	HWoDT Adapters	152
7.2.3	The WoDT Platform	156
7.3	Comparison with Related Works	158
7.3.1	HWoDT and Digital Twin Standards	158
7.3.2	HWoDT in the Landscape of DT Platforms	162
7.4	Prototype Performance Evaluation	168
7.4.1	Experimental Setup	169
7.4.2	Results	170
7.5	Final Remarks	173
8	Operational Management of Digital Twin Ecosystems	175
8.1	Operational Challenges	176
8.2	Phases and Stakeholders	177
8.3	Key Elements and Descriptions	181
8.4	Architecture of the Digital Twin Continuum	185
8.4.1	Functional Overview	186
8.4.2	Interaction Example	187
8.5	Proof of Concept Implementation	189
8.6	Final Remarks	195
 III Intelligent Applications with Digital Twins and Multi-Agent Systems		 197
9	Multi-Agent Systems and Digital Twins	199
9.1	Complementarity of DTs and MAS	199
9.1.1	Use of AAs and DTs in IoT Systems	201
9.1.2	Intelligent Functionalities	202
9.2	Design Principles	204
9.2.1	Specificity	204
9.2.2	Scoping	206
9.2.3	Timing	207
9.3	Matching AAs and DTs to Design Principles	209
9.4	Architectural Integration Patterns	211
9.5	Exemplary Application of the Principles	215

CONTENTS

9.5.1	A Smart Manufacturing Scenario	216
9.5.2	System Design with AAs and DTs	218
9.5.3	Discussion of Resulting Architecture	221
9.6	Final Remarks	223
10	Engineering BDI Multi-Agent Systems	225
10.1	BDI Agents in Digital Twin Ecosystems	225
10.2	Tooling for BDI Agent Programming	228
10.2.1	State of the Art	229
10.2.2	Desiderata for Mainstream BDI Tooling	230
10.2.3	The JaKtA Framework	233
10.3	Testing MAS with Simulation	236
10.3.1	Decoupling Concurrency in BDI MAS	237
10.3.2	Discrete-Event Simulation for BDI	238
10.3.3	Experimental Demonstrator	240
10.4	BDI Agents in Hypermedia Environments	242
10.4.1	A Generalized BDI Engine	243
10.4.2	Embodiment for Agents on the Web	245
10.4.3	Enhancing BDI Agent Adaptability	247
10.5	Explainability in BDI Agents	249
10.5.1	Multi-Level Explainability	249
10.5.2	Inter-Agent Explainability	251
10.6	Final Remarks	253
IV	Applications Scenarios in Healthcare	255
11	Pharmaceutical Supply Chain	257
11.1	Case Study Analysis	258
11.2	Digital Twin Ecosystem Proposal	261
11.3	Implemented Solution	263
11.4	Final Remarks	265
12	Trauma Management	267
12.1	Case Study Analysis	268
12.2	Digital Twin Ecosystem Proposal	269
12.3	Implemented Solution	270
12.4	Final Remarks	274

CONTENTS

13 Operating Room Management	277
13.1 Case Study Analysis	279
13.2 Digital Twin Ecosystem Proposal	282
13.3 Implemented Solution	283
13.4 Final Remarks	286
 Conclusion	 289
 Funding	 301
 Bibliography	 303

CONTENTS

List of Figures

1.1	Functional subsystems in a Hospital Information System, adapted from [Deg14].	17
1.2	Map of the AUSL Romagna territory, credits: WikiMedia	24
2.1	The original Digital Twin (DT) conceptual model by Grieves, from [Gri22].	28
2.2	Taxonomy of Digital Twin related concepts based on the data flows between physical and digital objects, adapted from [KKT ⁺ 18]. . . .	30
2.3	Conceptual representation of the 5D model of a Digital Twin, showing the five dimensions and the connections between them.	32
2.4	The DT-as-a-Service reference architecture, from [AXZL21].	33
2.5	Artificial intelligence with respect to DT capabilities, from [MCFA23].	38
2.6	An overview of DT for healthcare, showing technological enablers, target goals and the main axis that are relevant for healthcare DTs, namely time and scale. Image from [KWW ⁺ 24].	39
2.7	Interoperability Levels in Digital Twin, from [AKP24].	43
2.8	Conceptual model of the Web of Digital Twins. Physical Assets (squares) have corresponding Digital Twins (circles) connected through relationships that mirror the ones in the physical world. Applications and users (top) can navigate, query and observe the Web of Digital Twins to access data and services from the interconnected Digital Twins.	46
3.1	A slide showing the concept of the Semantic Web, with documents linked to real-world concepts through semantics, from [BL94b]. . . .	51
3.2	The Semantic Web “cake” showing the layers of the technological stack, from [Bra07].	54
3.3	The main layers of the World Wide Web Consortium (W3C) Web of Things (WoT) architecture, from [GT16].	58
3.4	An overview of the W3C WoT architecture, from [TLMM23].	59

LIST OF FIGURES

4.1	Classic schematic representation of an agent and its interaction with the environment, from [RN95].	64
4.2	High-level overview of the main programming dimensions of a Multi-Agent System (MAS), the <i>interaction</i> dimension, not shown, focuses on interaction between agents, from [BBH ⁺ 13].	69
4.3	Schematic representation of the main abstractions of Multi-Agent Oriented Programming (MAOP) and their relationships, from [BBH ⁺ 13].	70
4.4	Schematic, high-level representation of a Belief-Desire-Intention (BDI) agent, from [BEFSH ⁺ 20].	72
4.5	Schematic representation of the concept of Hypermedia Multi-Agent System showing how elements of a MAS can be integrated and represented in a hypermedia environment for agents to discover, from [CMG ⁺ 19].	75
4.6	Schematic representation of the integration between MAS and DTs through the environment dimension. DTs can be seen as artifacts which the agents interact with, from [MPR22].	77
5.1	A side-by-side mapping between 5D DT modelling (left) and the abstract DT architecture presented in this section (right). The architecture is presented with named boxes that highlight the main components of a DT and its interaction with the Physical Asset (PA) and external applications.	82
5.2	Example of a DT in an application scenario interacting with a physical asset and digital applications through its Physical Interface (<i>PI</i>) and Digital Interface (<i>DI</i>). The example showcases the presence of multiple communication channels on both sides to highlight the heterogeneity of physical and digital interactions.	88
5.3	Physical Interface design with modular physical adapters each producing the corresponding Physical Asset Description that is processed by the DT Model.	89
5.4	Digital Interface design with modular adapters. Each adapter publishes a description of the state, events and actions it supports so that consumers can dynamically understand and interact with the DT.	91
5.5	DT Life Cycle with its phases and transitions. Circles represent phases, rounded squares represent terminal phases, and arrows represented transitions. Phases are grouped in three major operating states of the DT which are Starting , Working and Stopping	95

LIST OF FIGURES

5.6	Zoomed view on the DT lifecycle with a structured example of the Synchronized phase from Figure 5.5. The images shows that within the Synchronized phase, the DT reflects the PA’s operational phases, such as Ready , Working , Busy	98
5.7	Schematic Event-Driven architecture of a DT to support Augmentation function management and execution. A central event bus supports asynchronous interactions between the shadowing and augmentation functions. Functions can access the DT models and history as needed.	107
5.8	Patterns for augmentation functions execution: internally (left), externally (center), or through a hybrid solution (right).	108
5.9	Extended WLDT software architecture aligned with the DT engineering model proposed in this thesis.	111
5.10	A schematic view of the microfactory production line, with the two departments.	114
5.11	Multiprocess Station Digital Twin architecture overview.	116
5.12	Relationships between DTs in the Multiprocess Station serve the purpose of tracking the physical relationships between different machines at the digital level.	117
5.13	The Digital Twin ecosystem architecture of the Indexed Department, showing the different Physical Adapter (PhA) and Digital Adapter (DiA) of each DT alongside a schematic representation of the conveyors.	118
5.14	The transition from physical to digital representation via the Physical Asset Description (PAD), the DT State, and the Digital Twin Description (DTD) for external consumers.	119
5.15	Cyber-physical complexity computed on different levels of the hierarchical DT system implemented for the microfactory, showing reduced complexity compared to an approach without modular DTs interfaces.	121
5.16	Lifecycle of a PA in the target industrial scenario, highlighting the different phases that the DT needs to account for when synchronized.	123
5.17	Performance indicators of the Multiprocess Station DTs: CPU usage percentage [%], Memory usage [Mib], Network usage [KBit/sec] and OEE [%]	125
5.18	Experimental setup for evaluating the design of augmentation functions for a pose-estimation task in a DT of a human operator.	126
5.19	Pose estimation latency computed within the DT compared to standalone execution, using the CPU (a) and using the GPU (b).	129

LIST OF FIGURES

6.1	Three architectural approaches for Digital Twin Ecosystems (DTEs): in 6.1a a single DT models the whole scenario; in 6.1b a single platform is used to build and deploy all the DTs in the ecosystem; in 6.1c maximum flexibility is allowed, possibly reusing existing DTs under a common interface. Squares represent PAs, circles representn DTs and colored shapes define boundaries between different organizations that may adopt different technological stacks.	137
7.1	The Hypermedia WoDT scheme, in the real world PAs (squares) belonging to different organizations and domains are connected by relationships. In the digital world, cross-domain ecosystems of DTs (circles) mirror different portions of reality, supported by the WoDT Platform. Consumers (top) can interact with the ecosystem and its DTs directly, or through the platform.	145
7.2	Interaction patterns to (a) access the DTD and Digital Twin Knowledge Graph (DTKG) and (b) invoke available actions on a DT through the uniform interface and standard Web patterns.	148
7.3	Overall abstract architecture of the Hypermedia Web of Digital Twins (HWoDT) framework, showing the main functional modules and their main interactions. On the top, the Web of Digital Twins (WoDT) platform manages the ecosystem of DTs by registering them through their DTDs, observing their DTKGs, and exposing the DTE Knowledge Graph (KG) to consumers. On the bottom, existing DTs can implement the HWoDT uniform interface through the HWoDT adapter and be integrated with the platform.	151
7.4	Main components of the Azure Digital Twins adapter, implemented as a standalone middleware that maps specific instances of DTs hosted on an Azure Digital Twins instance.	154
7.5	Main components of the Eclipse Ditto adapter, implemented as a standalone middleware that maps a specific instance of a DT hosted on the Ditto platform.	155
7.6	Main components of the White Label Digital Twins (WLDT) adapter, implemented as a Digital Adapter and illustrating its integration with the other components of the WLDT framework.	156
7.7	WoDT Platform modules and interfaces, represented using components and connectors.	157
7.8	Comparison of the steps required to perform a “query” across heterogeneous DTs versus with the HWoDT approach, which abstracts heterogeneity through a uniform interface.	162

LIST OF FIGURES

7.9	Latency in an HWoDT-based system. From left, to right, the PA, the DT, the HWoDT adapter, the WoDT platform and the client. Each connection between components (labelled with letters) may introduce latency.	169
7.10	Performance indicators with different loads of DTs and clients working on the ecosystem: (a) average time difference when observing DTs either directly or through the WoDT platform, and (b) average time per operation when querying the DT ecosystem either through the WoDT SPARQL endpoint or directly retrieving the state from each DT API.	171
8.1	Stakeholders involved in different phases of the development and management of DTs in an ecosystem.	178
8.2	Requirements associated to each phase of the DT development-to-deployment process.	180
8.3	Key elements and their descriptions depicted within a schema of the Digital Twin Continuum (DTC). PAs instances (bottom) are described by schemas and each have a DT instance mirroring them, running on possibly different platforms depending on which DT package among the ones implementing the correct DT Schema (top) is used to deploy the DT.	185
8.4	Functional Architecture of the DTC with the main components and the interactions with stakeholders. On the left, DTC nodes abstracting the interaction with physical computing resources. On top of them, DT packages belonging to a given schema are deployed as DT instances. The DTC Manager (right) is composed by a middleware and an orchestrator. The first is responsible for managing the registry of DTs and PAs, while the second is responsible for orchestrating the deployment of DTs across the DTC nodes.	186
8.5	The interaction flow example between DTC architectural components to enable DT registration and deployment.	188
8.6	Comparison of end-to-end DT state computation delays across edge (WLDT) and cloud (Ditto) platforms.(a) shows the same DT model being deployed on both edge and cloud, while (b) shows a coarser DT model being deployed on the cloud. Below each figure, the number of DT states computed in the last 10 seconds, to highlight the difference in the update frequency of the two DTs.	191
8.7	Number of active DT instances on the Edge and Cloud platforms during the incremental deployment process. The graph illustrates the gradual increase in the number of DTs over time, reflecting the phased deployment strategy adopted in the experiment.	193

LIST OF FIGURES

8.8	Evaluation of the DTC prototype enabling multi-platform DT deployment in the Microfactory. The figure compares resource usage (CPU, Memory, Network Sent and Received) for Ditto in the Cloud (top row) hosting multiple DTs within a single Ditto instance composed of several microservices (Connectivity, UI, Gateway, Nginx, Policies, Things, Things-Search) and WLDT at the Edge (bottom row) deploying individual DTs as microservices (DT1,..., DT12) as the number of active DTs increases.	194
9.1	The spectrum of <i>specificity</i> : from intelligent functionalities specific to an application goal to general-purpose services.	205
9.2	The spectrum of <i>scoping</i> : from intelligent functionalities requiring knowledge of the whole system, to those confined to individual assets.	206
9.3	The spectrum of <i>timing</i> : from intelligent functionalities requiring explicit modelling of physical or logical time, to those that don't care.	208
9.4	Exemplary intelligent functionalities placed within the spectrum of the proposed design principles. AAs and DTs are clustering those functionalities that they better match with, according to such principles.	209
9.5	Architectural perspective for the synergistic combination of AAs and DTs in distributing intelligence in IoT systems. Each letter denotes a specific <i>micro-architecture</i> that may arise from the application of our principles. Such architectures can be combined to give shape to the overall system architecture, in a bottom-up way. . . .	213
9.6	A reference Smart Manufacturing scenario with multiple physical entities and a set of applications interested in implementing intelligent functionalities on top of the physical deployment.	216
9.7	Three main objectives that concur to the energy optimization of a Smart Manufacturing plant: Energy Monitoring at the machine level, Intelligent Coordination between robots and operators, and overall Production Optimization based on external factors (e.g. energy cost).	217
9.8	Three illustrative instances showcasing the synergistic usage of AAs and DTs as guided by the principles and μ -archs, within the Smart Manufacturing context.	220
9.9	A layered architecture with the combination of DTs and Agents for a Smart Manufacturing reference scenario.	221
10.1	The main modules composing the JaKtA framework and their dependencies.	234

LIST OF FIGURES

10.2	The agent specifications (left) are completely platform-agnostic and reusable, some glue code (right) wires the logics with the underlying execution platform.	240
10.3	Average error with time for different granularities, with Atomic Control-Loop Iterations (ACLI) and Atomic Control-Loop Phase (ACLP) running at $f = 1Hz$ and $f = 2Hz$. Different charts show different values of relative drift τ . Coloured shadows represent $\pm 1\sigma$ over multiple runs.	241
10.4	The generalized BDI engine architecture. The core BDI reasoning cycle is decoupled from the specific implementations of beliefs, goals, plans, and reasoning mechanisms. This allows for specializing the engine to work with different representations and logics.	244
10.5	The explainability process converting system logs into narratives that can be used to derive explanations for BDI agents.	251
10.6	Graphical representation of the multi-level explainability framework for BDI agents.	252
11.1	Entities in the system and their relationships, blue ellipsis represent the ones selected to be reified as Digital Twins.	261
12.1	The HWoDT-based DTE for the trauma management case study. DTs implemented with different technologies (WLDL, Ditto, Azure) become interoperable through the HWoDT Adapters and are integrated through the WoDT Platform. Black dotted arrows represent the interaction with the platform, black dashed arrows represent relationships between DTs while grey arrows represent the interaction between consumers, the platform and DTs.	269
13.1	An example of how a DT ecosystem could support the management of ORs as well as the real-time tracking of elective surgery processes.	282
13.2	The final FHIR resources mapping data related to operating rooms and procedures. Each box represent a FHIR resource, while arrows represent fields holding references between resources.	284
13.3	The expanded view of an operating room, showing the surgical procedures of that room and their steps.	286
13.4	Snapshot of the 3D view of the AnyLogic simulation of surgical processes used to generate synthetic data for the DTE prototype. .	287

LIST OF FIGURES

List of Listings

7.1	The Thing Model that Thing Descriptions must implement to be recognized as a valid Digital Twin Description.	151
10.1	Overall structure of a MAS specification in JaKtA.	234
10.2	Example of paradigm-blending in JaKtA. The code snippet uses together Object-Oriented Programming (OOP), Functional Programming (FP), and Imperative Programming (IP) constructs to create a MAS.	235
10.1	A description of agent <code>alice</code> and its body artifact.	246
11.1	Knowledge Graph representing a model of the data collected by the DT system in FHIR RDF.	263
12.1	SPARQL Query performed by the Mission Agent to obtain the available ambulances and rescuers.	271
12.1	Portion of the Ambulance DTD that shows the relevant data for the <code>SetDestinationCommand</code> action	271
12.2	The Mission DTKG when the ambulance reaches the patient, showing the relationship with the patient.	272
12.3	SPARQL Query performed by the trauma team support tools to obtain the ambulance and patient observation endpoints.	273
12.4	SPARQL Query performed by the Centralized Emergency Unit (CEU) Dashboard to obtain the currently active missions.	274

LIST OF LISTINGS

Introduction

Context and Motivations

The last decade has been marked by a significant transformation driven by the integration of advanced digital technologies such as the Internet of Things (IoT), Artificial Intelligence (AI) and Machine Learning (ML) to improve the efficiency and automation of processes across manufacturing and service industries. This digital transformation, often referred to as the *Fourth Industrial Revolution* or *Industry 4.0* [LFK⁺14, KWH13], is now becoming a pervasive trend impacting even less technology-driven sectors such as agriculture [LMS⁺21] and healthcare [TFMCV⁺20].

In the healthcare sector, the integration of digital technologies is reshaping the way care is delivered, with a focus on improving patient outcomes, enhancing operational efficiency, and enabling personalized medicine [KSPI21]. Some see this transformation as a way to address the challenges of an aging population, rising healthcare costs, and the need for more efficient and effective care delivery towards accomplishing the United Nations' Sustainable Development Goal (SDG) 3 aiming to ensure healthy lives and promote well-being for all at all ages [Uni25].

This Ph.D. thesis is the outcome of a collaboration with Azienda Unità Sanitaria Locale della Romagna, the local public healthcare authority in the southern region of Emilia-Romagna, Italy, aimed at exploring the potential of digital technologies to enhance the internal processes of the local healthcare system.

This exploration follows the rising trend of digitalizing the core assets of an application domain with *Digital Twins (DTs)*: representations of Physical Assets (PAs), connected and synchronized through data streams from sensors and other

digital sources [Gri23]. The DT concept has emerged as a paradigm shift in how to model and interact with the physical world when designing Cyber-Physical Systems (CPSs), focusing on capturing the essential features of each relevant entity, creating personalized models and combining real-time data with modeling techniques, AI and simulation [SLPD21]. The focus of DTs towards individualized modeling is well-paired to the digitalization needs of the healthcare domain supporting a patient-centric approach [AKK⁺23]. Furthermore, the ability of DTs to integrate data from various sources – spanning from legacy systems to IoT devices – and provide a holistic view of the system is beneficial for the development of *intelligent applications* which can leverage the data and insights provided by DTs to support – or partially automate – healthcare processes.

Overview and Contribution

Adopting a DT-based approach for healthcare digitalization is not without challenges. Healthcare organizations often operate with fragmented processes and legacy systems that hinder the seamless flow of information. Moreover, healthcare processes often involve a variety of very different assets – including facilities, medical devices, patients and healthcare professionals – whose interrelations are crucial for the understanding of workflows. Finally, for an effective monitoring and decision-making support, it is crucial to have an always up-to-date view of all the assets involved. Given the safety critical setting of healthcare, automated decisions must be transparent and reliable, albeit capable of adapting to the dynamic nature of healthcare environments.

These challenges call for an integrated approach towards the development of both DTs and applications. This thesis, proposes a framework for the development of *Digital Twin Ecosystems (DTEs)* that can incorporate heterogeneous DTs and expose a holistic, synchronized view of the mirrored physical environment to applications under a unified interoperable interface. To implement *heterogeneous* DTEs the thesis explores architectural patterns and best practices taking inspiration from the (semantic) Web, as an example of large-scale decentralized system that successfully integrates heterogeneous resources and supports their interaction through standardized protocols and data formats [SBLH06]. Namely, the approach

is grounded on *hypermedia* representations that can semantically describe the assets and their relationships, and support navigation and interaction through hyperlinks [Fie00]. To support the development of intelligent applications, this work refers to Multi-Agent System (MAS) as an engineering paradigm for autonomous systems which model distributed applications in dynamic environments through sets of autonomous entities [Woo09]. Specifically, the thesis focuses on cognitive approaches such as the Belief-Desire-Intention (BDI) model [Bra87, RG91], as they merge adaptability and responsiveness with a declarative encoding of domain knowledge and business rules.

Problem Statement

Digital transformation, can be supported by Digital Twins creating personalized models of relevant assets and support their tracking over workflows. Yet, legacy systems, assets heterogeneity and dynamically evolving relationships hinder the creation of a holistic view that can be leveraged by intelligent, transparent, and adaptive applications to support and automate processes. Addressing this gap is essential for the successful implementation of Digital Twin Ecosystems making processes more efficient and supporting decision-making.

Towards this goal, the thesis aims to answer these research questions:

RQ1 *How to engineer DTs which integrate heterogeneous data sources and provide an up-to-date uniform representation of the mirrored assets?*

To effectively implement DTEs, it is necessary to investigate first how to engineer DTs that can mediate interaction with their physical counterpart and give access to external consumers to the asset's state and capabilities.

RQ2 *How to engineer DTEs which integrate heterogeneous DTs and support applications in interacting with them?*

Given the fragmented nature of DTs it is essential to ensure that DTEs can integrate heterogeneous DTs and offer features that facilitate the interaction of external applications with the set of DTs within an ecosystem.

RQ3 *How to support the operational management of DTEs?*

DTs have a complex development-to-deployment lifecycle that may involve

multiple stakeholders. Supporting the operational management of DTEs requires addressing the challenges of supporting these stakeholders alongside the lifecycle, giving them access to a comprehensive view of the DTs involved in the ecosystem and support the deployment of DTs in a complex cyber-physical computing infrastructure.

RQ4 *What are the synergies between DTEs and MAS for the development of intelligent applications?*

Given the dynamic nature of DTEs, applications interacting with DTs need to be flexible, and encapsulate strategies and rules to automate decision-making. This thesis explores how (cognitive) MAS can be used to this purpose and in general how to exploit both autonomous agents and DTs to engineer intelligent cyber-physical applications.

RQ5 *How to make the development of BDI agents more accessible and easier to integrate with external systems such as DT?*

BDI agents are a powerful tool for modeling intelligent applications, but their development is often complex as they require a deep understanding of the underlying concepts. This work explores how to make the development of BDI agents more accessible and easier to integrate with external systems looking at the design of new tools and frameworks.

Hypothesis

The central hypothesis of this thesis can be summarized as follows:

A Digital Twin Ecosystem, integrating heterogeneous DTs through semantic and interoperable representations, can provide a unified, always up-to-date view of PAs, thereby enabling the development of intelligent, adaptive applications capable of supporting and partially automating processes in complex domains, such as healthcare.

To validate this hypothesis, the research questions are addressed with the following strategy:

- **RQ1–2:** Investigate how heterogeneous DTs can be integrated and represented within a DTE to provide a coherent, semantically-rich view.
- **RQ3:** Focus on the operational management of the DTE, ensuring its practical deployment and lifecycle support.
- **RQ4–5:** Explore how multi-agent systems, and particularly BDI agents, can leverage DTEs to develop intelligent, adaptive, and explainable applications.

By investigating this hypothesis, the thesis aims to demonstrate that a structured approach to DTE engineering can effectively combine interoperability, real-time synchronization, and intelligent autonomous behavior, forming a foundation for robust and adaptable cyber-physical applications. The validation of the hypothesis is achieved through the definition of architectural frameworks, patterns, and prototype implementations, as well as the application of the proposed approach to real-world healthcare scenarios provided by the industrial partner AUSL Romagna.

Contributions

Despite being motivated by the digitalization of healthcare processes through DTEs, this thesis contributions can be interpreted in the broader scope of engineering CPSs with DTs. The area of DT engineering is a rapidly evolving field, with ongoing research aimed at improving the design, implementation, and management of DTs in various application domains where DTs are being employed [MYH⁺22]. In this context, this thesis contributions can be applied to different application scenarios sharing similar challenges in terms of integration of heterogeneous data sources, heterogeneous assets connected by dynamic relationships. Here are the key contributions of this work to the area of DT engineering:

- Proposal of an architectural framework for the development of DTs focusing on integration of heterogeneous data sources and interoperability with external applications [PMB⁺25a].
- Definition of architectural patterns and best practices for the implementation of DTs to accurately capture synchronization requirements and augmented

features [PMB⁺25b, FMB⁺25].

- Implementation of a prototype framework for the development of DTs that demonstrates the proposed architectural concepts and patterns [MBM⁺25].
- Definition of the concept of DTEs and its role in the context of DT-based systems [GBCR26].
- Proposal for a framework to implement DTEs that can integrate heterogeneous DTs and expose a unified interface for external applications using semantic hypermedia representations [GBCR26, GBCR24, BZPR24].
- Implementation of a prototype platform and toolkit to integrate DTs into a DTE [GBCR25].
- Proposal of an abstraction layer to handle the operational management of DTE and their deployment on a cyber-physical computing infrastructure [BBM⁺24].
- Alignment of the DT and MAS concepts to establish a synergetic approach for the development of intelligent applications [BMM⁺25, Bur23].
- Application of the proposed DTE approach to real-world healthcare scenarios [BMG⁺24, BMC⁺23]

Additionally, as this thesis leverages cognitive MAS to support the development of intelligent applications that can integrate with DTEs, it contributes to the area of BDI MAS engineering by:

- Proposing tools for the implementation of BDI agents in order to facilitate their integration with other programming paradigms [BBCP24, BBCP23, PBBC24].
- Exploring the role of concurrency in BDI agents and the use of simulation to test and verify BDI MAS in a controlled environment before deployment [BBCPed, BBC⁺24a, BBC⁺24b].

- Proposing a conceptual framework to derive explanations of BDI agents' behavior, enhancing the transparency and trustworthiness of BDI applications and exploring how explainability can support new agent-to-agent interactions [YBHR25, YBHR23, BYBC25, BYB25].
- Investigating the integration of BDI agents with hypermedia environments and with Domain Driven Design to bring development to the *knowledge level* [BBC⁺25, BCGR23, RBCC24, RCB25, RBCC24, LBMC23, RVG⁺22].
- Exploring the ability for agents to anticipate future scenarios and react accordingly modifying their behavior to find alternative courses of action [HBRM25b, HBRM25a]
- Exploring the role of BDI in the new landscape of Generative Artificial Intelligence (GenAI) agents [RMZ⁺24, CAB⁺25]

Thesis Structure

This thesis is organized as follows. Part I provides the necessary context and background on the key concepts that underpin this research, including an overview of the challenges of healthcare digitalization and the reality of AUSL Romagna in Chapter 1, moving on to introducing the concept of DT its technological characterization in Chapter 2, discussing the architectural principles of the (semantic) Web and how they can build interoperability, presenting the case study of the Web of Things (WoT) in Chapter 3, and finally giving an overview of the concepts related to MAS engineering that are relevant for this thesis in Chapter 4.

Part II delves into the contributions towards the definition of the concept of DTE. Namely, Chapter 5 presents architectural proposals and patterns for the development of DT that can be integrated into a DTE, then Chapter 6 introduces the concept of DTE, and explore its operational management and prototype implementation through the integration of heterogeneous DTs in Chapter 8 and Chapter 7 respectively.

Part III focuses on the role of multi-agent systems in the development of intelligent applications discussing their alignment for a synergetic exploitation with

DTEs in Chapter 9 and the contributions to the engineering of BDI agents towards the goal of making them more accessible for developers and users in Chapter 10.

Finally, Part IV presents application scenarios proposed by the industrial partner AUSL Romagna—namely, pharmaceutical supply chain (Chapter 11), trauma management (Chapter 12), and operating room management (Chapter 13)—to demonstrate the applicability of the proposed approach to DTE engineering in real-world healthcare settings, before concluding by discussing results and presenting open challenges to be addressed in future works.

Part I

Background and Context

Chapter 1

Healthcare Digitalization

As this thesis work is a partnership with the local public healthcare administration AUSL Romagna, this chapter focuses on providing context on information systems used in healthcare and the trends in digital transformation that have been shaping the sector in recent years. These have been driven by Internet of Things (IoT) and Artificial Intelligence (AI) technologies, enabling the automatic acquisition of large amounts of data, which need to be integrated across different systems and analyzed to extract useful information to support clinical decisions, motivating the need for interoperability and knowledge representation. The chapter provides an overview of the complexity of Hospital Information System (HIS), the main medical standards, and, finally, the context of AUSL Romagna and its digital transformation journey.

1.1 Healthcare 4.0

As any industrial sector, healthcare is constantly innovated by the introduction of new technologies. The adoption of Information and Communication Technology (ICT) technologies has been driving a digital transformation in healthcare [KSPI21] which has led healthcare institutions to require new digital capabilities to support their operations and manage the amount of clinical data they produce in order to improve quality of care and reduce costs.

Compared to other application domains, healthcare is particularly critical for

Area	Description
Patient-centered	Focus on improving patient experience, engagement, and outcomes through digital technologies.
Operational efficiency	Enhancing processes, resource management, and service delivery using ICT solutions.
Organizational factors	Examining how digital transformation affects organizational structures, culture, and management practices.
Impact on workforce practices	Assessing changes in healthcare professionals' roles, skills, and workflows due to digital innovation and the co-design of solutions.
Socio-economic aspects	Considering the macro implications of digital transformation in healthcare from an economic and social perspective.

Table 1.1: Main areas of research in healthcare digital transformation, adapted from [KSPI21].

a variety of societal factors, including the sensitivity of the personal medical data being handled, the need for high reliability and availability of services, and the complexity of the domain itself, which involves a wide range of stakeholders and professionals. This makes healthcare generally more conservative in adopting new technologies, making the process of digital transformation generally slower than in other sectors [HRC⁺20]. It is then not surprising that despite the *Industry 4.0* paradigm introduced in the early 2010s [LFK⁺14] and progressively adopted in many sectors is still far from being realized in healthcare today [KBA⁺22].

Healthcare 4.0 [TFMCV⁺20] is a term that has been used to refer to the application of Industry 4.0 technological drivers in healthcare. These include IoT, Big Data and AI technologies, which ease the collection and analysis of large amounts of data, support the continuous care and remote monitoring of patients and the support of clinical decisions through data-driven approaches. The general consensus is that Healthcare 4.0 represents a frontier for innovation in healthcare towards the virtualization and distribution of real-time healthcare services [KBA⁺22].

The survey by Kraus et al. [KSPI21] identifies five main areas of research in the

context of healthcare digital transformation summarized in Table 1.1. The survey highlights that healthcare digitalization is a complex phenomenon that is studied across multiple axes. The role of participatory approaches to the design of digital solution is also highlighted as a key factor for the success of digital transformation initiatives, as they can help break the barriers to adoption and acceptance of new technologies by both patients and healthcare professionals.

The first two areas of research in Table 1.1 focus on the direct impact of digital technologies on patients and efficiency of healthcare processes. This is confirmed in other surveys as well [AJMA20, TFMCV⁺20] which identifies the improvement of *quality of care* and *operational efficiency* as the main goals of Healthcare 4.0.

Namely, improving the quality of care is proposed to be achieved through improved access to patient’s past health records together with general lifestyle data to design personalized treatment plans, real-time (remote and on-site) monitoring, and improved utilization of resources and scheduling of appointments. Efficiency is instead targeted through proactive maintenance of equipment, automation of repetitive tasks, and support decision-making thanks to the analysis of data shared across different systems and sources to detect patterns and trends in advance. The analysis maps out several applications of Healthcare 4.0 targeting patients, healthcare professionals, medical equipment and resource management across the whole healthcare organization [AJMA20], confirming the wide scope of digital transformation in healthcare.

The pervasive scope of Healthcare 4.0 leads to several challenges that span from technical to organizational and socio-economic aspects. Table 1.2 summarizes the main challenges identified in the survey by Kotzias et al. [KBA⁺22]. Among the main technical challenges, issues related to data privacy and security, as well as data fragmentation and lack of standardization are the most relevant. These challenges stem from the heterogeneity of systems and the insufficient incentives for data sharing across organizations. Addressing these challenges require an integrated approach. On the technical side, the push towards interoperability and data management and integration is crucial to support all other applications relying on it. Al-Jaroodi et al. [AJMA20] propose to tackle the technical challenges through a service-oriented middleware approach. Recognizing that building healthcare services from the ground up is not feasible due to practical and economic constraints,

Challenge	Description
Data privacy and security	Medical data is sensitive and must be protected from unauthorized access complying with rules and regulations.
Data fragmentation	Medical data is often heterogeneous, complex and uncertain, often unstructured and hence difficult to integrate in a coherent picture.
Legal framework	The legal framework surrounding healthcare data is often fragmented and varies by jurisdiction. Ethical considerations must also be taken into account.
Closer Collaboration	There is very limited data sharing between organizations, and often it is shared by manual processes and limited by legal constraints.
Knowledge and skills gap	Healthcare professionals often lack the necessary digital skills and knowledge to effectively understand the impact of new technologies and data-driven approaches. At the same time medical data requires a high level of expertise to interpret and utilize effectively.
Users' acceptance and adaptability	The successful implementation of solutions depends on the acceptance, trust and adaptation to new technologies by both healthcare professionals and patients.
IoT known issues	The integration of IoT devices in healthcare raises concerns regarding data security, interoperability, and the management of resources linked to IoT systems.
Implementation costs	Healthcare 4.0 solutions often requires significant upfront investment in new technologies, infrastructure, and training, with no guarantee of return. This puts a lot of pressure on organizations that already operate at thin margins.
Standardization	The lack of standardized protocols and frameworks for data exchange and interoperability hinders the seamless integration of diverse systems and technologies.

Table 1.2: Challenges in Healthcare 4.0, adapted from [KBA⁺22].

they argue that leveraging existing systems and services through a middleware layer can facilitate interoperability across heterogeneous systems and enable the integration of new technologies and services. Among the main functionalities that such a middleware could provide, they highlight data brokering and translation to handle data heterogeneity.

This thesis looks at the emerging paradigm of Digital Twins (DTs) [SLPD21] as a potential solution to address the technical challenges of Healthcare 4.0, focusing on the role of DTs as enablers of interoperability and integration of heterogeneous systems and data sources in healthcare [AKK⁺23].

1.2 Hospital Information Systems

Hospital Information Systems (HIS) are the most prominent example of ICT systems in healthcare, supporting the management of clinical and administrative data concerning hospital operations. This has been historically one of the first applications of ICT in healthcare, given the relevance of hospital management in the overall operation of healthcare institutions. Hospitals are, in fact, complex organizations that involve a wide range of activities, and the major source of costs.

More recently the term HIS have been evolving to consider *Healthcare Information Systems* to represent a broader scope that includes not only hospital management, but also the overall management of healthcare services across different settings [KG01]. Nevertheless, hospital information systems continue to play a crucial role as the prominent example of healthcare ICT systems and hence why this section refers to HIS as hospital information systems.

Haux in [Hau06] reflects on the evolution of HIS over the years. He identifies the main drivers from the digitalization of healthcare data (compared to paper reporting) and the shift from local systems to interconnected systems within and across organizations. The initial use of ICT to support specific tasks (e.g., radiology, laboratory, administration) weights on the overall design of a HIS, which still often results—due to practical and economic constraints—an integration of several specialized systems rather than a coherent system designed from the ground up.

Other development trends of HIS include the progressive inclusion of patients as users of the system, rather than just passive subjects of care, and the aggrega-

tion of data from multiple sources to support medical research and public health monitoring rather than just clinical care. These additional use cases further increase the complexity of HIS which needs to be designed to support a wide range of users with very different needs. On the technological side, the evolution of HIS is further driven by the adoption of new technologies for continuous health monitoring and the availability of new data sources (e.g., genomics, lifestyle data) which can be integrated to support personalized medicine.

Finally, Haux finds that the digital transformation significantly permeated the organizational management of processes. Strategic management of HIS becomes crucial to ensure that operations run smoothly and that the system can evolve to meet new requirements. Given this historical perspective on the evolution of HIS, Haux suggests that the future evolution of HIS would require interoperability across different systems and organizations with the consequent development of new scalable architectures and new professionals working at the intersection of healthcare and ICT [Hau06].

Similar efforts to analyze the evolution of HIS have been carried out more recently [BK13, Deg14], confirming the overall trends and pushing for the role of standards to address the challenges in HIS. Figure 1.1 shows the main functional subsystems of a HIS. The decomposition identifies three main categories of processes associated with corresponding HIS sub-systems: clinical information, logistics and decision systems [Deg14]. The interrelation of these three areas is crucial as every process in a hospital spans across them, further complicating the design of a HIS.

The design of a HIS is generally component-based [VDVD03, WAH⁺23]. The components of a HIS follow a functional decomposition [WAH⁺23] separating the core functionalities across different processes. Classical components of a HIS include [WAH⁺23]:

- Patient Administration Systems including **Patient Identification, Appointment Scheduling (AS), Admission, Discharge and Transfer (ADT)** and **Billing and Accounting**;
- Medical Documentation Systems such as the **Electronic Health Records (EHRs)** to manage patient medical records, including clinical notes, diag-

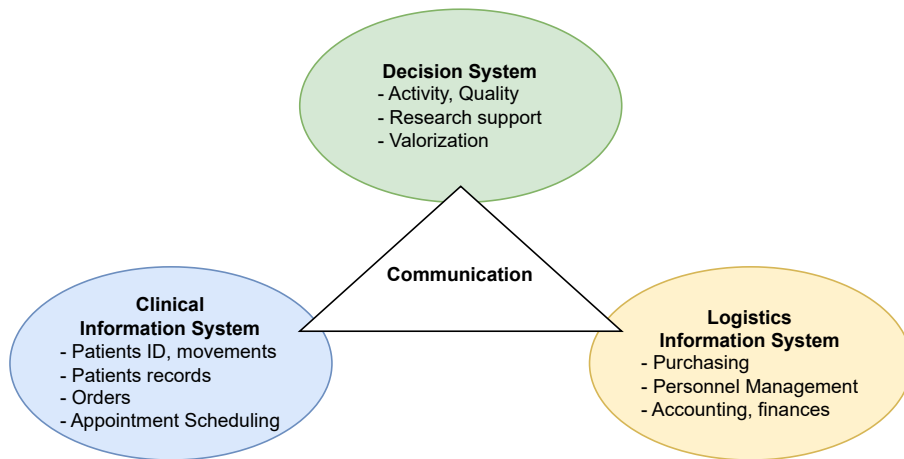


Figure 1.1: Functional subsystems in a Hospital Information System, adapted from [Deg14].

noses, medications, and allergies, now often integrated to provide a full view of patient data through **Clinical Data Repositories (CDR)**;

- **Operation Management Systems** to support logistics such as planning, bed management and staff schedules for surgeries.
- **Computerized Physician Order Entry (CPOE)** to manage orders for medications, lab tests, imaging, and other procedures;
- **Clinical Decision Support Systems (CDSS)** to provide clinicians with evidence-based recommendations and alerts to support clinical decisions;
- ancillary departmental systems such as **Laboratory Information Systems (LIS)** to manage laboratory tests and results, **Radiology Information Systems (RIS)** and **Picture Archiving and Communication Systems (PACS)** to manage imaging studies and results, and **Pharmacy Information Systems (PIS)** to manage medication orders and dispensing;
- **Document Archiving Systems** to store and manage medical documents for archival and legal purposes.

These core modules are often building the backbone of a HIS which can then be correlated with a variety of specific additional systems to support specific needs of the organization. These include telemedicine systems, patient portals, mobile health applications, and systems for managing specific clinical areas such as oncology or cardiology [WAH⁺23].

The level of digitalization and integration of these functional components can significantly impact operations in a healthcare organization. Maturity assessments of HIS are hence crucial to identify gaps and areas of improvement. Several maturity models have been proposed [GR18], assessing the maturity of HIS across various dimensions such as technology, processes, and organizational culture.

A notable example is the *Electronic Medical Record Adoption Model (EMRAM)* by HIMSS [HIM22], which assesses the adoption and utilization of electronic medical records in healthcare organizations across eight stages, from 0 (no electronic records) to 7 (fully electronic and integrated systems). The different levels offer a roadmap for healthcare organization to plan their digital transformation journey. Interestingly, the first levels (following also an historical perspective) focus on the digitalization of ancillary systems first, and follow with the integration of EHR in clinical data repositories, highlighting the importance of data integration as a key step in the digital transformation of healthcare organizations.

Other models such as the one proposed in [CRvdWA19] also identify different levels, characterizing the maturity of HIS in terms of interoperability, data analysis, strategic planning, involved people, availability of electronic medical records, security and technological infrastructure. The goal of the last levels is to achieve a fully national integrated HIS that utilizes real-time data, supports personalized medicine, flexible planning and clinical decision-making, in line with the goals of Healthcare 4.0.

1.3 Medical Standards for Interoperability

The complexity of HIS and the need for integration across different systems make interoperability a key requirement. This motivates the need for standards to ensure that different systems can communicate and exchange data effectively. The complexity of medical data and the variability in medical processes makes the

encoding of medical knowledge in such standards a challenging task.

Over the years, several independent organizations have developed standards to address different aspects of healthcare interoperability [BG16]. Medical interoperability standards can be broadly categorized into different types based on their purpose and scope:

- **Terminologies and Coding Systems** provide a common vocabulary for representing medical concepts, diagnoses, procedures and medications. Examples include International Classification of Diseases (ICD) (established by the World Health Organization) for disease classification, Systematized Nomenclature of Medicine Clinical Terms (SNOMED-CT) for clinical terminology, and Logical Observation Identifiers Names and Codes (LOINC) for laboratory and clinical observations.
- **Data Exchange and Transport** standards define how data is exchanged between systems. One of the main example is Health Level Seven (HL7) Clinical data Architecture and its new version Fast Healthcare Interoperability Resources (FHIR), which defines a structured information model for exchange of medical data across systems.
- **Specialized Document Formats** are used to represent specific types of medical documents. An example is the Digital Imaging and Communications in Medicine (DICOM) standard for medical imaging.

These standards are often used in combination to achieve technical and semantic interoperability. In particular, FHIR is the most modern proposal for data exchange in healthcare systems, adopting a modular approach based on the definition of *resources* that can be combined to represent complex clinical concepts, which can be associated with other standard terminologies [Bra18] FHIR adopts a REpresentational State Transfer (REST) [Fie00] approach to its Application Programming Interface (API), making it easier to integrate with modern Web technologies, breaking out of the traditional silos of healthcare systems. Despite this choice, FHIR is not limited to work in a request-response fashion, but can be integrated as a transport protocol for other communication paradigms such as

publish-subscribe or event-based architectures, or even simply to serialize documents for archival purposes [BG16]. FHIR resources can link to other resources, enabling the representation of relationships between entities (e.g., patients, practitioners, medications, observations). This approach favors the integration of data from multiple sources in a HIS, and the pervasive use of FHIR as a the format to represent and exchange data in a structured format [BG16].

FHIR resources have a hierarchical structure, with each resource having a defined set of attributes and relationships to other resources. The semantic interpretation of FHIR resources is supported by a fixed set of data types, and the support for referencing external terminologies and coding systems, providing unambiguous meaning to the data being exchanged. Furthermore, FHIR provides a mechanism for defining profiles and extensions, allowing customization of resources to meet specific use cases while maintaining interoperability with the core standard.

Due to its versatility and modern design, FHIR is increasingly being adopted in healthcare systems [APA⁺21] and also in healthcare research [VLK⁺22], making it a key standard for healthcare interoperability today.

A different approach to EHR interoperability is taken by the openEHR standards [ope21], which focus on the multi-level definition of stable *reference models* and domain-specific *archetypes* to represent clinical concepts in a structured and reusable way. The main focus of openEHR is on the modeling of clinical data, rather than on the exchange of data between systems. This makes openEHR more suitable for the design of clinical data repositories and EHR systems [DFM⁺24]. The standard further provides guidelines on management of EHR including versioning, auditing and access control which are crucial for the management of sensitive medical data. OpenEHR can hence be considered complementary to other standards, focusing on the management and storage of EHR data [BMMR15, PJGBF⁺23].

This section provides a brief overview of the main standards for healthcare interoperability. The abundance of standards clearly reflects the complexity of the healthcare domain both in terms of the intrinsic complexity of medical data and processes and the variety of stakeholders and organizations involved. Adopting and combining effectively these standards is a key requirement for the design of modern HIS towards the implementation of Healthcare 4.0 objectives. Semantic

interoperability is still an open challenge as the standards partially overlap and are not supported by formal ontological models which could remove ambiguities and improve deductive reasoning [dMRdC⁺22]. In this sense, alignment with Semantic Web technologies [BLHL23] and ontologies could provide a pathway towards achieving greater semantic interoperability in healthcare [SMC13].

1.4 Intelligent Applications in Healthcare

Among the drivers of Healthcare 4.0, AI is playing an increasingly important role in healthcare. Due to its inherent complexity, healthcare has always been a target domain for AI applications, as the possibility to ease the burden of healthcare professionals and improve the quality of care is a strong motivation for innovation. While originally most AI applications in healthcare focused on rule-based symbolic reasoning and expert decision-support systems to support clinical decisions, the recent advantages in Machine Learning (ML) and Deep Learning (DL) and the availability of large amounts of data have made data-driven approaches more feasible [YBK18]. Recent surveys confirm that the rate of scientific publications on AI in healthcare is growing at a very fast pace covering a wide range of applications [SCS⁺21, YDL⁺25].

Applications of AI techniques in healthcare span across a wide range of areas. One of the most significant is image-based diagnosis, where Convolutional Neural Networks (CNNs) for image classification have shown performances comparable or even superior to human experts in recognizing pathologies from medical images. Other relevant applications include genome interpretation, biomarker discovery, risk estimation and prediction of clinical outcomes, and continuous monitoring through IoT stream analysis [YBK18].

More recently, the advent of Large Language Models (LLMs) and Generative Artificial Intelligence (GenAI) technologies is opening new possibilities for AI applications in healthcare [SGS⁺24]. GenAI techniques can be used for data augmentation, to generate synthetic medical data and support training of ML models when real data is scarce. LLMs can be used to analyze and produce medical documents and their adoption is growing at a fast pace especially for the automatic draft of clinical documentation from conversations with the overall objective of

increasing efficiency by automating repetitive tasks [PLR⁺25]. Additionally, LLM can be used to interact with patients through chatbots, providing information and support for common medical questions and concerns. Fine-tuned medical LLMs can also be used to support clinical decision-making and summarization of medical literature [SGS⁺24].

Secinaro et al. [SCS⁺21] identify three main clusters of application of AI in healthcare:

- predictive models for diagnosis and monitoring of patients
- decision support systems to support decision-making
- big data analytics to optimize healthcare operations and management

This multi-faceted approach highlights the wide scope of AI applications in healthcare which can target both clinical and administrative processes.

Given the critical nature of healthcare, the adoption of AI requires careful consideration of ethical implications [MMB⁺20]. In general AI systems are seen as tools to improve human decision-making, making it possibly more objective and data driven. Of course this requires carefully designed systems and data collection processes to avoid introducing new sources of errors and biases which may undermine the benefits [NHA⁺21].

The reliance on AI systems for clinical decisions can also raise concerns of misdiagnosis due to the possibility of overestimating the confidence of automated systems [MMB⁺20]. New trends are hence focusing on explainability and interpretability of AI systems which becomes a crucial property for the deployment of AI systems in healthcare [BMP24]. Explainable Artificial Intelligence (XAI) techniques are being increasingly required in healthcare [FGL24] settings due to the need for transparency in the decision-making process which is crucial for building trust: a necessary condition for the successful acceptance of AI by both patients and healthcare professionals [AAOL25].

Finally, regulations play a crucial role in the effort to ensure that AI systems in healthcare are safe and ethical [FGL24]. These need to be accompanied by guidelines and best practices that promote careful design and implementation of

AI models, from data collection, selection and preprocessing, to model training and validation [dHLH⁺22].

1.5 The AUSL Romagna Context

Azienda Unità Sanitaria Locale della Romagna is the local public healthcare administration for the Emilia-Romagna region in Italy, covering a population of more than 1.1 million people across a territory of roughly 5100 km² which covers the provinces of Ravenna, Forlì-Cesena and Rimini and is divided into 8 health districts (Figure 1.2). The organization manages 7 main hospitals hubs comprising a total of 13 hospital facilities and several other care centers managed by a network of more than 16 thousands employees. Born from the merger of three different local healthcare administrations in 2013, AUSL Romagna is among the largest healthcare providers in Italy. The high complexity of the organization and the large territory it covers make AUSL Romagna a relevant case study for the challenges of digital transformation in healthcare as, due to its historical evolution it deals with integration of heterogeneous systems and processes across different settings and can represent a microcosm of the challenges faced by the healthcare sector in general.

AUSL Romagna, aligned with the objectives of the national healthcare plans, is pursuing a digital transformation process aimed at improving the efficiency and effectiveness of its services. A first essential step in this direction has been the adoption of a personal EHR system directly accessible by patients (*Fascicolo Sanitario Elettronico (FSE)*). This system allows patients to access their health records, book appointments, and communicate with healthcare providers more easily. Emilia-Romagna has been the Italian region with the highest success rate in the implementation of FSE [fse] with over 65% of the population having accessed their FSE at least once and more than 90% of the population agreeing to its consultation by healthcare professionals.

The digitalization program has been accelerated by the COVID-19 pandemic, which has highlighted the need for more agile and resilient healthcare systems worldwide. The NextGenerationEU programme of the European Union funds several initiatives in its member states as a response to the COVID-19 pandemic. In

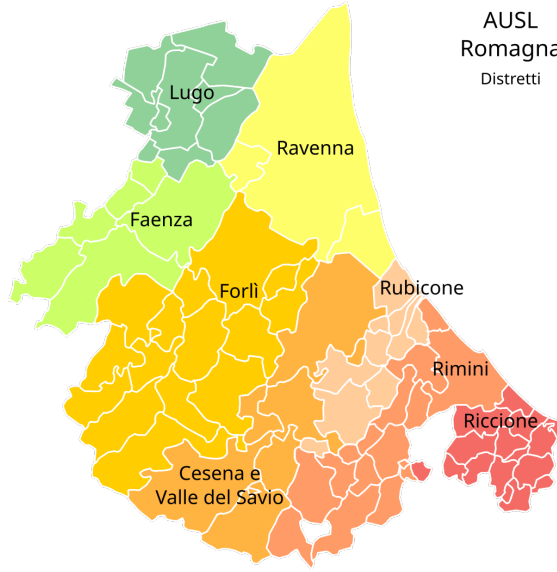


Figure 1.2: Map of the AUSL Romagna territory, credits: WikiMedia¹

Italy, this has been translated into the *Piano Nazionale di Ripresa e Resilienza (PNRR)* which includes several *missions*. Mission 1 includes as a key objective the which digitalization of public administrations, including healthcare organizations. This activity is pushing for the adoption of digital technologies to improve citizen-oriented services. The national plan for public administration digitalization [Age25] includes several guidelines and targets towards increasing the fruition of digital services, cloud technologies, AI and open data. Mission 6 is instead entirely dedicated to the innovation of healthcare services, targeting the technological improvement of hospitals, decentralization of healthcare services, telemedicine and remote care [Gov21]. This PhD work is also funded under PNRR, under Mission 4 promoting the collaboration of industrial partners and research institutions.

AUSL Romagna is continuing its digital transformation journey within this framework. A strategic plan for its digitalization (“Progetto Sanità Digitale”) has been defined in 2021 to guide the process as a collaboration between AUSL Romagna, the Computer Science and Engineering Department of the University of Bologna and the oncologic care and research center Istituto Romagnolo per lo Studio dei Tumori “Dino Amadori”(IRST), members of a local laboratory for health-

¹https://commons.wikimedia.org/wiki/File:Mappa_dell%27AUSL_Romagna.svg

care innovation (Laboratorio Sanità Digitale²) [Ric22]. Key objectives of the digitalization plan include the decentralization and distribution of care with local care centers and remote monitoring, the integration of data from multiple sources to support personalized medicine and the improvement of operational efficiency. A strategic technology envisioned in this context is the adoption of *DTs* [Gri23], which can offer an aggregation point for data and functionalities from multiple systems tailored to the representation of critical assets in the healthcare organization [Ric22].

This thesis work is framed within this context, exploring the role of DTs as enablers of Healthcare 4.0 objectives and focusing on the challenges of interoperability and data integration in a complex and distributed healthcare organization such as AUSL Romagna.

1.6 Final Remarks

This chapter provides an overview of Healthcare 4.0 which motivates and frames the work presented in this thesis. The digital transformation of healthcare is a complex process that involve multiple dimensions, from technological to organizational and socio-economic factors.

The chapter highlights the multi-faceted nature of the healthcare domain, and the efforts that are being pursued towards improving digitalization, with a focus on HIS as the exemplary case of ICT systems in healthcare which motivates the need for interoperability and data integration.

The chapter further reflects on the role of AI in healthcare, showing how it can address both clinical and organizational challenges, emphasizing the critical requirements the domain imposes on the design of intelligent applications that are trustworthy, reliable and fair.

Finally, the context of AUSL Romagna as the partner organization for this thesis work is presented, showcasing the relevance of the challenges faced by the organization in its digital transformation journey.

²<https://laboratorio-sanita-digitale.github.io/>

Chapter 2

Digital Twins

This chapter provides an overview of the DT concept, its historical development and the more recent models and technologies that have emerged to support its implementation. The ever-growing literature on the subject reflects the increasing interest in DTs and its multi-faceted nature, encompassing various domains and applications. This thesis focuses on the software engineering perspective of DTs, and the application of DTs to the engineering of IoT systems. Accordingly, this chapter focuses on how the DT concept has been interpreted in this context.

2.1 History and Definitions

The concept of DT was introduced in the early 2000s by Michael Grieves in the context of Product-Lifecycle Management [Gri23] where it was conceived as a virtual representation of a product throughout its lifecycle. At its essence, Grieves defined what later came to be known as a DT as a system composed of three main components (Figure 2.1):

- a *physical space* and its products;
- a *virtual space* containing the digital representation of the products;
- a *connection* between the two, making data flow from the physical to the virtual space and information flow from the virtual to the physical space [GV17].

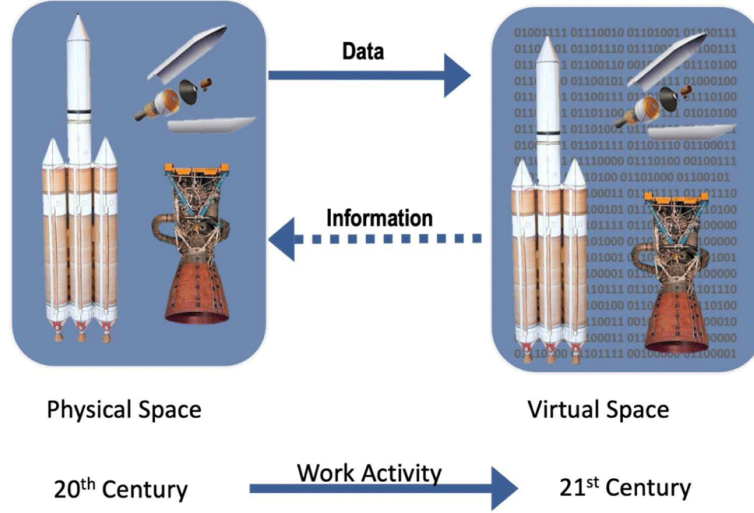


Figure 2.1: The original DT conceptual model by Grieves, from [Gri22].

Grieves referred to this concept as the *Mirrored Spaces Model* [Gri05], a name which echoes the idea of *Mirror Worlds* introduced by David Gelernter in the 1990s [Gel91] who also envisioned the ability to replicate the real world in a completely virtual space [SFH⁺21].

The idea was later associated with its modern *Digital Twin* name and popularized by NASA in the 2010s, when it was presented as a key technology for future development of aircraft and spacecraft in order to virtually simulate extreme conditions, integrate data from the physical system in operation, and provide feedback on the conditions of the *flying twin* while possibly enacting changes to mitigate damage [GS12]. NASA’s definition hence focused on the ability of the DT to integrate multiple simulation models and characterized the DT as *ultra-realistic*, as the goal was to virtually replicate the physical system with the highest possible degree of fidelity:

A Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin [GS12].

Given the various phases of a product’s lifecycle, the DT concept was later

refined to distinguish between the *Digital Twin Prototype* (DTP) and the *Digital Twin Instance* (DTI) and *Digital Twin Aggregate* (DTA) [GV17]. The DTP is the virtual representation of the product during its design and development phase, while the DTI is the virtual representation of the product during its operational phase. The DTA represents instead the aggregation of all DTIs, hence all products that have been built [Gri22]. Interestingly, the DTP exists before the physical product is even produced, and it is rather “just” a virtual model of the product. Grieves argues that requiring that a DT exists only when the physical product exists is a *fallacy* [Gri22]. Nevertheless, it is generally accepted within the community nowadays that, for a proper characterization, what is usually referred as DT is hence a DTI, which exists alongside its physical counterpart and is continuously updated with data from the physical world.

The role of such bidirectional data exchange between the physical and virtual parts of a DT has been, in fact, used to characterize the DT concept and distinguish it from other related concepts. A widely accepted taxonomy classifies the concepts of *Digital Model*, *Digital Shadow* and *Digital Twin* based on direction of the automatic data flow between the physical and virtual spaces [KKT⁺18] (Figure 2.2): namely, a Digital Model (Figure 2.2a) is a static model of a physical system (similarly to the DTP) that can be manually updated over time, the Digital Shadow (Figure 2.2b) gets automatically updated through an inbound data flow from the physical space, while the Digital Twin (Figure 2.2c) is the only one that holds a bidirectional data flow which can provide feedback to the physical counterpart. Such connection has been referred to as the *digital thread* [SW18, Gri23] to evoke the idea of a tie that binds the physical and virtual spaces. Other common terms include *twinning* [JSN⁺20] or *shadowing* [JYL⁺21, RCM⁺22a] process. This terminology emphasizes the continuous and dynamic nature of the relationship between the physical and virtual entities, and highlights the active role of such process in maintaining the DT up to date.

The scope and applicability of the DT concept has evolved and expanded to encompass a wide range of applications and domains. Today, DTs are used in various fields such as manufacturing, healthcare, smart cities, and more. This is also reflected in the abundance of terminology used to identify the physical counterpart of a DT which is nowadays often referred to as the *physical twin*

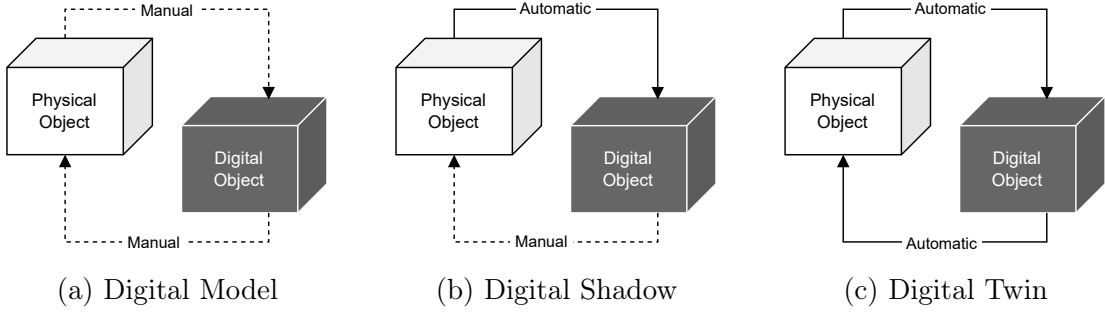


Figure 2.2: Taxonomy of Digital Twin related concepts based on the data flows between physical and digital objects, adapted from [KKT⁺18].

or *physical entity* [SFH⁺21, JSN⁺20, DJR⁺22]. This shift from identifying the physical counterpart as a *product* to a more generic *entity* highlights the broader applicability of the DT concept beyond its original manufacturing context. The DT concept has been adopted also to represent people [She21], processes (e.g., supply chain [BBKY20]) and organizations [PLT20], leading to a more abstract interpretation of what can be considered *physical*. Accordingly, the definitions of DTs have also diversified, leading to a plethora of interpretations [DJR⁺22].

For the scope of this thesis, since the focus is on how DTs can be used to engineer IoT systems, taking as the reference context the healthcare domain, the following definition is adopted, starting from the one in [MLC20]:

A Digital Twin (DT) is a comprehensive software representation of an individual Physical Asset. It includes the properties, conditions, and behavior(s) of the real-life asset through models and data. A DT is a set of realistic models that can simulate an asset's behavior in the deployed environment. The DT represents and reflects its physical twin and remains its virtual counterpart across the asset's entire lifecycle.

The definition emphasizes the software nature of the DT and its ability to model the properties and behavior of its physical counterpart. The term *Physical Asset (PA)* is deliberately used to refer to the physical counterpart of a DT to highlight the fact that an *asset* is something that has a strategic value in the context of an application domain, and that the DT is meant to represent and manage such assets. Compared to the original Grieves model which characterized the DT as

having three parts (physical, digital and connection), the remainder of this thesis will instead consider:

- the PA as the physical counterpart of a DT;
- the DT as the software implementing the virtual representation of the PA through a combination of models and data;
- the *twinning process* implemented by the DT software to keep the DT up to date with the PA and possibly provide feedback to it.

2.2 Reference Architectures and Properties

The conceptual definition of DT gives little indication on how to implement it, leaving a degree of freedom in the technical realization of the DT software. For this reason, a variety of reference architectures have been proposed, each emphasizing different aspects of the DT concept [FBB22].

One that has gained significant recognition in the DT community is the *five-dimensional model* (5D model) by Tao et al. [TZLN18]. As shown in Figure 2.3, the 5D model characterizes a DT as composed of five main components [QTH⁺21]:

- the *physical entity* (PE), as the entity in the physical world that is being represented by the DT (i.e., what is identified as PA in this thesis);
- the *virtual models* (VM), which is the digital representation of the PE possibly including 3D models, rules and behavioral models;
- the *data* (DD), which includes all data related to the PE and possibly generated by the VM, such as historical data, real-time data, simulation results;
- the *service* (Ss), which encompasses all services provided by the DT to users, such as monitoring, simulation, diagnostics, and optimization;
- the *connection* (CN), which represents the communication and data exchange between all other components, there including the bidirectional connection between the PE and the VM that is fundamental for the DT operation.

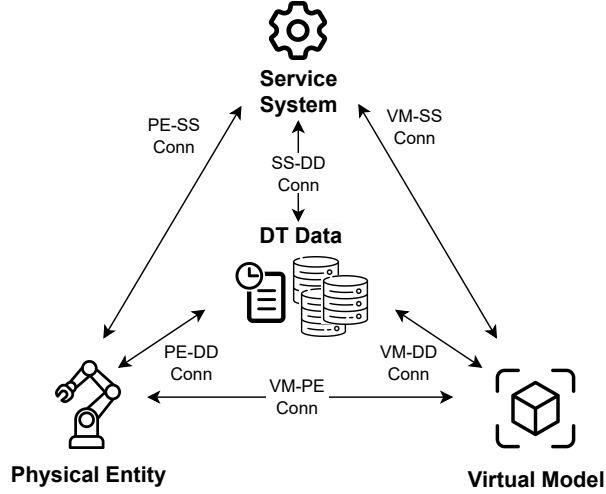


Figure 2.3: Conceptual representation of the 5D model of a Digital Twin, showing the five dimensions and the connections between them.

Tao’s 5D model extends Grieves’ original model by explicitly including the data and services dimensions, better characterizing the DT as a data-driven and service-oriented system.

The service-oriented nature of DTs has an essential role in their ability to provide value to users. The DT is hence not just a passive representation of a PA, but an active system that users can interact with to obtain insights and perform actions on the PA. This also contrasts the idea of the PA-DT system as a closed-loop control system, but rather envisions the DT as open to interactions with external entities such as human users or other software systems.

Following this perspective, the idea of DT-as-a-service has been proposed. As shown in Figure 2.4, the proposed reference architecture adds a cyber layer and an application layer on top of the classic physical, digital and communication layers of a DT [AXZL21]. The reference architecture further highlights different levels of integration following the taxonomy of Digital Model, Shadow, and Twin [KKT⁺18], with a further Digital Twin predictive level, which includes predictive models and analytics capabilities enabled by the cyber layer grounded on cloud technologies such as Big Data analytics. The vision additionally integrates DTs in an incremental and iterative development lifecycle, to accompany and evolve alongside the PAs they represent.

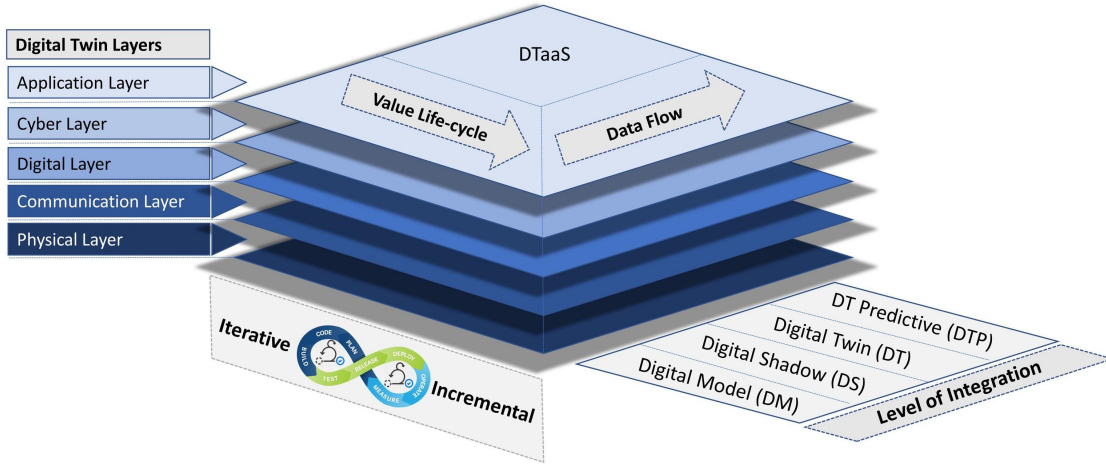


Figure 2.4: The DT-as-a-Service reference architecture, from [AXZL21].

Layered and service-oriented architectures for DTs are the most popular in the literature. These architectures support desired non-functional quality attributes such as performance efficiency, reliability and maintainability, but also compatibility with existing systems and scalability [FBB22]

On the functional side, [MLC20] proposes a set of properties DT should have, analyzing existing literature and implementations in the IoT context. The properties, summarized in Table 2.1 detail various aspects that characterize functional behavior of DTs. Interestingly, such properties go beyond the generic definition of DT as a virtual representation of a PA and highlights the various capabilities that a DT can provide to its users.

First, both the DT and the PA are required to be *univocally identifiable* in order to establish a clear relationship between the two. This vision supports the possibility of having possibly multiple DTs representing the same PA in different contexts or for different purposes. Accordingly, the second important contribution is the emphasis on *contextualization*. This shifts the focus from the ultra-realistic DT envisioned by NASA to a more pragmatic approach where the DT is representative of the PA in a specific context, hence possibly abstracting away details of the PA that are not relevant for the intended use of the DT in the target system. Additional properties emphasize the dynamic nature of the DT and its ability to reflect changes in the PA (*reflection*), in a timely manner (*entanglement*), persisting over time (*persistence*), memorizing past states (*memorization*) and offering

Table 2.1: Characterizing Properties of DTs, defined in [MLC20].

Property	Description
Contextualization	A DT models the PA in a way that is representative with regard to the target context.
Reflection	A DT reflects changes in the PA in real-time.
Replication	A DT replicates an object into different environments.
Entanglement	The degree to which a DT is interconnected with its PA.
Persistency	A DT persists over time even when the PA is not available.
Memorization	A DT stores and allows retrieving past states and events.
Composability	A DT can aggregate different assets or DTs.
Accountability	A DT recovers from errors and maintains a reliable state.
Augmentation	A DT can add new capabilities to the PA.
Ownership	A DT manages access control over its data and functionalities.
Servitization	A DT provides services to its users.
Predictability	A DT can anticipate future states and behaviors of the PA.

a reliable representation of the PA (*accountability*). Finally, the properties also highlight the ability of DTs to provide value to users through services (*servitization*) such as aggregation of multiple assets (*composability*), augmentation of the PA capabilities (*augmentation*), management of access control (*ownership*), and anticipation of future states and behaviors of the PA (*predictability*).

2.3 Enabling Technologies and Platforms

On a practical level, the implementation of DTs has been enabled by the convergence of various technologies. This includes the proliferation of IoT devices, advancements in Big Data analytics and AI, and an increased availability of computing power to run simulations, which make possible to collect and analyze large

amounts of data from the physical world, derive models of the PA and execute them to obtain insights [QTH⁺21, FFDB20, MYH⁺22].

On the physical side, the availability of network-connected sensors and actuators enable the acquisition of real-time data from the PA. Sensing technologies for DT broadly include classic IoT techniques such as RFID, digital sensors, but also computer vision and image processing. In many cases, the data that can be automatically collected from the PA is complemented by manual data entry, making the DT interact with other information systems. Sensing is followed by connectivity and data exchange, through a variety of communication protocols and network technologies. DTs may need to operate with different data sources, each with its own communication protocol and data format. Classic examples of IoT communication protocols include MQTT, CoAP, HTTP, and WebSocket, but also industrial protocols such as OPC-UA and Modbus [FS23]. Finally, the physical layer includes a computing infrastructure that can span from the edge to the cloud, to support the data acquisition, processing and storage needs of the DT.

On the virtual side, the DT requires modeling techniques to represent its properties and behaviors. Depending on the application domain this can range from geometrical and physical models, to behavioral models such as finite state machines or process models. Models can be manually created by experts, or automatically derived from data through data-driven modeling and machine learning techniques [TXQ⁺22]. Models can be used to simulate the behavior of the PA with different simulation software. The volume and variety of data that can be collected from the PA and from the DT models requires Big Data technologies to extract insights [TCQ⁺18]. Finally, the DT software needs to provide services to its users, including data visualization technologies, user interfaces and APIs to interact with other software systems. Some further include Augmented Reality (AR) and Virtual Reality (VR) technologies in the enabling technologies of DTs to provide immersive experiences to users and visualize DT [MYH⁺22].

Due to the complexity of the technological stack required to implement DTs, the industry has seen the emergence of various supporting tools. First examples of DT development tools were developed by large industrial companies such as General Electric's Predix, Siemens MindSphere, PTC ThingWorx, and IBM Watson IoT [FFDB20, AKN20]. Additionally, several simulation tools such as ANSYS

Twin Builder [Ans25], SimuLink [Mat24] and AnyLogic [Any18] started to include features to support DT development. Many of these tools, emerging from industrial domains, are often domain-specific and provide strong integration with the physical assets and systems produced by the same companies. These solutions are tailored to the specific needs of their respective industries, offering robust capabilities for managing and interacting with physical entities. Additionally, being proprietary software, they often lack openness and interoperability becoming closed silos from a broader system engineering perspective as the competitive nature of the market tends to prefer both hardware and software lock-in instead of promoting integration.

More recently, cloud providers have started to offer so-called *DT platforms*. Prominent examples are Microsoft’s Azure Digital Twins¹ and Amazon’s AWS IoT TwinMaker². These general-purpose cloud platforms provide an integration layer on top of existing cloud data storage and IoT middlewares and hence mainly function as data repositories, storing data from physical assets, with pre-processing handled externally. They complement the existing cloud-based service ecosystem with specific features to support the modeling of DTs. This includes the ability to model assets with defined data schemas, providing identification and tracking state properties, as well as tools to integrate data from various sources, and to grant access to the DT data and functionalities through well-defined common APIs. In such environments, the interaction with the physical world is typically managed using an IoT middleware that collects data and routes it to internal communication buses. These buses are consumed by serverless functions that implement the DT model. Additionally, such cloud platforms often provide digital communication capabilities with built-in visualization, interfaces, and query support. Data acquisition pipelines, supported protocols and data formats are often defined at the platform level, not on a per-DT basis, leading to limitations in defining clear boundaries for the DT digitalization process and client interactions.

Interestingly, though, the broader scope of cloud platforms put forth the idea of having multiple DTs connected together to represent a complex application domain. As an example, Azure Digital Twins is based on the concept of a *digital*

¹<https://azure.microsoft.com/en-us/services/digital-twin/>

²<https://aws.amazon.com/iot-twinmaker/>

twin graph where multiple DTs can be connected together to represent complex systems [Mei22]. Additionally, these platforms are not tied to specific application domains, paving the way for a broader adoption of the DT concept.

Finally, open-source DT platforms are emerging [GMGL24a]. Among them, Eclipse Ditto³ is a notable example, integrating with the Eclipse IoT ecosystem. The platform allows storing and managing digital representations of PAs, providing a RESTful API to interact with them. Other relevant examples from the academic community include the DT-as-a-Service platform proposed in [TGM⁺23] supporting developers in developing and sharing DTs by managing a combination of services including simulation models and data sources, and TwinBase [AST21], a platform to manage DT models in the open Web with an implementation leveraging GitHub repositories to store and serve static DT models.

The shift towards more general and open DT platforms is reflecting the broader application of DTs beyond their original industrial context, as well as the need of integrating DTs in larger software ecosystems. Section 7.3 will present an analysis of the positioning of this thesis' contribution with respect to the state of the art in DT platforms.

2.4 Digital Twins and Artificial Intelligence

As highlighted in the previous section, despite not being considered a core requirement when DTs were initially introduced, AI and machine learning techniques are increasingly being recognized as one of the fundamental enabling technologies for DTs [KPZ24]. This is due to the fact that the amount of data that can be collected from DTs is often too large and complex to be manually analyzed and interpreted. Hence, AI techniques can be used to implement different DT functionalities.

In [MCFA23], the authors propose a classification of DT based on the level of behavioral intelligence, correlating the different levels to different AI techniques as shown in Figure 2.5. They introduce four levels of DT intelligence:

- **Passive** if the DT purpose is only to provide a virtual representation of the PA for monitoring and simulation purposes;

³<https://eclipse.dev/ditto/>

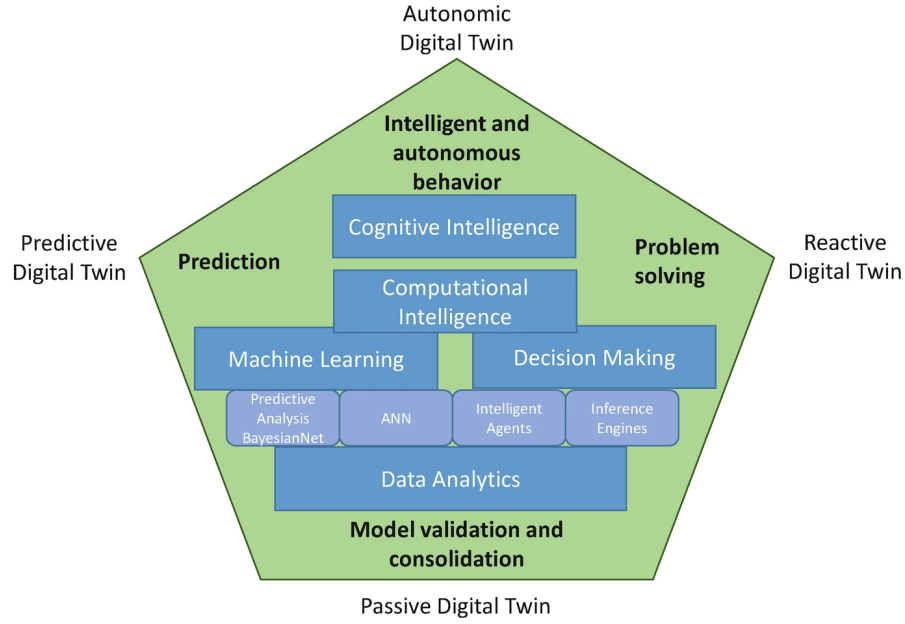


Figure 2.5: Artificial intelligence with respect to DT capabilities, from [MCFA23].

- **Predictive** if the DT can forecast future PA states and behaviors based on historical data and models;
- **Reactive** if the DT can respond to changes in the PA state by diagnosing issues and taking corrective actions using reasoning techniques;
- **Proactive or Autonomic** if the DT can understand the PA context and make decisions adapting to it to autonomously reach or maintain goals without human intervention.

Passive DTs can be supported by statistical data analytics techniques to create models of the PA. Predictive DTs can leverage different techniques such as regression, time series analysis, deep learning but also fuzzy logic to implement the prediction capabilities. This of course requires the availability of historical data to train such models. Reactive DTs can instead use classification techniques, decision-trees and rule-based systems to implement decision-making. Finally, Autonomic DTs can combine different reasoning techniques with planning to implement self-adaptation and autonomous behavior. This level of intelligence is often associated with the vision of *Cognitive Digital Twins* [ZLK22, IAPGB⁺21] in which the DT

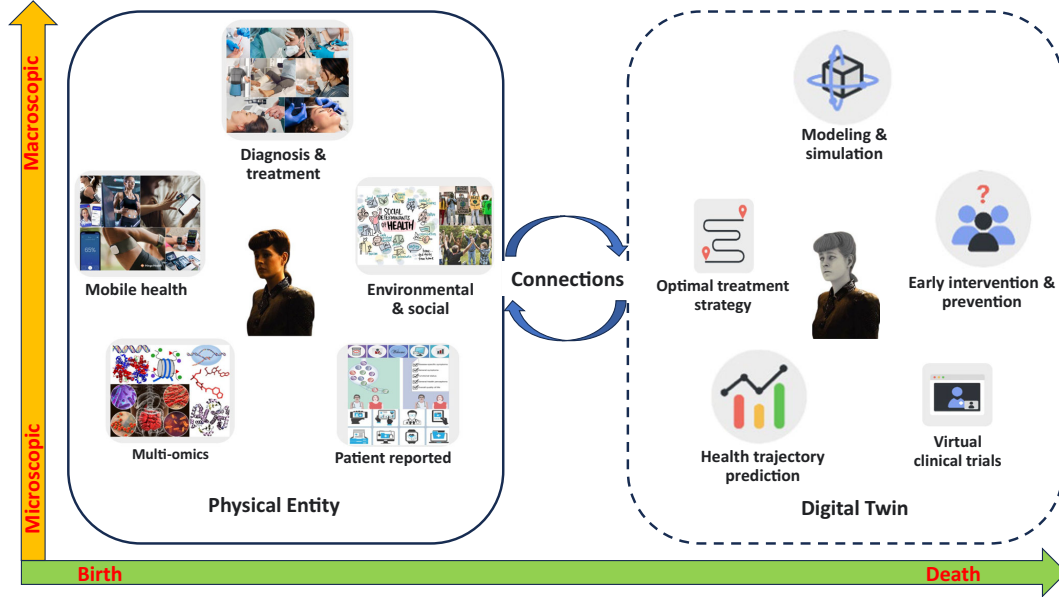


Figure 2.6: An overview of DT for healthcare, showing technological enablers, target goals and the main axis that are relevant for healthcare DTs, namely time and scale. Image from [KWW⁺24].

has a strong degree of autonomy with respect to its PA and can influence its behavior to achieve specific goals. The technologies tied to this vision include knowledge representation, ontologies and reasoning, hence including the more symbolic side of AI techniques, rather than only relying on data-driven techniques.

2.5 Digital Twins in Healthcare

DTs have been used to tackle the tracking, prediction and simulation of physical assets in several domains, and in healthcare it is considered a true revolution [EMD20]. DTs are seen as a key technology to implement the vision of *Healthcare 4.0* [AKK⁺23] (see 1), as they align with the core principles of this new paradigm, promoting digitalization, data integration and proactive and personalized care [BBE⁺19].

DTs can hence be used in various specific healthcare applications, with the common goal of improving the quality and efficiency of healthcare services. Figure 2.6 shows a conceptualization of DT for healthcare, highlighting how the DT can have

a role in various healthcare applications. This is supported by an ever rising trend of scientific publications on the integration of DTs in healthcare [KWW⁺24].

Among the main applications of DTs in healthcare, the ability to create a *Human DTs* is one of the most sought after [She21]. The vision is that each person could have a DT that holds all its relevant healthcare-related data, including biological, genetic and lifestyle information to provide a comprehensive view of a patient's health status over its entire life. The vision supports the idea of personalized medicine, where treatments and interventions are evaluated first on the *virtual patient* to assess their effectiveness and possible side effects before being applied to the real patient [KBZ21, BBE⁺19]. This has been also extended towards processes of drug discovery and development, simulating biochemical reactions and drugs interactions [KWW⁺24].

Other applications focus instead of building DTs of specific organs to better model their behavior and diagnose diseases or detect anomalies [EMD20]. A primary example of this application is cardiology, as creating a DT of the human heart is a complex task with great potential to improve the care of heart pathologies, requiring the combination of mechanical and data-driven models [CAMM⁺20]. Such DTs can be used to understand the medical data collected by continuous monitoring devices for IoT-based remote healthcare [EAG21].

Moving on from the more ambitious clinical applications of DTs in healthcare, there is also potential for their use in operational contexts, such as optimizing hospital workflows and resource management. DTs of hospitals [PZY⁺20] can model patient flows, support allocation of resources and staff as well as support classic building management tasks such as energy optimization and maintenance of critical systems. Similarly, DT of medical devices can be used to monitor their status and predict maintenance needs, ensuring availability [PZY⁺20].

Finally, DTs can be also used to model healthcare processes, such as patient care pathways and clinical workflows [RCM22b]. This can help to track the ongoing processes, automatically produce medical documentation and identify bottlenecks and inefficiencies to improve the overall quality of care and optimize future planning of activities and investments [RCM22b].

Overall, the integration of DTs in healthcare holds great promise, despite the field being still in its infancy and heavily focused on the creation of simulation

models rather than fully connected DTs. Significant challenges remain to be addressed, starting from the typical challenges of healthcare systems such as data quality, integration, privacy and security, but also the need of validation of DT models to ensure their reliability and accuracy for clinical applications [KWW⁺24].

This thesis focuses on the use of DT to model healthcare processes and workflows, as a more pragmatic approach to leverage the potential of DT in healthcare, while addressing the needs of Healthcare 4.0 healthcare systems.

2.6 Digital Twin Interoperability

The rapid development of DT technologies and the adoption of DTs in various domains has led to a fragmentation of the field. Moving on from the technological silos of the early DT implementations towards a more service-oriented view of DTs, it is not surprising that interoperability is emerging as a key challenge for DTs [Pir21, RM24].

Interoperability is seen as a necessary condition for DT maturity. In [KAA24] the authors propose a general maturity assessment framework for DTs. Table 2.2 summarizes the proposed levels of maturity.

Authors correlate level one and two to the Digital Model, level three to the Digital Shadow and level four and five to the Digital Twin concepts [KKT⁺18]. Level six, which they identify as the highest level of maturity is tied to interoperability and envisions *Connected Digital Twins* that can interact with each other to provide joint decision-making capabilities.

In this context, interoperability is hence seen as a key enabler for the realization of DT-based systems and integrate the DT concept as a Cyber-Physical System (CPS) engineering paradigm. Accordingly, in [AKP24], the authors survey DT literature to characterize interoperability in CPS and DT systems. Figure 2.7 visually summarizes the identified levels of interoperability in six levels:

1. **Technical Interoperability** is the lowest level and refers to the ability of different systems to communicate and exchange data using common protocols and formats;

2.6. DIGITAL TWIN INTEROPERABILITY

Level	State	Requirement	Enabled potential
1	Replication of assets	Digitization of physical assets and their state at moment of capture (e.g. 2D maps or 3D models)	Awareness of assets, rudimentary decision support
2	Connection	Connect processes and models to static data and metadata of level 1	Realistic simulations and asset planning
3	Synchronization	Enrich with timely data (sensors and other IoT technologies)	Real-time situational awareness and immersive environments
4	Interaction	Two-way data communication and interaction	Remote control of physical assets and processes
5	Automation	Transparent explainable systems with broad control potential	Autonomous operations optimization and self-maintenance
6	Interoperability	Highly linkable systems characterized by a high level of standardization, ontology definition, and semantic modelling	Joint decision-making among various systems, enhanced system of systems performance

Table 2.2: Levels of Digital Twin maturity from [KAA24].

2. **Syntactic Interoperability** refers to the ability of different systems to understand the structure and format of the exchanged data, enabling them to parse and interpret it correctly;
3. **Semantic Interoperability** refers to the ability of different systems to understand the meaning of the exchanged data, enabling them to interpret it consistently;
4. **Pragmatic Interoperability** refers to the ability of systems to understand context of the exchanged data and use it to
5. **Dynamic Interoperability** refers to the ability of different systems to

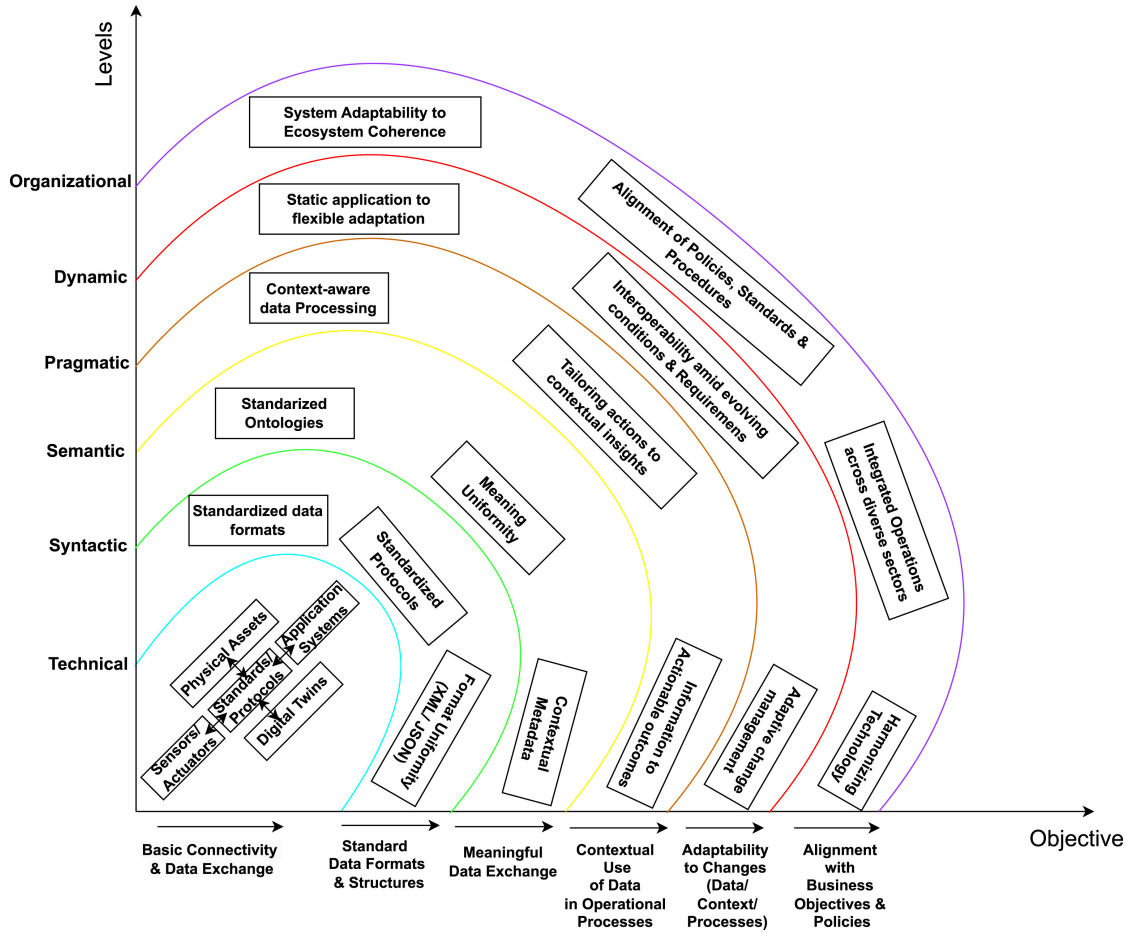


Figure 2.7: Interoperability Levels in Digital Twin, from [AKP24].

adapt and evolve over time, enabling them to maintain interoperability in the face of changing requirements and technologies

6. **Organizational Interoperability** refers to the ability of different organizations to work together effectively, through the adherence to standards and the integration of operations.

These studies highlight the importance of interoperability for the future of DTs, emphasizing the need for standardization and common frameworks to enable seamless integration of DTs in larger systems. The envisioned ultimate goal, tends towards DTs as building-blocks of more complex systems, interacting with each other to provide additional value.

Of course, there are significant challenges to achieve this vision. In [KAA24], the authors identify the lack of trust as a key barrier to interoperability, especially for an organizational level as the exchange of data is often perceived as a loss of competitive advantage. Initiatives such as the European data space [BFRLG21] are trying to address this issue by proposing a federated data infrastructure where data can be shared in a secure and controlled manner.

Other than societal challenges, technical issues such as data silos, proprietary formats, and lack of common standards further complicate interoperability efforts. Standardization initiatives such as the one from European Telecommunications Standards Institute (ETSI) [ETS23] aim to address these challenges by providing guidelines and frameworks for interoperable DTs. Different challenges arise at different levels of interoperability [KAA24].

At the technical level, a plethora of standards originally developed within the industrial IoT domain are being proposed to address the communication and data exchange needs of DTs [Bar24]. Examples are the Asset Administration Shell (AAS) [Pla19], the OPC Unified Architecture (OPC-UA) [MLD09], and AutomationML (AML) [DLPH08]. These standards, given their different origins and motivations, cover different aspects of a DT [Bar24] and are generally tailored to a specific problem, namely AAS focuses on creating a representation of an asset through its static and dynamic properties, operations and services; OPC-UA primarily serves as a secure and efficient Machine to Machine (M2M) communication protocol and data model; and AML captures the geometry, topology, and behavior of machines in automation processes.

At the semantic level, authors recognize the role of shared semantic models to provide common understanding of the exchanged data. In this setting, ontologies and Knowledge Graphs (KGs) are seen as essential tools to implement DTs [KPGD24]. They can provide a common vocabulary and shared understanding of an application domain, fostering interoperability, explainability and reasoning capabilities. The survey finds that most DT implementations use ontologies to represent the physical layer as it is the one more tied to the application domain. Few leverage ontologies to exchange data and build KGs to share the DT knowledge, limiting their interoperability potential [KPGD24].

2.7 Towards Digital Twin Ecosystems

As discussed in the previous sections, DTs are evolving from the original view of virtual copies of physical products, to serve as a general-purpose engineering abstraction for the strategic design and development of CPSs. This shift is reflected by both technological advancements with the emergence of DT platforms that support the management of multiple DTs, and the increasing attention to interoperability to enable DTs to be integrated in larger systems.

Complex CPSs, in fact, typically consist of interrelated assets whose connections can change dynamically. These relationships are currently difficult to track and model with a single DT, hence several approaches are proposing to consider a more collective dimension in which each asset is modeled with its own DT, and the overall system is modeled as an *ecosystem* of interconnected DTs [RCM⁺22a]. *Digital Twin Ecosystem (DTE)*, which are the core concept explored in this thesis, are hence emerging as a key abstraction to address the challenge of effectively model complex CPSs through a set of interconnected DTs. The call to “*make more Digital Twins*” [TQ19] is hence opening a perspective in which DTs are pervasively used to model strategic PAs and their interaction becomes crucial to effectively leverage their potential in complex scenarios.

The idea of DTE can be traced back to the concept of *Digital Twin Aggregate* introduced by Grieves [Gri23] which understood the potential of aggregating multiple DTs instances to derive aggregate insights. In that vision, though, the aggregate was composed of multiple DTs of the same type of asset, to observe the behavior of different instances of the same product and capture collective trends.

The more modern interpretation of DTE can be derived, instead, from the United Kingdom’s *National Digital Twin* program [BBD⁺18]. As described in [Ken21] the National Digital Twin is “*an ecosystem of digital twins and the protocols by which they can be integrated securely and resiliently*”. Finalized at the tracking of the national built environment, the initiative fostered the development of a DT integration framework to share and exchange data across different stakeholders. Although the program was focused on a specific domain, the underlying idea of composing an ecosystem of interconnected DTs rather than a monolithic put forth by the *Gemini principles* [BBD⁺18] can be generalized to other applica-

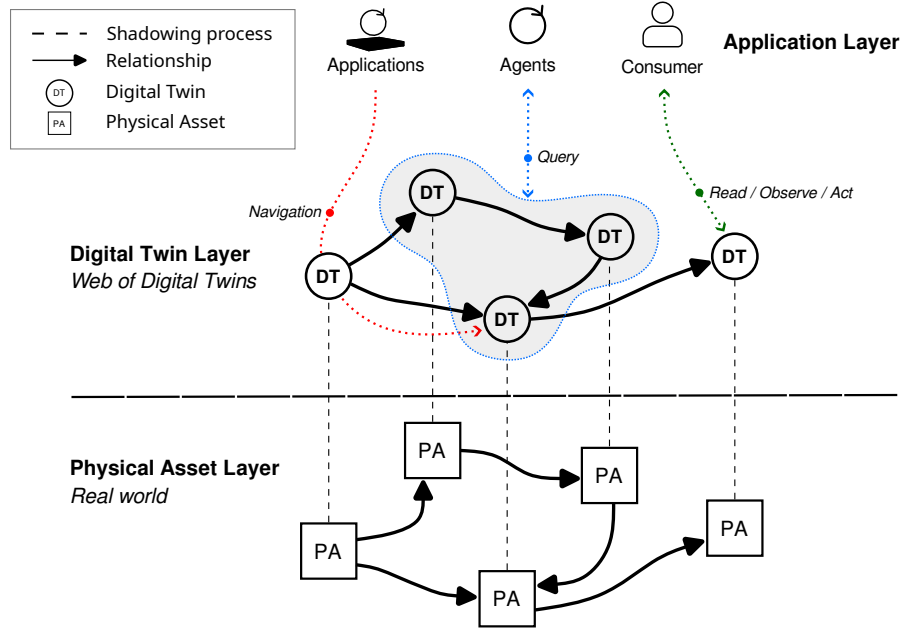


Figure 2.8: Conceptual model of the Web of Digital Twins. Physical Assets (squares) have corresponding Digital Twins (circles) connected through relationships that mirror the ones in the physical world. Applications and users (top) can navigate, query and observe the Web of Digital Twins to access data and services from the interconnected Digital Twins.

tions. Today, DTE are hence envisioned as systems where multiple *different* DTs are interconnected to model complex scenarios spanning different domains and organizations. Building on this vision, Ricci et al. proposed the idea of a *Web of Digital Twins (WoDT)* [RCM⁺22a] as a conceptual framework to model DTE in an open, distributed and dynamic manner. The core principle states that:

Every strategic Physical Asset of an organisation must have a corresponding Digital Twin, mirroring and augmenting its functionalities and services at the digital level, resulting in an ecosystem of connected Digital Twins [RCM⁺22a].

The WoDT hence strongly emphasizes the service dimension of DTs, envisioning a pervasive digitalization of the physical world. With this pervasive approach, it becomes essential to manage the relationships between PAs at the DT level through links that connect the corresponding DTs. From a knowledge represen-

tation perspective, the WoDT envisions DT as entities that expose *domain-level* knowledge. The proposal is to encode such knowledge through semantic models, making it possible to see the WoDT as a distributed KG. Finally, from an application-oriented perspective, the WoDT aims at supporting the discovery, navigation, and exploitation of DTs and their services to support complex applications that span multiple DTs. Figure 2.8 summarizes the conceptual model of the WoDT. Three main layers are identified:

- The *Physical Layer* which includes the PAs which can be connected by dynamic relationships in the physical world;
- The *Digital Twin Layer* which includes the DTs that model the PAs, linked by relationships that reflect the ones in the physical layer;
- The *Application Layer* which includes applications, users and possibly autonomous software agents that can interact with the DT layer to access DT data and services.

The resulting holistic conceptualization enables services that extend beyond the boundaries of a single DT to encompass the ecosystem as a whole, improving the design of large-scale systems. The application layer implements high-level APIs to expose functionalities to users and applications. These include the possibility to *observe* the evolving state of (subgraphs of) DTs, as well as to *query* the KG to selectively retrieve data from the WoDT.

The WoDT proposes a general-purpose metamodel to support a common understanding of DTs in the ecosystem, fostering interoperability and integration across different application domains. Hence, each DT in the ecosystem is modeled through a set of:

- *Properties* that represent the observable attributes of the PA, as labelled data that can evolve over time based on the state of the PA;
- *Relationships*: that reflect, at the DT level, the connections between the PAs as links to other DTs;
- *Actions*: that represent operations which can be invoked to modify the state or behavior of the PA;

- *Events*: exposing notifications representing domain events of the PA.

The WoDT serves as this thesis conceptual foundation to explore the challenges and opportunities of practically implementing DTE.

2.8 Final Remarks

This chapter provides an overview of the research landscape around DTs, from the historical definition of the concept to the more recent trends leading towards the idea of DTE. This serves the purpose of identifying what is the interpretation of DT adopted in this thesis in the broader scope of DT research. A DT is considered as the software system that implements the *twinning process* to create and maintain a virtual representation of a generic PA, providing services to external users and applications.

In this scope, the chapter highlights the essential properties a DT should have, as well as the complexity of the technological panorama required to implement DTs, with a special focus on the relationship between DT and AI. The service-oriented perspective and the complexity of the many technologies involved, call for a strong emphasis on interoperability, which will be a crucial theme in this thesis to enable DTs to be integrated in larger systems.

Among the various applications of DTs, the chapters also provides a deep-dive into the role of DTs in healthcare, being the target domain explored in this thesis.

Finally, the chapter introduces the idea of DTE as a key abstraction to model complex CPSs through a set of interconnected DTs, highlighting the WoDT as the conceptual framework adopted in this thesis to explore this vision.

Chapter 3

The Semantic Web and The Web of Things

In this thesis, Web technologies and standards are adopted to build interoperability in DTE. Accordingly, this chapter provides an overview of the principal Web technologies, standards and architectural patterns and discuss how they build interoperability in heterogeneous systems. Additionally, the Web of Things (WoT) is presented as an example of how Web technologies can be applied in the IoT context to achieve interoperability.

3.1 History and Definitions

The World Wide Web, or simply, the Web, was initially proposed as an *information management* system by Tim Berners-Lee in 1989 while working at CERN as a way to facilitate information sharing among researchers [BL89, GC00]. The fundamental intuition of the Web is to apply the concept of *hypertext* [Nel67]—text containing links to other documents—to a global network of computers, enabling users to navigate and access information seamlessly across different systems through following Web links. This allows to:

- easily add and link new information without requiring changes to existing systems;

- access and share information across a distributed network of computers in a location-independent manner.

The Web quickly evolved from a private information-sharing system, to a *universe of global accessible information* [BL96] To implement this vision, the Web has been built on a set of fundamental building blocks:

- the Uniform Resource Identifier (URI) scheme for identifying resources on the Web;
- the Hypertext Transfer Protocol (HTTP) protocol for communication between clients and servers;
- the Hypertext Markup Language (HTML) markup language for structuring and presenting information.

These essential Web architecture components are standardized by the World Wide Web Consortium (W3C), an international organization that maintains and develops open standards to ensure the long-term growth of the Web. The W3C defines the Web as:

The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URIs) [WJ04].

The definition highlights the role of *resources* as the first-class entities composing the Web. Resources can represent anything, from documents and images, to services, people and physical objects.

The history of the Web is often divided into three main generations. Web 1.0 refers to the early days of the Web, characterized by static web pages and limited interactivity. In the early 2000s, the concept of Web 2.0 emerged to highlight the shift towards a more dynamic and interactive Web, in which users could not only consume content but also create and share it through social media, blogs, and wikis. The third and current generation, Web 3.0, is often associated with the emergence of structured data, shifting the focus from human consumption to machine-readable representations [Agh12].

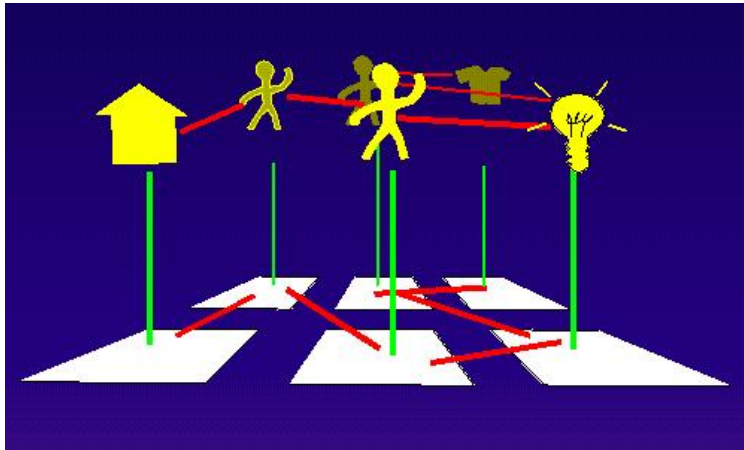


Figure 3.1: A slide showing the concept of the Semantic Web, with documents linked to real-world concepts through semantics, from [BL94b].

The *Semantic Web* [BLHL23] represents a vision for a Web of data that can be processed and understood by machines, enabling intelligent applications and services. The Semantic Web builds on the existing Web infrastructure and introduces new standards and technologies to represent, share, and reason about data in a machine-readable way.

In its keynote talk outlining future directions for the Web at the 1994 World Wide Web Conference, Tim Berners-Lee discussed the role of semantics on the Web (see Figure 3.1):

But to a computer, it looks kind of flat [...] what it sees is a bunch of documents [...] and in fact we know that there is a world that is being described by these documents [...] but there is no semantics out there [...] the World Wide Web scaled, and it was sufficiently fun for everybody to use, but I missed out on putting semantics in [...] Semantics allow machines to manipulate reality. Well, machines at the moment can browse the Web, but if you put semantics over this then machines basically are browsing through reality [BL94a].

Following this original vision, this thesis leverages (semantic) Web technologies and standards to build interoperability in DTE, with the goal of representing physical reality with machine-readable descriptions through networks of interconnected DTs.

3.2 The Web Architecture and Hypermedia

The Web's success is a testament to how its design rationale, open standards, and protocols can enable seamless integration across diverse systems and data sources. The architecture of the Web is based on a client-server model, where clients (e.g., web browsers) request resources from servers (e.g., web servers) using the HTTP protocol. This strong separation of concerns allows clients and servers to evolve independently, as long as they adhere to the same protocols and standards. The principles of the Web architecture were formalized by Roy Fielding's PhD dissertation [Fie00], where he introduced the concept of the *REST* architectural style.

REST is defined by a set of constraints that promote scalability and interoperability in distributed systems. Constraints include client-server separation with stateless interactions for a strong separation of concerns, a *uniform interface* between the two, cache mechanisms to improve performance and scalability supporting a layered system architecture, and code-on-demand for extensibility of client functionality [Fie00]. The *uniform interface* constraint is the cornerstone of REST, achieved through:

- resources identified by URIs which can unambiguously reference any entity in the global Web;
- manipulation of resources via a small set of methods (e.g., HTTP GET, PUT) for exchanging representations (e.g., JSON, XML) of their current or intended state;
- self-descriptive messages using standard methods, media types, etc.; and
- server-guided navigation presenting both data and currently available controls using Hypermedia as the Engine of Application State (HATEOAS).

The last point highlights the importance of *hypermedia* in the Web architecture. Although originally coined to extend the concept of hypertext to include multimedia content, Fielding defined hypermedia as:

Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information. Distributed hypermedia allows the presentation and control information to be stored at remote locations [Fie00].

This perspective shifts the focus from merely linking static documents to enabling dynamic interactions and navigation in the Web. This makes the Web not only a repository of information but also an interactive platform for applications and services. The Web’s hypermedia nature, in fact, allows clients to discover and interact with resources dynamically, following links and controls provided by servers, rather than relying on fixed, predefined interactions.

The HATEOAS sub-constraint of the uniform interface is particularly relevant for supporting the dynamic discovery of resources and actions in the open Web. In a famous blog post, Fielding clarified the importance of HATEOAS, explaining that hypermedia is the “*simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects action*” [Fie08]. The Web’s hypermedia-driven interactions enable clients to navigate and interact with resources dynamically, following links and controls provided by servers, based on the current context and state of the application. This mechanism further cements the Web as a platform for building interoperable systems that can evolve and adapt over time and that can support automated clients in discovering interactions and adapting to changes.

3.3 Semantic Web Technologies

The Semantic Web vision [BLHL23] exploits the hypermedia-based nature of the Web and advances interoperability by introducing machine-readable representations with shared semantics. The Semantic Web is not “a separate Web, but an extension of the current one” [BLHL23], as such, it shares the same fundamental building blocks (e.g. URI, HTTP, REST) and extends them with a set of dedicated standards and technologies to represent data in order to be processed and understood by machines.

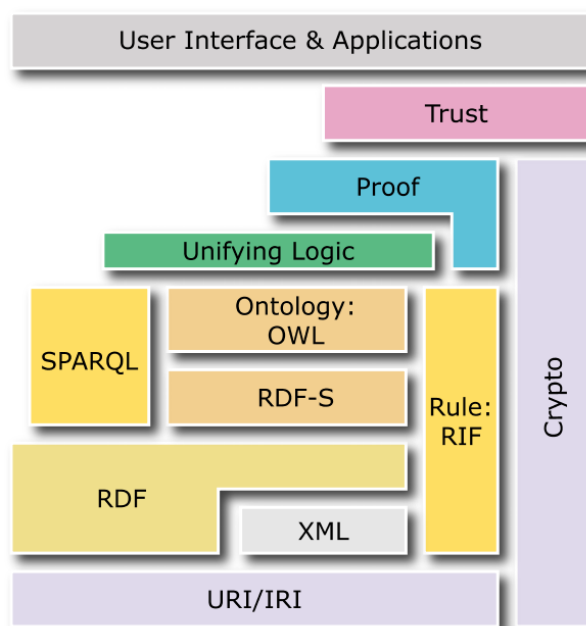


Figure 3.2: The Semantic Web “cake” showing the layers of the technological stack, from [Bra07].

Figure 3.2 illustrates the layers of the Semantic Web technological stack with the main standards associated with each layer.

The foundational building block is the Resource Description Framework (RDF), which provides a common data model for representing information about resources on the Web in the form of subject-predicate-object *triples* [KSCH25]. RDF triples compose a directed labeled graph, where nodes represent resources and edges represent relationships between them. Nodes can be either Internationalized Resource Identifiers (IRIs) (an extension of URIs) identifying a resource or *literals* representing a typed value (e.g., string, number, date). Additionally, RDF supports *blank nodes* to represent resources without a global identifier and support more complex structures than the basic triple structure allow. In its latest version, RDF officially supports also having *triple terms* enabling the use of triples as objects of other triples, previously introduced by the RDF-star extension [HCKS21]. Predicates of a RDF triples are always IRIs, which ensures a consistent and unambiguous identification of the relationships between resources. Through the combination of these basic elements, RDF enables the representation of relationships between

resources, which can encode machine-understandable knowledge about a domain.

RDF can be serialized in various formats, with the original being an XML-based syntax, while more recent formats include the JSON-LD syntax popular due to its compatibility with JSON-based systems [KLC20] and *Turtle* syntax, which provides the most compact and human-readable representation of RDF graphs [CP14].

The next step in the Semantic Web stack is the introduction of vocabularies to provide shared semantics for the terms used in RDF graphs. This is essential to ensure that different systems can understand and interpret data with consistent meanings. The concept of *ontology* [Vic97] is used to refer to a formal definition of concepts and relationships within a specific domain.

The basic vocabulary layer is provided by RDF Schema [TH25], which introduces a simple set of terms to define classes and properties. The more expressive Web Ontology Language (OWL) [olw12] builds on RDF Schema to provide a richer set of modeling primitives. Through OWL ontologies, it is possible to establish taxonomies of concepts and inference rules that enable reasoning over the represented knowledge, as well as axioms that can be used to validate data against the defined schema and discover inconsistencies.

The combination of RDF triples and OWL ontologies enables the creations of *KGs* [HBC⁺21], creating a structured representation of knowledge through the primitives provided by these standards. KG have the benefit of not requiring a predefined schema, creating a flexible and extensible representation of knowledge that can easily evolve over time.

Accessing data in a KG requires dedicated query languages that can traverse the graph structure and retrieve relevant information. The SPARQL Protocol and RDF Query Language (SPARQL) [HS13] is the standard query language for RDF data. It is designed to be SQL-like, making it familiar to use for developers used to relational databases. SPARQL allows the retrieval of data by matching graph patterns. The result of a query is a list of variable bindings that satisfy the specified pattern.

The core set of standards presented above simply provides the foundations for the Semantic Web. Other relevant technologies include validation of RDF data by means of matching *shapes* and conditions using Shapes Constraint Language (SHACL) [KK17], and rule-based reasoning with Semantic Web Rule Language

(SWRL) [HPSB04] which extend the inference capabilities of OWL with Horn-clause rule logic [Hor51].

Interestingly, the technologies described above, can be used outside the Web context, for example for building local KGs to serve as databases for specific applications. When integrated with the Web architecture and principles, however, they enable the creation of so-called *Linked Data* [BHBL23]. Originally proposed by Tim Berners-Lee in 2006 [BL06], Linked Data is grounded on four main principles:

1. User URIs as names for things;
2. Use HTTP URIs so that people can look up those names;
3. When someone looks up a URI, provide useful information, using the standards (e.g., RDF, SPARQL);
4. Include links to other URIs so that they can discover more things.

Together, these principles promote the interrelation of RDF resources, enabling the creation of a global distributed KG that applications can navigate by following and dereferencing HTTP URIs. In Linked Data, URIs are used not only to identify documents and web pages, but also real-world entities, concepts, and abstract ideas. The disambiguation between information resources (e.g., documents) and non-information resources (e.g., people, places, physical objects) is achieved through the use of *hash* or *303* URI patterns [SC08].

To complete the consumption and interaction with Linked Data, the Linked Data Platform (LDP) specification [MSA15] defines rules and best practices on how to apply REST principles to the management of RDF resources. LDP servers expose *containers* of RDF resources, that clients can interact with using HTTP methods to create, read, update and delete resources. Depending on the type of container, the containment relationship is implicitly or explicitly represented, to ease the discovery of related resources.

The combination of Semantic web technologies and Linked Data principles provides a powerful framework for creating distributed knowledge bases that can be easily shared, integrated and queried across different systems. Of course, the adoption of this approach is not without challenges. Despite being around for

over two decades, the Semantic Web is still far from the original vision of a global Web of data that can be easily consumed by machines [Hog20]. Nevertheless, the technological standards and principles described above have been successfully applied and still offer a solid foundation for building open interoperable ecosystems in which data is shared and integrated with clear semantics, supporting automated reasoning and advanced analytics.

3.4 The Web of Things

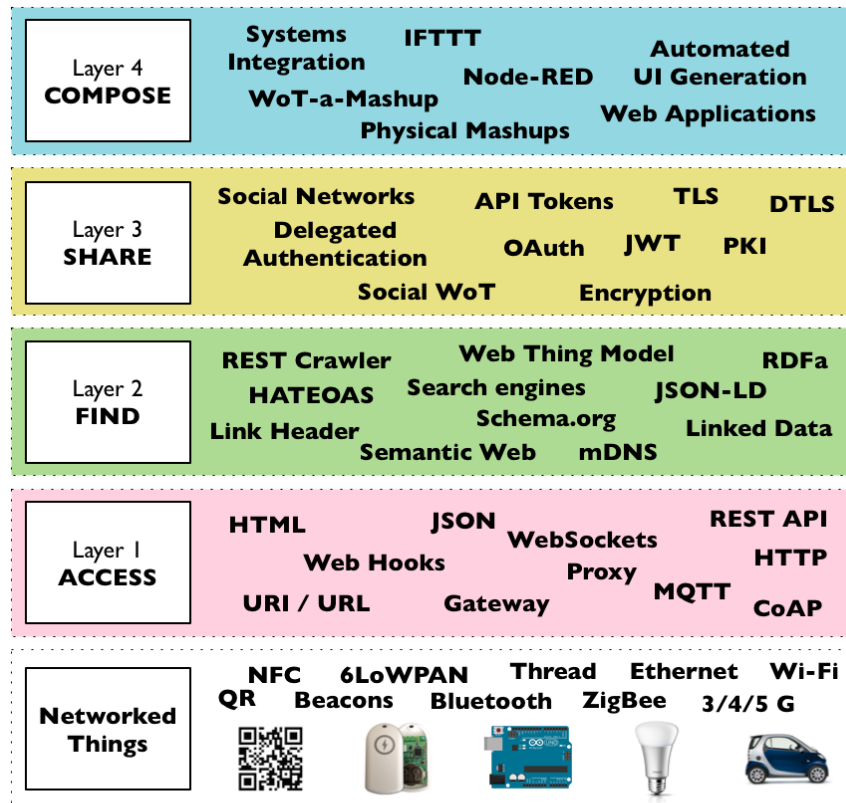
Among the domains in which the application of (semantic) Web technologies has been explored to build interoperability, the WoT is particularly noteworthy for the scope of this thesis as it emerged to connect physical objects to the Web, and enable their seamless integration through Web principles and standards.

The original vision of the WoT [Wil07, GT09] was to leverage the REST architectural style to integrate physical *Things* into the Web, enabling their interaction and composition through standard Web protocols and formats, rather than relying on proprietary or custom solutions.

The WoT original proposal consisted in a REST architecture for *Things*, embedding Web servers directly into devices or using gateways to expose their functionalities as Web resources [GTW10]. The HTTP protocol was used to interact with physical resources using standard methods (e.g., GET, PUT) to retrieve or update their state (e.g., read a temperature sensor, turn on a led). Finally, integrating things into the Web, allowed using Web techniques to discover resources and compose them into applications, for example by using links between resources representing physical devices [Gui11]. The resulting layered architecture is illustrated in Figure 3.3, showing how different technologies from the Web and IoT domains can be combined to support the creation of applications based on *mashups* of physical *Things*.

From its initial proposal, the WoT went under a standardization process within the W3C resulting in a set of specifications that define the architecture and data model to represent and interact with *Things* on the Web [TLMM23, MKK23].

The key component of the W3C WoT is the *Thing Description (TD)* [MKK23], a machine-readable document that describes the metadata, interfaces, and inter-



Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Figure 3.3: The main layers of the W3C WoT architecture, from [GT16].

action patterns of a *Thing*. A TD describes the capabilities a *Thing* exposes in terms of *interaction affordances*, borrowing from the concept of affordance in design theory which represents what interactions can be perceived from an environment [Gib14, Nor13]. TD interaction affordances are of three types:

- **Properties**, which represent the state of a *Thing* that can be read or written (e.g., the current temperature of a sensor);
- **Actions**, which represent operations that can be invoked on a *Thing* (e.g., turn on a light);
- **Events**, which represent asynchronous notifications that a *Thing* can emit (e.g., a door sensor reporting that the door has been opened);

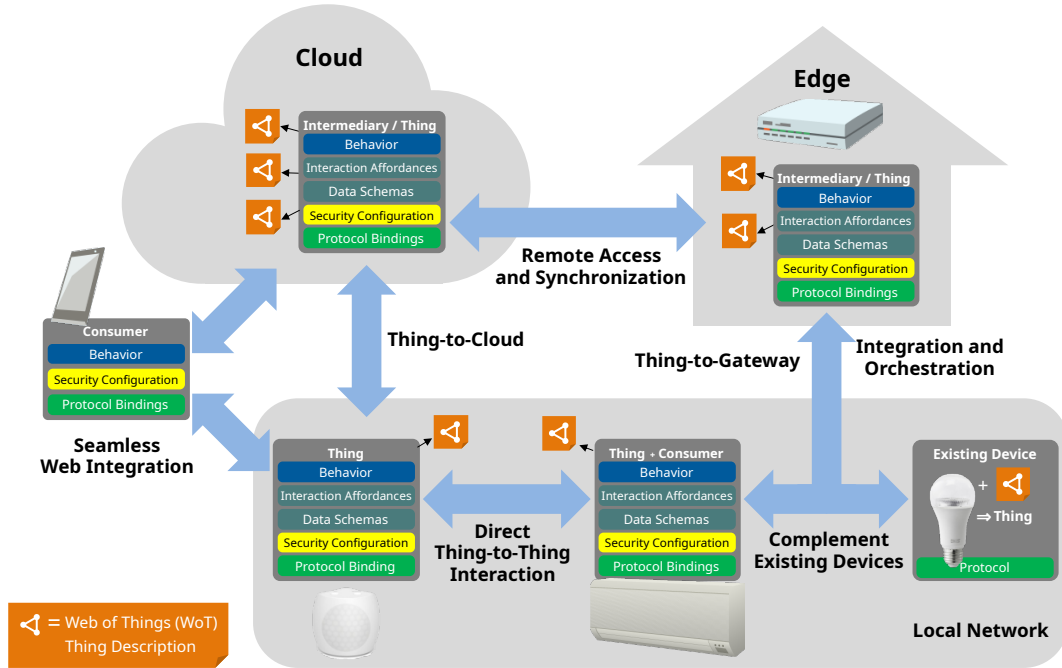


Figure 3.4: An overview of the W3C WoT architecture, from [TLMM23].

Through this simple, yet expressive, metamodel, TD can describe a wide range of physical devices with sensors and actuators, supporting both request-response and publish-subscribe interaction patterns. This supports a variety of scenarios for seamless interaction between TD-enabled components as showcased in Figure 3.4.

In the WoT architecture, an *exposed* thing represent the logical abstraction of the system component which is capable of responding to interactions as described in its TD, hence implementing the behavior behind the exposed affordances. A *consumer* can discover TDs through classic Web discovery mechanisms (e.g., search engines, directories), consume them and interact with the corresponding device. To do this, interaction affordances in a TD are associated with *forms*, which specify the data formats for inputs and outputs of a given affordance as well as the *protocol binding* which describes how to actually communicate with the device through one of the supported protocols.

Decoupling the interaction description from the actual protocol binding allows TDs to possibly support multiple communication protocols and data formats. Ex-

amples of supported protocols include HTTP, Constrained Application Protocol (CoAP), Message Queuing Telemetry Transport (MQTT), and WebSocket, while data formats include JSON, XML, but also binary formats like CBOR [MKK23]. This design ensures that the TD can be used in a variety of contexts, and that clients can seamlessly interact with different devices without needing to know the specifics of their communication protocols beforehand. Additionally, a TD interaction affordances can stay the same even if the underlying physical device is replaced or upgraded, greatly improving maintainability and evolvability of systems built using WoT principles.

Since the WoT is inspired by REST principles, TDs are hypermedia documents. They hence can include *links* to other resources (e.g. other TDs of related devices), as well as the aforementioned *forms* that present to consumers the available interactions offered by the *Thing*.

Finally, TDs can include security metadata to describe the security mechanisms supported by a *Thing*, such as authentication to secure interaction between consumers and exposed things [MKK23].

The TD structure is standardized, and it is usually serialized in JSON-LD, meaning that it is also a valid RDF document and is described through a dedicated ontology [CLPVK23, CKK16]. This makes TD semantic representations of IoT devices, which can be easily integrated with other RDF data and ontologies to provide rich descriptions of devices and their capabilities [JOB⁺14].

The WoT is a successful example of how the transposition of Web principles to a different domain can enable interoperability in heterogeneous systems and support automation in open ecosystems. WoT is currently being applied in various domains, including smart homes, manufacturing, smart agriculture, smart cities, and healthcare [SGM⁺22].

Notably, DTs have been included as an application scenario in the W3C WoT architecture [TLMM23], and there has been work on integrating DT and WoT technologies [JU20, STM⁺22] given the close relationship between the two concepts. This thesis explores the intersection between WoT and DT technologies, investigating how the WoT can be used to build interoperable DTs and DTE.

3.5 Final Remarks

This chapter introduces fundamental architectural principles and technologies from the Web domain that can be leveraged to build interoperability in open distributed systems. It further presents the WoT as an example of how these principles have been applied to the IoT domain to enable integration of IoT devices under a uniform interface, supported by semantic descriptions of device capabilities.

The principles of the REST architectural style, in particular the uniform interface and HATEOAS constraints will be explored in this thesis with the twofold goal of:

- investigate the application of hypermedia-driven interactions in DTE to support dynamic discovery and interaction with DTs;
- explore how autonomous agents can leverage such hypermedia-driven interactions to operate in open and dynamic Web environments.

The Semantic Web technologies and Linked Data principles will be used to advocate for a shared and machine-readable representation of DTs and their capabilities, supporting structured knowledge representation and reasoning in DTE. Finally, the WoT architecture and TD specification will be used as a basis to design and implement interoperable DTs and DTE, exploring how the WoT can be integrated with other DT technologies to build open ecosystems of interconnected DTs. Web technologies and standards hence serves, for the scope of this thesis work, as the *glue* to connect DTE with intelligent applications based on autonomous software agents.

Chapter 4

Multi-Agent Systems

This chapter provides an overview of Multi-Agent Systems (MAS), as a mean to engineer intelligent applications. MAS are a broad research area at the intersection between AI and software engineering. Autonomous agents are used as the main abstraction to model and implement complex software systems. The behavior of such agents can be either programmed, learned or, more recently, generated through the use of GenAI techniques.

This thesis mainly considers *programmed* agents, as the goal is using agents to encapsulate business logic that can automate processes and activities in the context of DT-based systems. Accordingly, the chapter introduces the research area which follows under the broad umbrella of Agent-Oriented Software Engineering (AOSE) and more specifically Agent-Oriented Programming (AOP) as a paradigm to engineer and program MAS. Belief-Desire-Intention (BDI) agents are the central focus of the chapter, as they are amongst the most widely adopted agent model for programmed agents, capable of encoding domain and procedural knowledge through the cognitive abstractions they provide, thus making BDI agents excellent candidates for reliable and explainable intelligent applications.

4.1 History and Definitions

In order to understand the concept of MAS, it is important to first understand what an *agent* is. Despite the term being widely used in different contexts [WJ95]

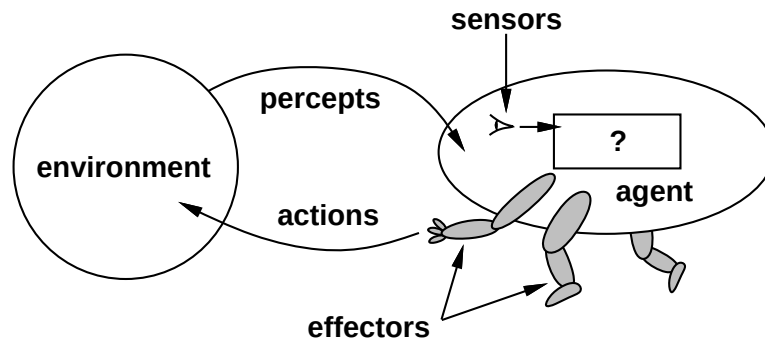


Figure 4.1: Classic schematic representation of an agent and its interaction with the environment, from [RN95].

with different interpretations, a commonly accepted definition is the following:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives [Woo09].

The definition highlights the key characteristics of an agent. Namely, an agent is *situated* in an environment which can be either physical (e.g., for a robotic system) or virtual (e.g., for a software agent). The environment provides the agent with an external context that can be perceived, influencing the behavior of the agent, and affected by the agent actions to reach its objectives. Figure 4.1 shows the relationship between an agent and its environment. The agent receives *percepts* from the environment through its sensors, and performs actions upon the environment through its *effectors*. The agent is also *autonomous*, meaning that it can operate without direct intervention from humans or other systems, and has control over its actions and internal state. Finally, the agent is *goal-oriented*, as it is designed to achieve specific objectives, which can be explicitly defined or learned over time.

It is possible to distinguish between a *weak* and *strong* notion of agency [WJ95]. The weak notion of agency refers to systems that exhibit some characteristics of agents, namely autonomy, reactivity, pro-activeness, and social ability. This kind of agents are usually employed to model concurrent and distributed systems. The strong notion of agency, instead, derives from research on AI and envisions agents

as *intelligent* entities, modeled using *mentalistic* notions [Sho93] to mimic human cognitive capabilities.

The definitions of *agent* focus on the properties of an agent and its behavior, leaving implementation details open. This has led to the development of different agent architectures to implement agents in software systems. The taxonomy of agent models proposed by Russell and Norvig [RN95] distinguishes between increasingly complex agent models:

Simple Reflex Agents operate on a set of predefined trigger-action rules and to respond to stimuli from the environment.

Model-Based Reflex Agents keep an internal model of the environment, updating it over time, and allowing them to make decisions based on both current perceptions and past experiences.

Goal-Based Agents are designed to achieve specific goals. They can evaluate different actions not only based on the model of the environment, but also on the potential of achieving their goal, allowing for more flexible and adaptive behavior.

Utility-Based Agents not only aim to achieve goals but also consider the desirability of different outcomes. They use a utility function to evaluate the potential benefits of various actions, enabling them to make decisions that maximize overall satisfaction.

Learning Agents can improve their performance over time through learning from experiences. They can adapt to changing environments and optimize their behavior based on feedback, making them more robust and effective in dynamic settings.

Implementing an agent essentially boils down to specify its behavior in response to its perceptions of the environment, with different degrees of complexity. How this is achieved depends on the agent architecture and model adopted, but it is possible to generally distinguish two main categories: *programmed* or *learning* agents, depending on whether their behavior is explicitly defined by a developer or learned from data and experience. Although the boundary between the two

categories is not always clear-cut, as learning techniques can be used to complement programmed behavior [BEFSH⁺20, BR20], they provide a useful distinction to understand the different approaches to implement agents. The involved techniques are, in fact, quite different. While programmed agents rely on either pre-specified behavior (e.g., rules, decision trees) or assemble it dynamically through planning algorithms [GNT04], learning agents require a different a training phase where machine learning techniques are employed to learn patterns and behaviors from data. The most common learning technique for agents is Reinforcement Learning (RL), where agents learn to make decisions by interacting with the environment and receiving feedback in the form of reward or punishment based on their actions, thus learning a *policy* to maximize the cumulative reward over time [SB98].

Recently, the advent of GenAI techniques, has brought new opportunities to implement agents. *Agentic AI* [AKD25] refers to the use of GenAI models, such as LLMs, to create agents that can perform tasks autonomously by generating responses and actions based on natural language prompts. Agents based on these techniques can be seen as a third category of agents, where the behavior is neither explicitly programmed nor learned from experiences, but rather synthesized on-the-fly by the generative model based on the context and instructions provided.

Having defined what an *agent* is, a *Multi-Agent System* is, finally, a system composed of multiple interacting agents, which can be either cooperative or competitive, working together to achieve individual or shared goals [Woo09]. The interaction between agents can be direct, through agent-to-agent communication [KSN00], or indirect, through the manipulation of the shared environment they operate in [WOO07].

MAS are particularly useful for modeling and solving complex problems that are difficult to tackle with traditional approaches. The behavior of a MAS emerges from the interactions of its agents, allowing to break down complexity into manageable components whose behavior can be defined independently. In the case of learning agents, Multi-Agent Reinforcement Learning (MARL) techniques tackle the problem of training multiple agents at the same time, considering the interactions and dependencies between them [ACS24].

Due to their versatility, MAS have been adopted in various domains, including robotics [IBDdRRT⁺12], (social) simulation [UW18, Dav01], and distributed con-

trol systems for smart grids [MET⁺14], IoT [SC17], or traffic management [TWS20].

Considering this very broad landscape, for the scope of this thesis the term *agent* is used to implicitly refer to *programmed agents* and specifically on BDI agents as an example of systems with a *strong* notion of agency. The goal is to leverage the cognitive abstractions provided by BDI agents to model and implement predictable, reliable and explainable intelligent behavior in DT-based systems, encoding domain knowledge and business logic in the agents' mental state and plans.

4.2 Agent-Oriented Software Engineering

The engineering of MAS is a complex task, as it involves designing each individual agent and their interactions with each other and the environment. Hence, the research area of AOSE has been developed as a sub-field of software engineering, focusing on the engineering of distributed systems using agent-oriented abstractions [Jen99, WC01].

A big effort in AOSE has been devoted to the definition of methodologies to guide the analysis, design and implementation of distributed systems through the agent abstraction [WC01]. Notable examples of such methodologies include Gaia [WJK00], Tropos [GMP02], Prometheus [PW02], INGENIAS [PGS03], and PASSI [Cos05]. Nevertheless, despite the abundance of methodologies born within the AOSE community, their adoption in mainstream software engineering has been limited [GSFF15]. Most methodologies focused only on requirement analysis, modeling agent interactions, roles, and behaviors and some provide supporting tools in the form of diagrams and code generation plugins [BGZ06, GSFF15]. The developed system was then free to be implemented using any general-purpose programming language, without necessarily having a direct mapping between the agent-oriented design abstractions and the implementation constructs.

Complementing the high-level design-oriented efforts of AOSE methodologies, AOP has been proposed envisioning agents not only as a design abstraction, but also as a programming concept to practically implement software systems [Sho93]. In AOP agents are considered the fundamental building blocks of a software systems (just like objects are in Object-Oriented Programming (OOP), and functions

are in Functional Programming (FP)), and the development of a software system involves defining the agents, their behaviors, and their interactions. AOP specifically targeted the *strong* notion of agency, hence characterizing the state of agents using mentalistic notions such as beliefs. Furthermore, agents in AOP interact by means of messages following an Agent Communication Language (ACL) based on speech act theory [Aus75]. The intended purpose of AOP is to encode complex behavior in a natural and intuitive language, that supports high-level abstractions to model the decision-making process of agents, as well as their interactions.

The work on AOP has led to development frameworks and programming languages specifically designed specifying agent behavior, providing tools and libraries to facilitate the implementation of MAS [CF21].

When scaling the complexity from single agents to a MAS, it has been recognized that also other programming abstractions are needed other than agents alone (Figure 4.2). These *dimensions* of MAS programming, other than the aforementioned *agent* dimension, include the *environment* dimension [WOO07] which focuses on the design of the shared environment in which the agents operate, the *interaction* dimension [Huh01] which focuses on the communication and coordination mechanisms between agents, and the *organization* dimension [BHS07] which focuses on the structure and roles of agents within the MAS. Focusing on different dimensions, allows to address the same problem in different ways, as each dimension provides different abstractions and mechanisms to model and implement the behavior of a MAS [BBHR19].

The environment programming dimension has been introduced to recognize the environment as a first-class abstraction in MAS programming [WOO07]. In this perspective, the environment is not just a passive context, but an active component that can influence and be influenced by agents. Namely, the environment can serve three major roles in supporting the activity of a MAS at different levels:

- at a *Basic Level* the environment provides access to the deployment context of the agent, allowing it to interact with external resources either them being physical (e.g., sensors, actuators) or virtual (e.g., databases, web services);
- at an *Abstraction Level*, the environment provides a higher-level representation of the context in which agents operate bridging the gap between the

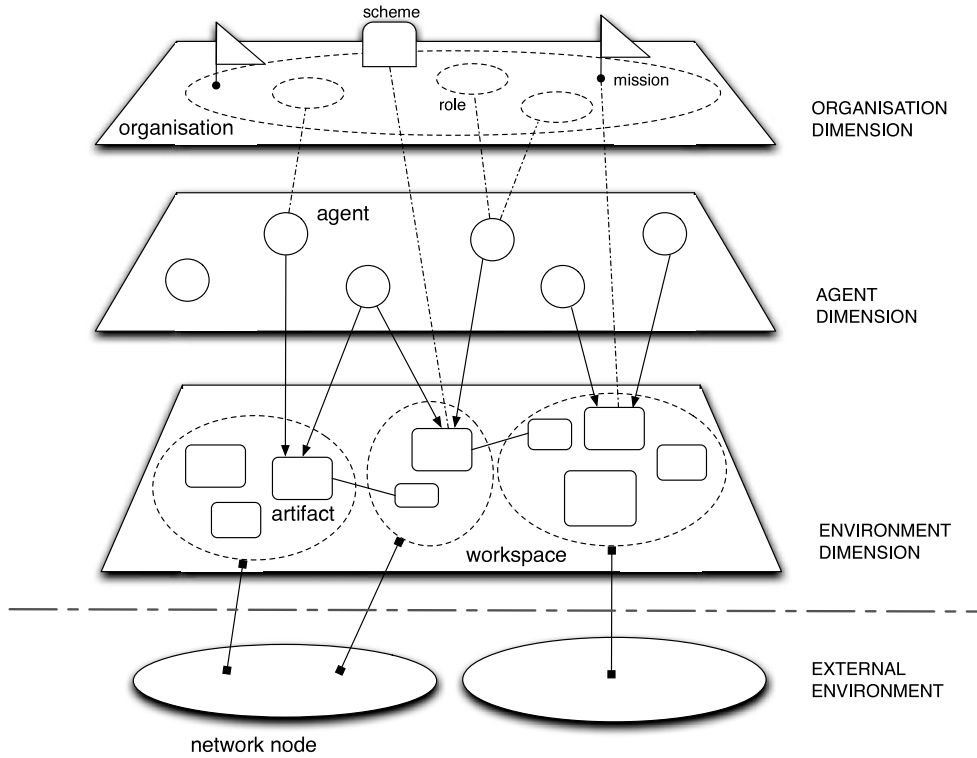


Figure 4.2: High-level overview of the main programming dimensions of a MAS, the *interaction* dimension, not shown, focuses on interaction between agents, from [BBH⁺13].

low-level details and the agents' behavior;

- at a *Interaction-mediation Level*, the environment facilitates the communication and coordination between agents, regulating access to shared resources and hence becoming an active entity in the MAS.

Elaborating on this perspective, the *Agents and Artifacts (A&A)* metamodel, proposes to model the environment as a collection of *artifacts* shared amongst agents [ORV08]. Artifacts are defined as computational entities that encapsulate both state and behavior, providing a means for agents to interact with the environment in a structured way. This approach allows for a more flexible and dynamic modeling of the environment, introducing a further *reflection level* where artifacts can be created, modified, and destroyed at runtime by the agent themselves, reflecting the changing needs and goals of the agents [ROV⁺07].

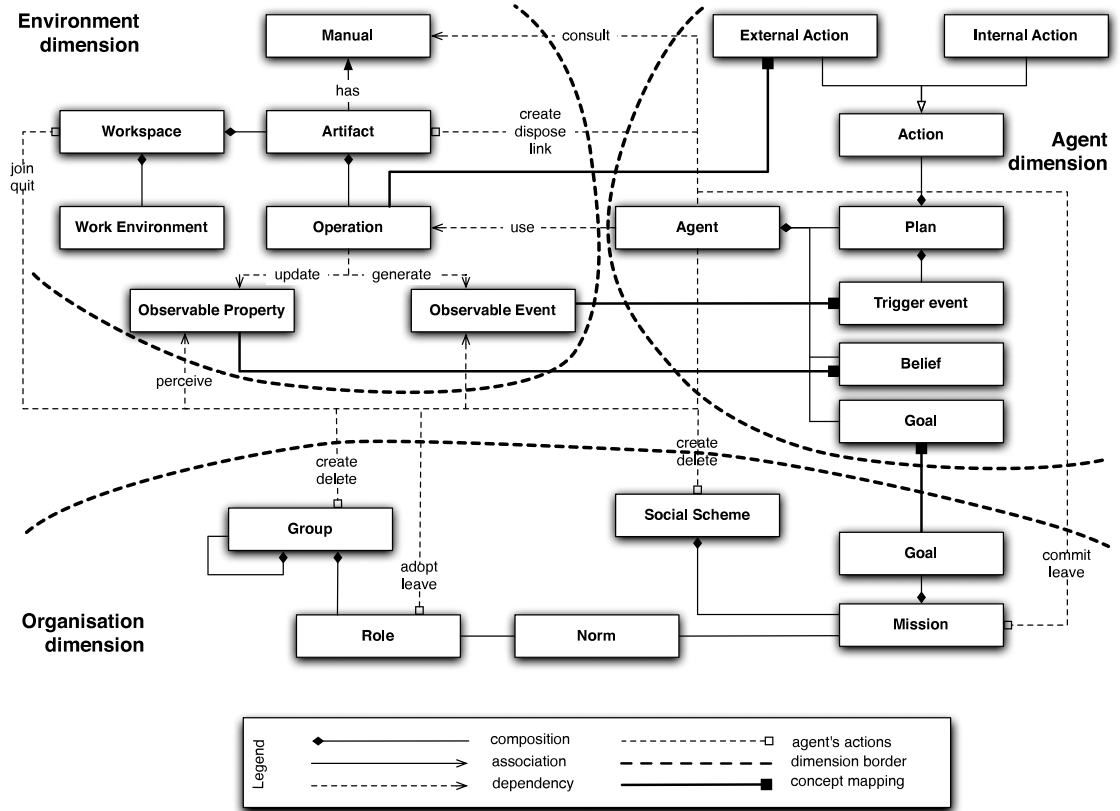


Figure 4.3: Schematic representation of the main abstractions of Multi-Agent Oriented Programming (MAOP) and their relationships, from [BBH⁺13].

The organization programming dimension [BHS07] focuses on the structure and roles of agent societies within a MAS. MAS organizations shift the focus from individual agents to the collective behavior of groups of agents [FGM04]. Organization programming targets those kinds of MAS where the structure and roles within an organization are explicitly defined and managed in a top-down fashion, as opposed to emergent organizations where the structure arises from the interactions of individual agents in a bottom-up fashion [BHS07].

The proposal of MAOP evolves AOP to consider all the different dimensions of MAS programming in a unified framework [BBH⁺13, BBHR20]. The paradigm has been supported by the development of the JaCaMo [BBHR20] framework which integrates three main components:

- **Jason:** a Java-based interpreter of AgentSpeak(L) [Rao96] for programming

and executing BDI agents [BHW07];

- **CARTAgO**: a framework to model and implement the environment as a set of artifacts that agents can use to perceive and act upon the environment [RPVO09];
- **Moise⁺**: a framework to define and manage the organization of agents in terms of roles, groups, and missions [BHS07].

MAOP and the JaCaMo framework have been widely influential in the AOSE community, providing an integrated foundation for the engineering of complex MAS [BBHR20]. Figure 4.3 provides a representation of how the different programming dimensions of a MAS are integrated in the MAOP paradigm, showing the main abstractions and their interrelations. When discussing the engineering of MAS in this thesis, a MAOP-inspired approach is adopted, with a focus on the agent and environment dimensions (and the A&A metamodel in particular), as these are most relevant for the integration of MAS with DT-based systems. While the organization dimension is also of interest, it is not specifically addressed here, as the focus is on relatively small-scale MAS where the organization remains implicit.

4.3 Belief-Desire-Intention Agents

The BDI model originated from the philosophical theory of human practical reasoning developed by Michael Bratman [Bra87]. The theory focused on the notion of *intention* (i.e., the commitment to carrying out a plan of action), to describe future-directed decision-making, explaining how humans decide what to do when they have multiple desires and goals to achieve.

The model influenced the development of computational agents [RG91], leading to agent architectures such as the Procedural Reasoning System (PRS) [GL86], which provides a framework for implementing BDI agents in software systems. Computational BDI agents are characterized by the following main abstractions:

Beliefs: a set of facts and rules representing an agent’s memory, collecting knowledge about its perception of the world, itself, and other agents;

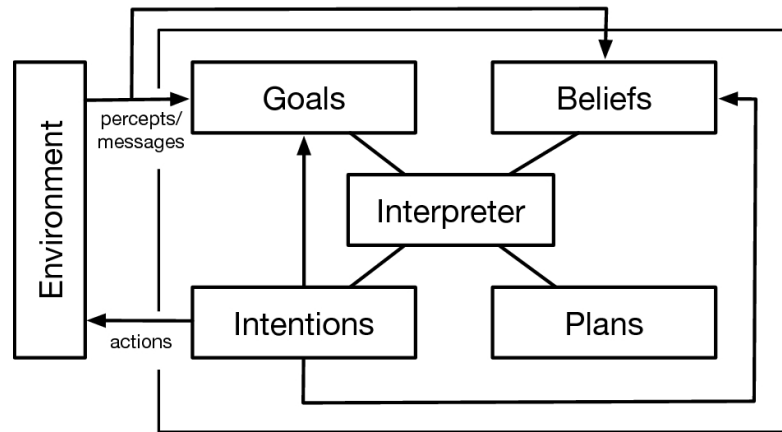


Figure 4.4: Schematic, high-level representation of a BDI agent, from [BEFSH⁺20].

Desires: a set of goals, representing (possibly partial) descriptions of desirable states of the world the agent is willing to achieve, test, or maintain;

Intentions: a set of tasks the agent is currently committed to, in order to satisfy some of its desires;

Plans: a set of *recipes* representing the agent’s procedural memory, hence encoding the know-how about achieving a given intention under certain conditions.

Figure 4.4 shows how these abstractions relate to each other in a schematic representation of a BDI agent. The agent acquires percepts and receives messages from other agents through the environment. This can lead to an update of the agent’s beliefs or goals. Based on its beliefs and goals, the agent deliberates on intentions to commit to, and selects plans to achieve them. Finally, pursuing an intention, through the execution of the selected plans may lead to actions that affect the environment. In the figure, a central *interpreter* of the agent program is shown. This component is responsible for managing the agent’s execution, often referred to as the *reasoning cycle* of the agent.

BDI agents continuously cycle through a reasoning process which involves three main phases:

1. *Perception:* the agent perceives its environment through its sensors, updating its beliefs accordingly;

2. *Deliberation*: the agent deliberates on its desires and intentions, possibly revising them in light of the updated beliefs;
3. *Action*: the agent selects and executes one or more plans to carry on its current intentions, possibly affecting the environment through its actuators.

Through this mechanism, agents continuously adapt their behavior to the changing environment, acquire new beliefs that may influence their future behavior, and pursue their goals by executing plans.

Programming a BDI agent involves defining its initial beliefs, goals, and a set of plans that support the agent in achieving its goals under different conditions. The BDI model has been initially formalized with a logical language, AgentSpeak(L) [Rao96] which later has inspired several agent programming languages. Amongst them Jason [BHW07] is a Java-based interpreter for an extended version of AgentSpeak(L), keeping the original syntax and semantics.

Custom BDI agent programming languages, support developers in implementing BDI agents by providing high-level abstractions and constructs that facilitate the definition of beliefs, goals and plans. This has the benefit of allowing developers to focus on the agent's behavior, rather than low-level implementation details.

4.4 Web-based and Hypermedia MAS

When introducing the Semantic Web (see Chapter 3), Berners-Lee et al. envisioned a future where software agents would roam the Web, accessing and processing information on behalf of users [BLHL23, Hen01].

The relationship between MAS and the Web has been widely explored especially in the context of service-oriented computing [SH04]. Agents were seen as a natural fit to automate the composition of Web services due to their autonomy, social ability and goal-driven behavior. The Web, in turn, provides an open ever-evolving environment rich of resources and services that agents may need to discover and (learn to) use.

Despite these early visions, the adoption of MAS in Web-based systems has been limited. The famous question “Where are all the intelligent agents?” [Hen07] provocatively highlighted the gap between the early expectations and the actual

adoption of MAS in the Web context. As pointed out by the answer of McBurney and Luck [ML07], agent-based technologies have actually been adopted “behind the scenes” in various application domains, including the Web, not necessarily as killer applications themselves, but as the enabling technology for the engineering of complex distributed systems.

Nevertheless, early integration of MAS and Web services in the early 2000s were still far from the original vision of the Semantic Web. They mostly focused on the composition of services through the standards available at the time (e.g., SOAP, WSDL) which relied on the Web for transport [New02]. Similarly, on the MAS side, the focus was primarily on the development of agent communication protocols and frameworks that could operate over the Web, with the prominent example being the Foundation for Intelligent Physical Agents (FIPA) specification for communication between agents over HTTP [Fou02].

Motivated by more recent trends in Web development such as the pervasive adoption of REST [Fie00], and the WoT [GT16] enabling Web-based interactions with physical environments, the research area of *Web-based MAS* has regained interest in the last decade. This has led to a community effort to revisit the integration between MAS and the Web [BCHR21, BCH⁺23] and the creation of the *Autonomous Agents on the Web* W3C Community Group¹ to tackle the challenges of engineering Web-based MAS.

In this context, the proposal by Ciorrea et al. envisioning *Hypermedia Multi-Agent System (hMAS)* [CMG⁺19] as a new class of Web-based MAS is particularly relevant (Figure 4.5). The proposal leverages the principles of REST—and especially HATEOAS—to design MAS that are fully integrated with the Web. The key, following the A&A metamodel [ORV08], is to consider the Web as a shared environment where agents can operate. The proposal of hMAS is grounded on a set of principles [CBR19]:

- **Uniform Resource Space:** every entity in the hMAS (e.g., agents, artifacts, organizations) should be represented as resources in the hypermedia environment.
- **Single Entrypoint:** given a single entrypoint to the hMAS (e.g. a URI),

¹<https://www.w3.org/community/webagents/>

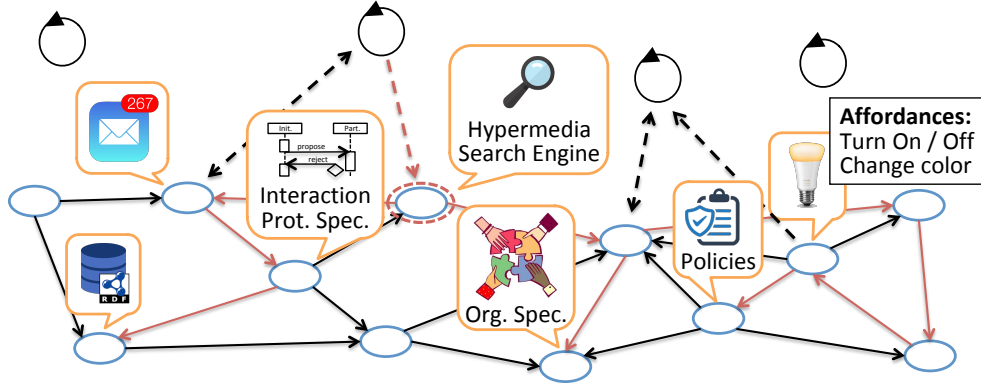


Figure 4.5: Schematic representation of the concept of Hypermedia Multi-Agent System showing how elements of a MAS can be integrated and represented in a hypermedia environment for agents to discover, from [CMG⁺19].

agents should be able to discover the knowledge required to participate in the MAS by navigating the hypermedia environment. This is typically referred to the challenge of *arrive and operate*.

- **Observability:** Resources in the hMAS should be observable by agents, allowing them to perceive changes and react accordingly.

The proposal of hMAS has been supported by the development of the Yggdrasil framework [CBR19], which provides a set of tools to develop hypermedia environments based on the A&A metamodel, extending CArtAgO with RESTful interfaces and hypermedia representation of artifacts.

Developments in the area of hMAS have shown the potential of using MAS in dynamic Web environments, with a focus on the WoT [CMM18]. In this context, the WoT model based on *affordances* [Nor13] is particularly interesting as it provides a natural way to model the discovery of interaction possibilities for agents in the hypermedia environment. This concept has been explored proposing the addition of *signifiers* [Nor08] as a first-class abstraction for the design of hMAS [VCML23], hence introducing a way to guide agents in the discovery of relevant resources and interaction possibilities in order to more efficiently perceive and navigate the hypermedia environment, similarly to how humans do when browsing the Web.

The principles and research efforts in the area of hMAS are significant for this thesis, as they serve as a bridge between intelligent applications designed with MAS and services and resources available on the Web, including DT-based systems.

4.5 Multi-Agent Systems and Digital Twins

The relationship between MAS and AI tends toward the development of DTs that are more and more autonomous and capable of adapting to changing conditions in the PA they mirror, to optimize their performance and achieve specific goals (see Section 2.4). So-called *Autonomic* or *Cognitive* DTs [MCFA23] are being studied to endow DTs with self-* properties (e.g., self-optimization, self-healing, self-protection) [KC03, FGG⁺22]. In this context, DTs are seen as components to engineer complex autonomous CPS through the exchange of simulation models to understand the behavior of other elements of the systems and achieve better overall coordination towards global goals [EGF⁺21]. It is hence not surprising that there is a significant overlap between the research areas of autonomous agents and MAS and DTs and that integration between the two abstractions are being investigated [HCMM21].

Surveys highlight how the convergence of MAS and DTs is growing predominantly within the manufacturing domain, with smart agriculture, smart cities and energy domains following [PMN⁺24, KC24]. Healthcare is also considered as a relevant domain for MAS and DTs combined. Croatti et al. [CGMR20] discuss how DTs of relevant assets of a healthcare organization could be used by agents to support decision-making and coordination of activities in a hospital setting.

Most work do not disclose the specific frameworks and tools used to implement MAS and DTs, and some propose new ones, suggesting a fragmentation of uncoordinated proposals despite the substantial number of tools established within both areas [PMN⁺24]. This calls for more efforts in the integration of existing tools and frameworks, following established paradigms and standards, to facilitate the adoption of MAS in DT-based systems.

Whereas most surveyed works focus on a one-directional view of the combination of MAS and DTs— with agents supporting the implementation of DTs [KC24] — other combinations are possible. Mariani et al. discuss how DTs and MAS

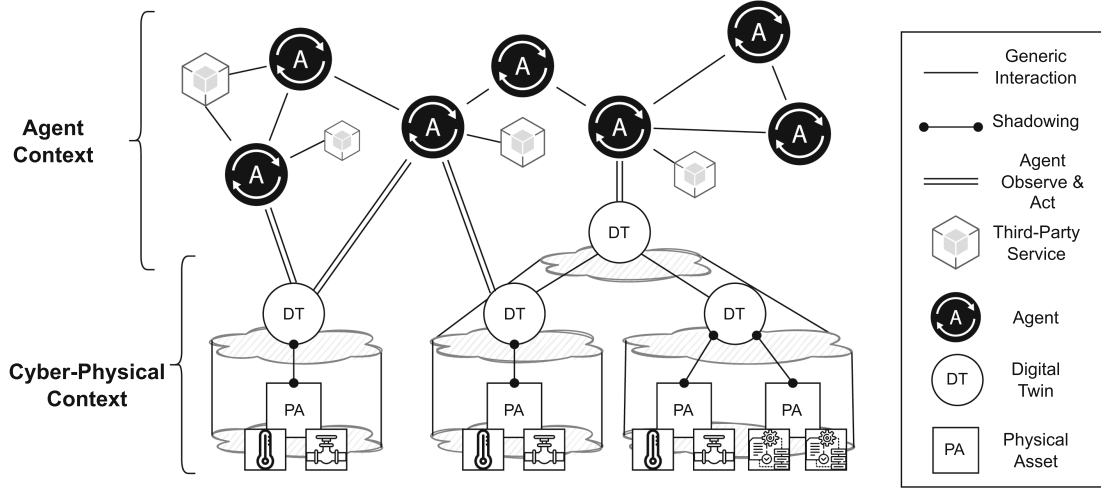


Figure 4.6: Schematic representation of the integration between MAS and DTs through the environment dimension. DTs can be seen as artifacts which the agents interact with, from [MPR22].

have different characteristics and can complement each other in a synergistic way [MPR22]. The integration between MAS and DTs is envisioned through the environment dimension (Figure 4.6). In this perspective, DTs can be seen as artifacts, digitally representing physical entities in the MAS environment.

This opens up new possibilities for the design of autonomous DT behavior. On an *individual* perspective each DT can be paired with an agent keeping separation of concerns between the DT physical asset representation and the agent's autonomous behavior and decision-making. Additionally, this opens up a *system* perspective, in which agents can instead reason on top of an environment composed of multiple DTs, coordinating their behavior to achieve system-level goals, a vision also shared by the idea of WoDT [RCM⁺22a] (Section 2.7).

Both perspectives are relevant for this thesis, as the goal is to explore how to engineer intelligent applications that rely on DTEs to interact with the physical world. Understanding the role of MAS in the DTE context and proposing structured means to integrate them is one of the open challenges that this thesis contributes to address.

4.6 Final Remarks

This chapter provides context on MAS and BDI agents, which are the main abstractions used in this thesis to model and implement intelligent applications. In particular, the chapter highlights how historically MAS have been studied both as a design abstraction for the engineering of intelligent systems. The thesis follows the angle of *programming* MAS, specifically through the MAOP paradigm that integrates the different programming dimensions of a MAS in a unified framework that considers agents, environment, interaction and organizations.

As the thesis explores the integration of MAS with DTEs envisioning the possibility of using Semantic Web technologies to achieve it, the chapter also discusses the relationship between MAS and the Web, highlighting recent efforts in the area of hMAS that leverage REST and hypermedia principles to design MAS that are fully integrated with the Web. Finally, the chapter presents how the proposals targeting the embedding of intelligent functionalities and autonomy in DTs are overlapping with research on autonomous agents and MAS, motivating the need for structured approaches to integrate the two abstractions which will be further explored in the contributions of this thesis.

Part II

Digital Twin Ecosystems

Chapter 5

Engineering Digital Twins

*This chapter contains contributions from the
following works I authored:
[MBM⁺ 25, PMB⁺ 25a, PMB⁺ 25b]*

In an attempt to answer **RQ1**, this chapter presents a proposal for the engineering of DTs focusing on the challenges of integrating heterogeneous sources of PA data and exposing DT functionalities as services for digital applications. Additionally, the chapter addresses the challenge of managing the *lifecycle* of a DT, which is crucial to ensure that the DT can accurately reflect the state of the PA, or signal when it may not be able to do so, in order for applications to make informed decisions on how to interpret the data provided by the DT. Finally, the modelling of *augmented* functionalities of a DT is discussed, with a modular approach in order to decouple them from the primary DT responsibility of *shadowing* the PA state. The resulting contribution is the proposal of an architectural framework and patterns for the engineering of DTs, which has been implemented in a supporting open-source framework: the White Label Digital Twins (WLDLT) Java Library¹.

5.1 An Abstract Architecture for Digital Twins

Starting from the principles outlined by the 5D model [TZLN18], and incorporating insights from both recent academic research [RCM⁺22a, BBF⁺23] and relevant standards from industry bodies [ETS24]. This section presents a proposal aiming

¹<https://wldt.github.io>

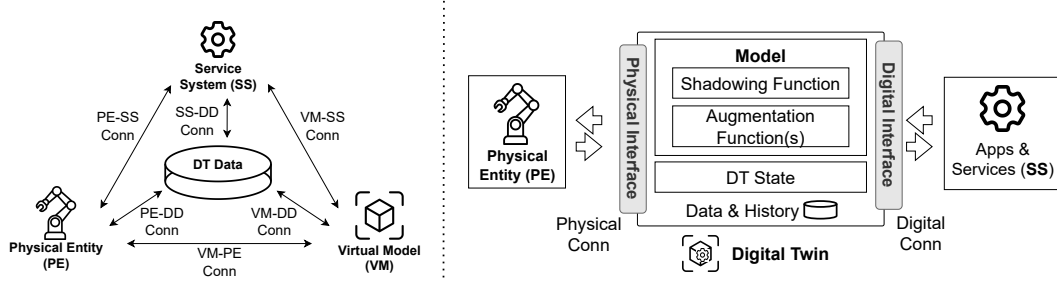


Figure 5.1: A side-by-side mapping between 5D DT modelling (left) and the abstract DT architecture presented in this section (right). The architecture is presented with named boxes that highlight the main components of a DT and its interaction with the PA and external applications.

to generalize an abstract architecture that captures the core software requirements for DT development.

In the 5D model (Figure 5.1, left), the first dimension, Physical Entity (PE), represents the actual physical object or system being modeled as a DT. The second dimension, Virtual Entity (VE), is the digital representation of the physical entity, replicating its characteristics and behaviors. The third dimension, Connection (C), ensures linkage between the physical and virtual entities, including communication technologies, data transmission protocols, and synchronization mechanisms for real-time interaction and data exchange. The fourth dimension, Data (D), covers data management aspects, ensuring accurate and timely data for simulation, prediction, and optimization. Finally, the fifth dimension, Service (S), involves various services enabled by the DT, such as monitoring, simulation, prediction, control, and optimization, leveraging data and insights from the virtual entity to enhance the performance and efficiency of the physical system.

Moving this towards an abstract architectural perspective, a DT can be described as the combination of three main high-level components as schematically illustrated in Figure 5.1 (right):

- *Physical Interface (PI)*: tasked with both the digitalization process and the ongoing synchronization of the DT and PA throughout their lifecycle based on its characteristic cyber-physical nature and the supported protocols and data formats (e.g., HTTP and JSON, MQTT and binary);

- *Digital Interface (DI)*: complementing the *PI*, it manages the routing of DT’s internal variations and events directed towards external digital entities and consumers ensuring the DT’s interaction, interoperability and observability;
- *Model (M)*: Defines the DT’s behavior through the digitalization process together with augmented functionalities. The *shadowing* process is responsible for handling events from both the *PI* and *DI* to model and replicate the state of the asset, while *augmentation* [MLC20] functions extends functionalities of the PA through additional features and capabilities supported by the software nature of the twin (e.g., simulation, machine learning inference models).

Table 5.1 discusses how the proposed architecture supports the core properties of DTs [MLC20] compared to both existing research from the scientific literature and features offered by major production-ready platforms. Existing DT cloud and vendor-specific platforms – such as the ones described in Section 2.3 – either mostly serve as data repositories, storing data from physical assets, with pre-processing handled externally by a combination of services, or implement monolithic solutions with limited flexibility and adaptability to different application contexts. In contrast, the proposed approach is *event-driven*, where both the *PI* and *DI* are responsible for capturing events from their respective domains (physical and digital) and forwarding them to the *M* for processing. This allows DTs to operate as independent software entities, communicating with the external world through finely configurable interfaces.

Following the proposed architecture, a DT can be formally represented as:

$$DT = \langle PI, M, DI \rangle \quad (5.1)$$

Where the *PI* captures data from the Physical Asset (PA), represented as a stream of time ordered *physical* events e_{PA} , and controls physical actuation on the object through time ordered actions a_{PA} , the model *M* encapsulates the PA behavior, processes physical events e_{PA} , and generates time ordered *digital* events e_{DT} , and, finally, the *DI* consumes digital events e_{DT} , exposes them to

Property	Description
Representativeness & Contextualization	Traditional: DTs are limited to store properties and relationships received from external entities, and do not implement a model or a behavior themselves. DTs fail to encapsulate the PA, and the responsibility of the contextualization is entirely externalized. Event-Driven: DTs are active entities aware of the characteristics of the PAs through dedicated adapters able to support a fine-grained representation and contextualization thanks to internal models and the event-driven approach.
Shadowing	Traditional: The shadowing is not correctly encapsulated within a DT and is dispersed across different components. The constraint to rely on a fixed set of communication protocols and platform-specific data formats creates a strong vendor lock-in, limiting re-usability. Event-Driven: DT directly encapsulates the functionalities and responsibilities to communicate with the PA through the adoption of any required protocols and data formats and decouples the responsibilities of interacting with the PA through re-usable and interoperable modules.
Replication	Traditional: Adoption of centralized and monolithic solutions (e.g., Cloud-only) represents a critical limitation to cross-domain interactions, scalability, and fault tolerance since DTs are designed as passive singleton instances operating in isolated tenants. Event-Driven: The event-driven approach natively supports the creation of multiple active, independent, and modular DTs interacting by means of events and operating through multi-layered and cross-domain deployments.
Composability	Traditional: The composition logic and its management (regardless of whether it is linking or an effective interconnection) are fully externalized, and the DT is passively subject to them. Event-Driven: Each DT is active and directly handles the composition process through behavior defined by its model, supported by dynamic adapters and a native, event-based notion of observability.
Augmentation	Traditional: DT augmentation is a concept entirely missing from state-of-the-art platforms, and it is outsourced to external computational services. Event-Driven: Augmentation is implemented through a flexible plug-in mechanism allowing the injection of novel functionalities and relying on existing modules to effectively augment the DT.

Table 5.1: Comparison of DT capabilities: limitations of existing (traditional) approaches versus the benefits of the proposed event-driven solution.

external consumers, and supports the invocation of *digital* actions a_{DT} to modify the behavior of the PA through the DT.

To better align with multimodel perspectives of DTs, the model M can be actually interpreted as a set of models:

$$M = \{m_1, m_2, \dots, m_n\} \quad (5.2)$$

In the proposed approach, the core functionality of the DT is the shadowing process, which handles the transformation of data received from the PI , feeding the models that characterize the DT. The result of the shadowing process is to compute the state of the DT S_{DT} , which—following the WoDT metamodel [RCM⁺22a]—can be represented through sets of:

- **Properties** (P): labeled data that change with the evolution of the PA state;
- **Relationships** (R): capturing the existing and dynamic connections among PAs within the system, mirroring them between the corresponding DTs;
- **Events** (E): non-persistent signals captured by the information gathered from the associated PA;
- **Actions** (A): the possible operations that the DT allows to be invoked on behalf of the PA, to either send feedback to the physical entity or exploit a service exposed by the DT.

At any given time t , the state of the DT can hence be represented as:

$$S_{DT} = \langle P, R, E, A, t \rangle \quad (5.3)$$

Computed states can be stored to maintain a history H of the DT, which can be modeled as a time ordered sequence of states:

$$H = \langle S_{DT_1}, S_{DT_2}, \dots, S_{DT_n} \rangle \quad (5.4)$$

Additionally, to fulfil the *shadowing* process, state updates can then be communicated to applications through the DI , fulfilling the objective of the DT in

providing an up-to-date representation of the PA. The shadowing process can be represented as a function $Shad$ that given an event from the PI e_{PA} , the set of models M , and the history of previous states H , computes the new state of the DT and generates an event e_{DT} to be sent through the DI :

$$e_{DT}(S_{DT}) = Shad_{PA \rightarrow DT}(M, e_{PA}, H) \quad (5.5)$$

The shadowing is bidirectional: when actions are invoked, the DT receives an action request a_{DT} on its DI , validates it through the models M possibly evaluating the history H , and then it may propagate the request through its PI sending a command a_{PA} to have an effect on the PA.

$$a_{PA} = Shad_{DT \rightarrow PA}(M, a_{DT}, H) \quad (5.6)$$

It is crucial to note that a digital action request does not directly change the state of the DT. Any changes of the state occur only as a result of the propagation of physical changes from the PA to the DT. This abstract architectural model addresses the main functional requirements of a DT. It provides a clear separation of concerns between different components focusing on the *shadowing* process as the core responsibility of the DT encapsulating the complexity of physical and digital interactions through the PI and DI .

Finally, considering these additional specifications, the resulting DT model enhances (Eq. 5.1) to include also the shadowing $Shad$ and the history of states H in the DT definition:

$$DT = \langle PI, M, Shad, H, DI \rangle \quad (5.7)$$

In the following sections the abstract architecture will be refined to highlight how it can concretely be implemented to address challenges in DT engineering.

5.2 Modular Design of Digital Twin Interfaces

As introduced in Section 2.6, interoperability is a key requirement for DT development [AKP24, KAA24]. Considering interoperability as the ability to integrate

effectively with other system components, in the DT context, such ability can be considered twofold due to their cyber-physical nature.

Figure 5.2 represents a DT implemented following the abstract architecture presented in Section 5.1 in an exemplary deployment setting. It highlights the role of the DT as a bridge between *physical* devices and *digital* applications.

On the one hand, DTs must effectively integrate with a heterogeneous physical world, interacting with diverse IoT devices and systems that operate on different standards, protocols, and architectures. For example machines from different vendors may utilize different communication protocols and interaction patterns. Similarly, in healthcare applications, DTs must integrate various electronic health records across different data sources, from traditional repositories to sensor networks. This *physical* heterogeneity poses a challenge for the development of DTs, as, most often, it is unfeasible to modify the existing conditions and the DT system must be constructed on top of the legacy ones *adapting* to their requirements.

On the other hand, although DTs could work in isolation, they are increasingly being envisioned as part of complex software systems. Thus, DTs needing to integrate with different kinds of services ranging from system automation tools, to data platforms, machine learning pipelines and visualization tools, may expect to face several degrees of *digital* heterogeneity. As for the physical realm, the DT might need to *adapt* to the requirements of the digital ecosystem it is part of. Additionally, the modeling of complex CPS can be tackled through the idea of DTEs [RCM⁺22a] which envision multiple connected DTs, each modeling individual entities of the same domain. Such DTEs can benefit from standardized interactions between DTs and with higher-level services.

Tackling such twofold interoperability challenges requires on one side the adoption of common data models, semantic interoperability frameworks, and compliance with industry standards [ETS24] such as OPC-UA², oneM2M³, ISO-23247⁴, or the W3C Web of Things⁵. On the other side, the DT architecture should accommodate physical and digital heterogeneity *by design* to avoid being locked down in those situations where adherence to a single standard can not be enforced.

²OPC-UA: <https://opcfoundation.org/about/opc-technologies/opc-ua/>

³oneM2M: <https://www.onem2m.org/>

⁴ISO-DigitalTwin: <https://www.iso.org/standard/81442.html>

⁵W3C WoT: <https://www.w3.org/WoT/>

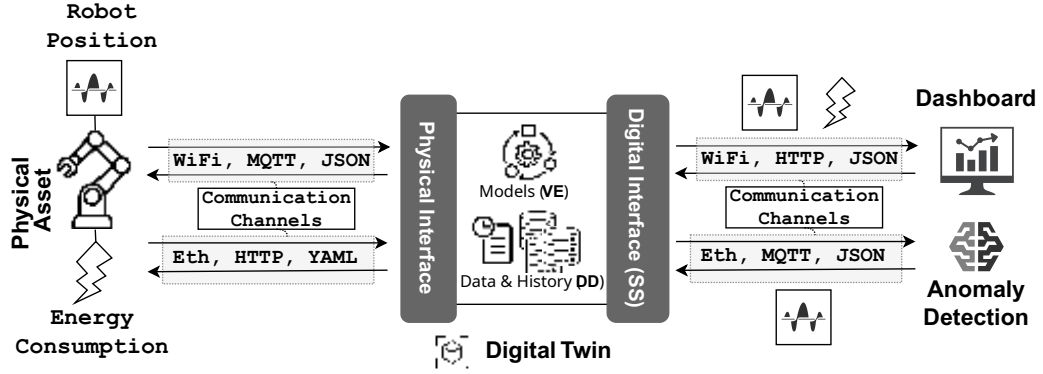


Figure 5.2: Example of a DT in an application scenario interacting with a physical asset and digital applications through its *PI* and *DI*. The example showcases the presence of multiple communication channels on both sides to highlight the heterogeneity of physical and digital interactions.

Without such measures, DTs risk becoming fragmented, limiting their ability to collaborate and exchange information across different platforms and applications. The proposal of modular DT interfaces aims to address these concerns at the architectural level, to enhance the interoperability and re-usability of DT components in cyber-physical applications.

At an appropriate abstraction level, a PA may be digitalized by obtaining and sending data through possibly *several* different **Communication Channels** (Figure 5.2), which encompass network protocols, data formats, and physical connections. The *PI* manages interactions with the PA, integrating such channels. Similarly, on the digital side, the DT supplies data and insights to applications via the *DI*, adapting to different protocols and representations.

Engineering the *PI* and *DI* requires addressing the challenge of effectively integrating such heterogeneous communication channels. To ensure flexibility and adaptability, it is essential that the core of the DT, including its models and implemented behaviors, remains decoupled from the heterogeneous physical and digital communication channels. The DT's shadowing process should focus solely on understanding the available physical data, receiving and transmitting telemetry information, and executing action requests, with no concern for the underlying implementation details.

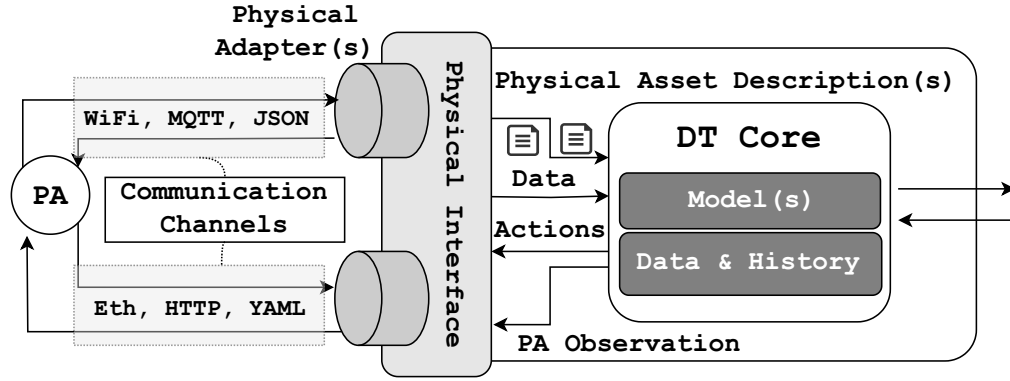


Figure 5.3: Physical Interface design with modular physical adapters each producing the corresponding Physical Asset Description that is processed by the DT Model.

The remainder of this section describes a proposal for the design the DT’s *PI* and *DI*, refining and extending the conceptual model originally proposed in [RCM⁺22a] with a concrete implementation based on the concept of modular *adapters*.

5.2.1 Physical Interoperability

One of the main challenges in facilitating effective communication through the *PI* is the wide variety of communication protocols and data formats used by PAs. While it can be the case that one PA is directly sending all the data concerning its digitalization through only one channel, it is far more common to build a DT aggregating different sources of information [QT18]. IoT protocols often differ considerably based on the manufacturer, device type, or application domain, making it difficult for the DT software to integrate diverse assets. Arguably, the modularity of the *PI*, which encapsulates these responsibilities, is a crucial factor in the design and implementation of interoperable DTs. Accordingly, the proposal is to design the *PI* as a composition of multiple *Physical Adapters (PhAs)*: specialized modules capable of interacting with the PA using diverse protocols and data formats. As shown in Figure 5.3, each PhA is responsible for mediating the bidirectional interaction through a single communication channel, simplifying

the design and reusability of the component, and making it configurable to adapt to different application contexts. The responsibility of the *PI* becomes then to manage the different PhAs and make sure the DT model can accurately interpret, process, and leverage the data generated by the physical world to create the digital replica and implement its behaviors. Note that although terminology of PhA is borrowed from [RCM⁺22a], where it is originally introduced, the *Physical Asset Adapter* proposed in [RCM⁺22a] is in fact a conceptual monolithic component, whereas in this concrete proposal a PhA is considered as a single-responsibility module of a potentially more complex *PI*.

Generating Physical Asset Descriptions

To facilitate managing different PhAs, the concept *Physical Asset Description (PAD)* is introduced: a representation of the capabilities provided by a communication channel in terms of *properties*, *actions*, *events*, and *relationships* that characterize the associated PA. Each PhA, since it encapsulates the characteristics of a channel, can generate the corresponding PAD, effectively decoupling the asset’s functionality from the specific communication protocols used. The implementation of the PAD generation can be challenging due to the varying capabilities of different communication protocols. For instance, protocols like CoAP [SHB14] often come with built-in description and discovery functionalities, which can simplify the process of creating a PAD by providing standardized representations of the physical asset’s capabilities and behaviors. On the other hand, protocols like MQTT [OAS19] may require developers to add additional information manually to define how information is structured and exchanged, as they do not natively include asset metadata. In the case of custom or proprietary protocols, the challenge becomes even more pronounced, as these protocols are tailored to specific systems and may lack any standardization or descriptive capabilities.

In all the aforementioned scenarios, the mechanism of PAD generation offers a way to encode knowledge about the PA bridging the gap by interpreting and extracting relevant information from the protocol used in the associated communication channels. Confining this complexity within the *PI* allows the DT model to be agnostic with regard to the complexity of the underlying physical world.

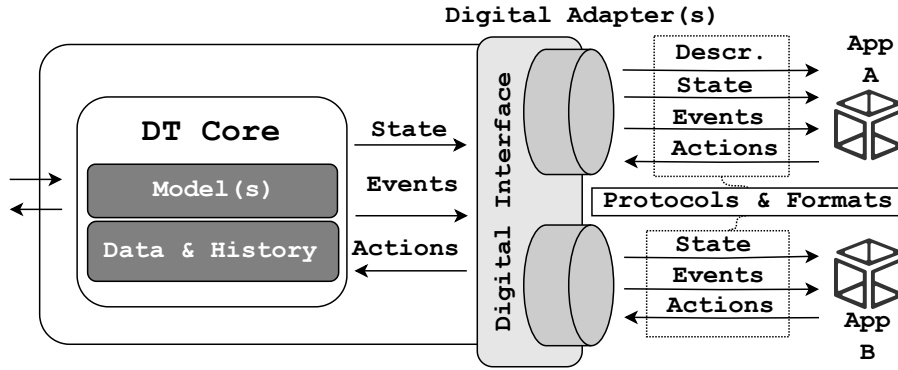


Figure 5.4: Digital Interface design with modular adapters. Each adapter publishes a description of the state, events and actions it supports so that consumers can dynamically understand and interact with the DT.

The PAD allows the *PI* to discover, extract, and manage asset information and present it to the model that can choose which ones are relevant for the implementation of the DT behavior. For example, to create the DT of a temperature sensor streaming binary data on MQTT, the *PI* could be composed of a generic MQTT adapter, configured to correctly parse the payload as a decimal number, and generate a PAD which advertise the available temperature property to the DT model as a Celsius value.

5.2.2 Digital Interoperability

As DTs are meant to bridge between the physical and digital spaces, interoperability is not only a matter of interfacing with heterogeneous devices, but also other digital applications. Indeed, despite their initial conception as vertical silos, the concept of DTE (Section 2.7) is emerging to support the digitalization of complex scenarios through a combination of several DTs. In this context, DTs can be used *as-a-service* by other applications implementing the business logic by spanning across different software entities.

To foster interoperability in DTEs, DTs must then expose either a standardized general purpose Digital Interface (*DI*) that can serve different applications or — following the same design principles adopted to address physical interoperability — have a modular interface that can satisfy the different needs of different applica-

tions as shown in Figure 5.4.

As for PhA the terminology is borrowed from [RCM⁺22a] and consider to have a *DI* composed of modular *Digital Adapter (DiA)*. The original abstract concept of DiA is hence refined to represent a modular component of a more complex *DI*. Using the concept of DiA, the *DI* can expose the DT state and services supporting multiple data formats and interaction patterns. This can be beneficial for integrating it with applications and, especially, legacy systems. The existence of legacy applications usually implies having little control over the integration requirements, making having more flexible DTs beneficial to better integrate with the application requirements.

Using several DiA, a DT could, for instance, support both request-response and publish-subscribe mechanisms to access its current and previous states, support different query languages to access the same data store, expose its current state using different representation formats, etc. This would make the development of the application simpler because adding an application-specific DiA won't require intervention in the underlying physical system. Additionally, the developed application would be more robust and stable since it would depend only on the DT, and changes to the physical configuration of the PA (e.g., software updates, sensor replacement, network reconfiguration) would have no impact on the application software. Even if the PA were to change, (e.g., a software update on a robot changes the telemetry format) the *DI* of the DT could stay the same, as the changes would occur within the boundaries of the *PI* and DT model. Through this mechanism, DTs effectively achieve their bridging role, shielding applications from the complexity of physical deployments.

Describing Digital Interfaces

A further level of interoperability is possible when allowing DTs to describe their *DI*, advertising capabilities and available communication channels that applications can exploit. This is especially relevant in contexts where applications are not bound to a specific interface but can instead process machine-readable descriptions of DTs and choose to interact with specific assets. A DT could then use a *Digital Twin Description (DTD)*, which, similarly to the PAD, can represent the features

of the DT to its observers. The idea of exposing DTDs is somewhat present in the major platforms supporting the creation of DTEs, such as Microsoft’s Azure Digital Twins⁶ and Eclipse Ditto⁷ and is advocated by standardization activities on interoperability in DTEs [ETS24].

The way such descriptions are implemented may differ significantly, but essentially, they should at least allow representing the DT features and APIs to access them. Notably, differently from the PAD, the DTD is targeted to external consumers, hence, it should preferably adhere to standard formats and representations to be useful in practice in achieving interoperability. To this end, using Semantic Web technologies (see Section 3.3) could be a possible solution to implement standard machine-readable DTDs.

5.2.3 Web of Things: enabling DT Interoperability

WoT [TLMM23] standards (Section 3.4) share the goal of adopting uniform API and data format descriptions to hide low-level complexities and promote interoperability with the proposal of self-descriptive PhA and DiA.

Despite their similarities, PADs and TDs serve distinct purposes. A TD provides a structured description of a physical twin’s available interactions to external consumers, detailing protocols and interaction possibilities. In contrast, a PAD is designed for internal use by the DT modules, decoupling the twin’s core from the complexities of communication channels. Its primary role is to facilitate the discovery of available resources and capabilities on the PA after establishing a connection with the physical counterpart, providing a structured description of its functionalities and data. This description is directly interpretable by the DT model and independent of the physical characteristics, allowing the reuse of the same model for similar PAs employing different communication channels.

In contexts in which WoT standards are applicable, the devices’ TDs can be automatically mapped to PADs, streamlining the DT creation process. Leveraging WoT TDs can significantly reduce the effort required to generate PADs, particularly in environments where numerous physical twins already have associated

⁶Azure Digital Twins: <https://azure.microsoft.com/en-us/products/digital-twins>

⁷Eclipse Ditto <https://eclipse.dev/ditto/>

TDs. This convergence between WoT and DT architectures has the potential to accelerate the development and deployment of DT solutions by promoting standardization and reusability. Nevertheless, the more general concept of PAD can be tailored also to those scenarios in which WoT is not applicable. In those cases the responsibility falls back to development (or configuration) of a PhA for a specific device to encode the necessary knowledge for the generation of the PAD.

WoT TDs can also be used as DTDs. The WoT architecture actually lists DTs as one of the possible deployment patterns, with a DT mediating the interaction with a PA behind a WoT interface⁸. This is especially useful when devices are not WoT compatible or can not be made so due to other constraints, essentially retrofitting the capabilities of the devices with a WoT compatible interface. Adhering to REST constraints [Fie00] and specifically to the HATEOAS and self-descriptive principles, a TD-based DTD facilitates the discovery of the DT model and services. Its machine-readable nature ensures interoperability for applications and consumers. In particular, a TD-based DTD facilitates the inclusion of DTs in WoT mashups, promoting DTs as valid WoT Things usable by WoT Consumers.

Despite its flexibility, and broad applicability, using of TD for DTs presents several limitations in capturing the specific characteristics of DTs that distinguish them from *Things*. A path to address these limitations could be extending TDs or supporting custom descriptions for DTs in DTEs as explored in Chapter 7.

5.3 Managing the Digital Twin Lifecycle

Recently, there has been an increasing recognition of the importance of the lifecycle of DTs, particularly in distinguishing the properties that define the relationship between the DT and PA. Key concepts such as *reflection* and *entanglement* are critical for accurately representing the PA [MLC20, RCM⁺22a]. These properties underscore the necessity for a structured lifecycle for the DT, ensuring that its state remains consistently aligned with the PA throughout various stages. As highlighted in recent surveys [FBB22, LPT⁺22, HCMM21], while the body of literature on DT software architectures is growing, most papers are domain-specific

⁸<https://www.w3.org/TR/2023/REC-wot-architecture11-20231205/#digital-twins>

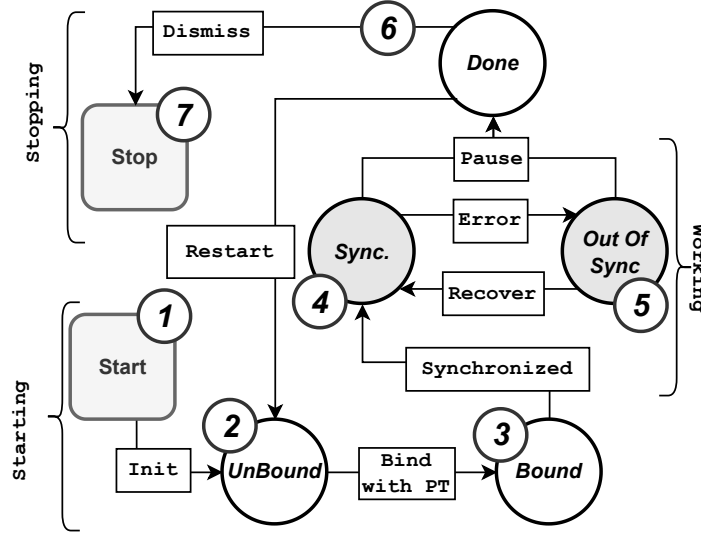


Figure 5.5: DT Life Cycle with its phases and transitions. Circles represent phases, rounded squares represent terminal phases, and arrows represented transitions. Phases are grouped in three major operating states of the DT which are **Starting**, **Working** and **Stopping**.

and focus on reference models. However, these models often lack concrete guidance on how to implement the internal components of a DT. Many of the existing models envision multiple parallel components or models working together, but fail to address how they communicate and interact to maintain consistency in the DT state. This gap leads to potential inconsistencies between the DT and PA, as the interaction between models and the management of state changes is not adequately captured [AE17, MSPM⁺19, L621].

5.3.1 A Digital Twin Synchronization Lifecycle

The concept of a DT lifecycle has recently been introduced and examined, focusing on its core components and the integration of both the software nature of the DT and its ability to synchronize with the PA over time [RCM⁺22a]. This lifecycle encompasses the various phases that a DT undergoes, as shown in Figure 5.5, from its creation to deactivation. Since a DT is fundamentally a software entity,

it is vital to track the evolution of such phases considering the different stages of synchronization with its PA. Proper observation and modeling of this lifecycle are essential to ensure that the DT representation can be trusted to reflect the physical counterpart's state consistently when the DT is **Synchronized** and can instead be considered to be outdated in the other phases.

The execution lifecycle of a DT can be modeled as follows. Upon creation (from the **Started** phase ①), the DT enters the **Unbound** phase ②, where all internal modules are initialized and prepared for the binding process with the PA. Once the DT successfully binds to its PA, meaning the PA meets the necessary requirements (expected functionalities and properties) and the DT can communicate with it, the DT enters the **Bound** phase ③. During this phase, the DT is connected to the PA and is ready to begin the shadowing process.

The next phase is the **Synchronized** phase ④, this is different from simply having established a connection with the PA, but rather indicates that the DT is complying with application-specific requirements and is receiving the necessary and correct amount of information to be fully aligned with its PA. During the synchronization phase the DT can compute the DT state (S_{DT}), ensuring that it is able to accurately reflects the status of its physical counterpart.

If any issues arise, such as network failures, and the DT synchronization performance falls under the expected requirements, the DT enters the **Out of Sync** phase ⑤. In this state, the DT—while still operational—is unable to update its state reliably. Eventually, the DT may return to the **Synchronized** phase.

Finally, when the DT is no longer required or has completed its function, it transitions to the **Done** phase ⑥. In this phase, the DT remains accessible to external consumers and retains its memory, but it is no longer bound to the PA or in sync with it. At the end of the lifecycle, the DT can be dismissed and moved to the **Stop** phase ⑦. The DT may also return to the **Unbound** phase if errors are detected during the binding process or if it is rebooted or restarted.

5.3.2 Binding the DT with the PA

The transition from the Unbound to Bound phase in the DT lifecycle remains an open research area: a key challenge is identifying whether all the capabilities of

the PA relevant to support the DT's behavior are available so that the shadowing process can start.

The modular design of the *PI* presented in the previous section (Section 5.2.1) can have support this transition. Namely, once a PhA successfully connects to the PA's channel and starts receiving data from it, it can send the generated PAD to the DT model. The generation of the PAD can be used as a synchronization step to signify that the PhA is successfully connected to the PA. The DT model is then responsible for collecting the different PADs, assessing whether all the relevant information to start computing the DT state is available and consequently moving on to the **Bound** phase. This mechanism enables managing the DT behavior consistently, even with the additional complexity of the modular *PI* design.

Of course the challenge still stands in managing each individual PhA connection, but this decoupling through PADs allows DT developers to establish functional binding requirements that are independent of technical details.

5.3.3 Managing DT and PA Synchronization

The most significant challenge in current DT lifecycle modeling is the broad characterization of the **Synchronized** phase, which only generically considers synchronization requirements between the PA and the DT, without having explicit awareness of the PA lifecycle and the potential changes in its operational context. This general and unstructured approach limits the ability to model the lifecycle in a consistent and uniform manner, failing to properly address the dynamism of the PA behavior.

The **Synchronized** phase of the DT's lifecycle assumes a continuous exchange of information between the DT and its associated PA usually measured to maintain a frequency within a given threshold, to guarantee that information on the DT is *fresh* and can hence be trusted as a valid representation of the PA state.

However, this general assumption may not always hold true due to variations in the PA's internal states. In some cases, PAs can enter operational states that temporarily inhibit their ability to communicate, even while maintaining an active connection to the DT. These are scenarios in which the absence of messages from the PA does not necessarily indicate a failure or misalignment but rather reflects

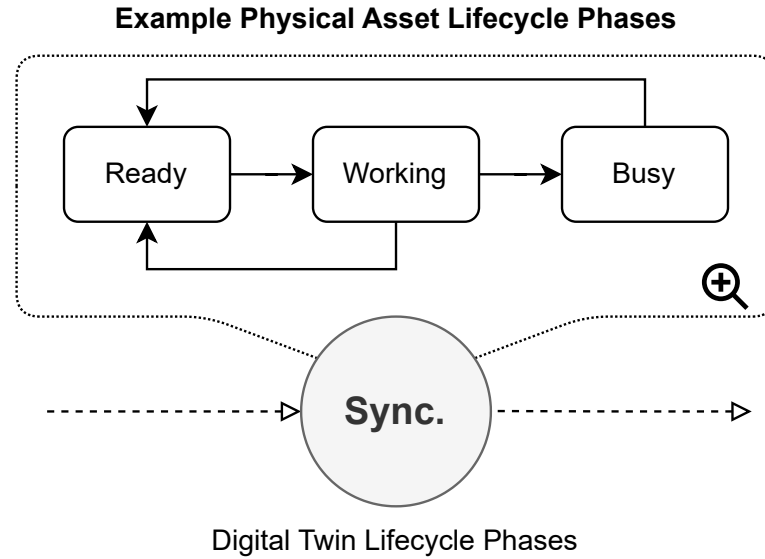


Figure 5.6: Zoomed view on the DT lifecycle with a structured example of the Synchronized phase from Figure 5.5. The image shows that within the Synchronized phase, the DT reflects the PA’s operational phases, such as Ready, Working, Busy.

the PA’s operational context. For instance, a PA might enter a **Rebooting** state during which it cannot send telemetry data. Similarly, resource constraints on the PA, such as *CPU overload* or *network bandwidth exhaustion*, may result in disrupted or paused communication that may or may not be tolerable for the DT model and application. In other cases, the PA may not entirely cease communication but instead alter its update frequency in response to operational changes. For example, an industrial robotic arm might increase its data transmission frequency during a high-precision assembly task to provide real-time feedback, while it could reduce the update rate to conserve energy and network bandwidth when in an idle or maintenance mode.

This adaptive behavior necessitates that the DT *dynamically adjust its expectations* and processing strategies based on the PA’s operational state, avoiding false-positive anomalies caused by *intentional* communication variability. By distinguishing between inhibited communication (e.g., **Rebooting** or **Overloaded**) and adjusted communication patterns (e.g., frequency scaling in **Idle** vs. **Active** states), the DT can better align with the PA’s lifecycle. This alignment ensures

the DT remains a reliable digital counterpart, accurately reflecting the PA's operational context and providing a robust foundation for decision-making.

To address these situations, the DT must incorporate mechanisms to:

- *Detect and Classify Non-Communication States:* Recognize when the lack of communication from the PA is due to a valid operational state (e.g., rebooting) rather than a system failure;
- *Model PA State-Dependent Communication Behavior:* Explicitly account for PA states that imply non-communication, such as *idle*, *rebooting*, or *overloaded*, within the lifecycle framework.

This requires extending the DT lifecycle model (Figure 5.5) to represent such scenarios accurately.

The proposal is to refine the **Synchronized** phase by introducing sub-phases that correspond to the PA's operational states (Figure 5.6). For example, in the industrial domain, a DT of a machine may enter the **Synchronized** phase but should then transition through more specific sub-phases such as **Ready**, **Working**, or **Busy**. Each of these sub-phases would have clearly defined transition criteria, phases, and associated S_{DT} , providing a more granular and accurate representation of the DT's behavior.

To formalize this approach, the overall DT lifecycle phase at a given time t $LP_{DT}(t)$ can be defined as a composition of the PA lifecycle phase $LP_{PA}(t)$ and the DT software lifecycle phase $LP_{Soft}(t)$, as shown in Eq. 5.8.

$$LP_{DT}(t) = \langle LP_{PA}(t), LP_{Soft}(t) \rangle \quad (5.8)$$

Here the focus is on representing an asset in terms of its present (and past) state, hence the DT representation R_{DT} is considered as a function of time. Recalling Eq. 5.3 the state includes all relevant properties, actions, events and relationships of the DT at a given time instant t_i . Other relevant aspects such as DT models and interfaces are not considered here for the sake of simplicity, but they can be easily included in the representation if needed, and they may evolve over time as well, depending on the lifecycle phase. For instance, actions in the DT state might be enabled or disabled depending on the lifecycle phase (e.g., in the

Done phase, no actions can be executed), and with those, also the DT interface may change to expose only the available functionalities.

The DT representation of the asset R_{DT} can hence be composed by the DT's lifecycle phase $LP_{DT}(t_i)$, its current state $S_{DT}(t_i)$, and its history $H(t_a, t_b)$ over a specific time interval (t_a, t_b) , as shown in Eq. 5.9.

$$R_{DT}(t_i) = \langle LP_{DT}(t_i), S_{DT}(t_i), H(t_a, t_b) \rangle \quad \text{and} \quad t_a, t_b \leq t_i \quad (5.9)$$

The values of the different components of the DT at the time instant t_i can then be mapped as a function of the different possible values of $LP_{Soft.}$, as detailed in Eq. 5.10.

$$R_{DT}(t_i) = \begin{cases} LP_{PA} = \emptyset, S_{DT} = \emptyset, H = \emptyset, \\ \quad \text{if } LP_{Soft.} \in \{\text{Started}, \text{Stopped}\}, \\ LP_{PA} = \emptyset, S_{DT} = \emptyset, H = H(t_f, t_o), \\ \quad \text{if } LP_{Soft.} \in \{\text{UnBound}, \text{Bound}, \text{OutOfSync}\}, \\ LP_{PA} = LP_{PA}(t_i), S_{DT} = S_{DT}(t_i), H = H(t_f, t_i), \\ \quad \text{if } LP_{Soft.} = \text{Synchronized}, \\ LP_{PA} = \emptyset, S_{DT} = \emptyset, H = H(t_f, t_d), \\ \quad \text{if } LP_{Soft.} = \text{Done} \end{cases} \quad (5.10)$$

When $LP_{Soft.}$ is **Started**, it represents the initialization phase where the DT software has been instantiated. At this stage, LP_{PA} is undefined since no communication with the PA has been established. Similarly, S_{DT} has not been computed, as the DT has not acquired any state information. The history H is also empty, as no synchronization or state updates have occurred yet.

When $LP_{Soft.}$ is **Unbound**, it indicates that the DT is operational but not yet synchronized. In this phase, LP_{PA} remains undefined because the PA lifecycle phase cannot be detected. S_{DT} is also undefined since the DT can not compute the state yet. However, the history H retains information about events recorded from the first connection data received from the PA t_f to the moment the DT

transitioned out of the synchronized phase, marked by t_o . The same happens when the DT is **Bound**, as the state has not been computed yet, or if the DT is **Out of Sync**, as the state can no longer be updated. Note that, since the DT history is computed only when the DT is synchronized, there might be cases in which the state at time t_i shows only a partial history and hence in Eq. 5.9 t_b may be different from t_i .

When LP_{Soft} is **Synchronized**, the DT is fully synchronized with the PA. In this state, LP_{PA} corresponds to the lifecycle phase of the PA (e.g., in the example of Fig. 5.6, this could be the **Working** phase) at the time t_i . At this point, S_{DT} contains the current computed state of the DT, reflecting the alignment between the DT and PA. The history H encompasses all events and states up to the current time t_i , documenting the DT's evolution in sync with the PA. If the DT operates correctly, it can remain in the *Synchronized* phase for the duration of its lifecycle, continually updating as the PA's lifecycle progresses. During this phase, multiple records of lifecycle evolution can be captured as LP_{PA} evolves (e.g., moving from **Working** to **Busy**), and multiple computations of S_{DT} may occur within the same LP_{PA} . For instance, while in the **Working** phase, the DT could compute multiple states corresponding to variations in physical properties such as accelerometer readings or energy consumption, ensuring the DT continuously reflects the PA's behavior in a dynamic and precise manner.

When LP_{Soft} is **Done**, the DT synchronization has been intentionally paused, and the DT remains active for accessing stored information. In this phase, LP_{PA} is empty, S_{DT} is empty, and H contains the history until the time the DT is decommissioned denoted as t_d .

Finally, when LP_{Soft} is **Stopped**, the DT lifecycle is terminated. LP_{PA} is empty, S_{DT} is empty, and H is also empty, as the DT is offline and data is no longer accessible.

By clearly defining and distinguishing between the various phases, the approach enhances the alignment between DTs and their physical counterparts, ensuring that each transition and state change is accurately captured and reflected. The main benefits of this approach include:

- *Enhanced Cyber-Physical Awareness*: a structured lifecycle model ensures

that critical transitions of phases and states of the PA are precisely tracked and understood, enhancing the overall awareness and responsiveness of the system;

- *Better Decision-Making*: with a clear understanding of each phase and its impact, applications and services can make more informed decisions based on accurate and timely information about the current relationship between PA and DT;
- *Adaptability*: this structured approach can be adapted to various applications, making it versatile and applicable across different domains.

The approach assumes that the L_{PA} is either provided by the PA or can be identified by the DT model (e.g., through a classification model). Modeling these phases can be challenging, especially for complex DTs. Despite improved cyber-physical awareness, phases may still be incorrectly mapped due to unforeseen external factors. Additionally, phase tolerance could impact anomaly detection within the DT, necessitating methodological trade-offs to set this tolerance correctly.

Despite these open challenges, the proposed structured lifecycle model represents an advancement in managing DT synchronization with their physical counterparts, providing a robust foundation for future developments.

5.4 Modeling Augmentation Functionalities

PAs are typically limited by the nature of their physical characteristics throughout their lifecycle. In contrast, DTs can capitalize on software-based flexibility to modify, update, and enhance functionality over time. These enhancements are accessible through DT APIs and can be powered by data-driven models possibly enabling adaptive and intelligent behaviors. This core property of DTs is known as *augmentation* [MLC20].

Examples of DT-enabled augmentation include adding technical features to retrofit devices through external software, such as enhancing interoperability of IoT devices via protocol and data format translation. Another example involves improving security and addressing unresolved bugs by mediating interactions with

outdated devices that no longer receive updates. Additionally, DTs can enhance device services by introducing new high-level actions for operations, or by implementing advanced functionalities powered by machine learning and AI algorithms, such as forecasting, anomaly detection, and pattern recognition (Section 2.4).

Despite the acknowledgment of augmentation as a defining feature of DTs, this property is often defined only conceptually, with implementations tailored to specific vertical cases. The lack of a reference model for defining DT augmentation results in unclear design patterns and potential performance limitations. Indeed, without a well-defined and measurable boundary for augmentation features—including resource allocation—the DT risks failing to achieve its design goal of accurately representing the PA with sufficient *fidelity* [BBF⁺23].

5.4.1 Extending the DT Model with Augmentation

The DT model introduced in Section 5.1 is here extended to explicitly include augmentation functionalities. In practical terms, augmentation serves as a bridge transforming raw data into meaningful, actionable outcomes guided by the underlying model. DTs may expose additional services and functionalities that enhance the capabilities of the PA. Such functionalities can be represented as a set of *augmentation functions* F , where $Augm_i$ denotes the individual augmentation functions implemented within the DT. Each augmentation function $Augm_i$ may leverage a specific model $m \in M$, to interpret and respond to changes in the DT.

$$F = \{Augm_1, Augm_2, \dots, Augm_n\} \quad (5.11)$$

The model in Equation (5.7) can hence be extended to:

$$DT = \langle PI, M, Shad, F, H, DI \rangle \quad (5.7)$$

Augmentation functions can be distinguished based on two aspects:

Triggering mechanism (τ): defines how and when the function is activated in response to events within the DT.

$$\tau \in \{e_{DT}, e_{Aug}, e_{Time}\} \sigma \in \{S_{aug}, \emptyset\} \quad (5.12)$$

τ can be an event of three types:

- e_{DT} represents the output from the shadowing process indicating a new computation of the DT state S_{DT} ;
- e_{Aug} can be generated either as the output of another augmentation function or as a direct invocation by the shadowing process;
- e_{Time} refers to temporal triggers and can be used to model periodic execution or specific scheduled tasks.

Notably, augmentation functions can not respond to triggers coming directly from the PI as such e_{PA} . Physical events need to be processed by the shadowing process first, which may in turn select the activation of a relevant augmentation function firing a specific e_{Aug} .

State management (σ): indicates whether the function maintains an internal state that persists over time or is *stateless*.

$$\sigma \in \{S_{aug}, \emptyset\} \quad (5.13)$$

A stateful function possesses a state S_{Aug} that evolves and persists over time in alignment with the evolution of the DT. A stateless function simply reacts to triggers without maintaining any memory of previous invocations. Notably, modeling stateful functions allows for the representation of processes that may even run in parallel with the DT main control flow, producing new events e_{Aug} asynchronously. Since the function is stateful, it is triggered to start but can persist over time, as its execution is fully encapsulated within the function context. Additionally, the function may receive new triggers over time that feed into its ongoing process.

The output of an augmentation function $Augm_i$ is determined by its inputs and can be expressed as:

$$Augm_i(\tau, m, \sigma, H) = \begin{cases} e_{Aug} & \text{if } m \text{ matches} \\ \emptyset & \text{otherwise} \end{cases} \quad (5.14)$$

The triggering event τ may carry important information for the function execution. The parameter $m \in M$ represents the model used within the augmentation function, encapsulating the rules, logic, and conditions that define how the function interprets its inputs and determines the output. The model is crucial for evaluating whether the function should generate an output based on the current state of the function σ and context, which is represented by the DT history H which includes the latest state S_{DT} .

The output e_{Aug} represents an event generated by the augmentation function, signaling that the conditions defined by the model m have been met. Conversely, if the conditions are not satisfied (i.e., if m does not align with the current state or inputs), the output will be $\emptyset$, indicating that no new event has been produced. This approach captures the idea that the execution of an augmentation function does not always result in the generation of a new event. For example, in the case of an anomaly detection model, an event is only triggered when an anomaly is detected with a certain confidence threshold, meaning that no event is generated if the conditions for an anomaly are not met. Since $e_{Aug} \in \tau$, augmentation functions can also be chained to model processing pipelines triggered by the outputs of previous functions.

5.4.2 Impact on Shadowing

The introduction of augmentation functions requires to adapt the shadowing process to handle augmentation events. Indeed, the shadowing process should be capable of receiving relevant augmentation events (e.g., detected anomalies) and use them to update the state of the DT, as illustrated below:

$$e_{DT}(S_{DT}) = Shad_{DT \rightarrow DT}(M, e_{Aug}, H) \quad (5.15)$$

The same principle applies when a physical event triggers an augmentation function. There is no direct link between physical world events (e_{PA}) and augmentation functions; instead, the decision to trigger an augmentation function based on the receipt of a physical event is mediated by the shadowing process. The shadowing process analyzes the e_{PA} and determines whether it is relevant for an augmentation function in F (or multiple functions simultaneously). If the event

is deemed as relevant, the shadowing process generates a trigger by producing an augmentation event (e_{Aug}), which then serves as the trigger τ for one or more augmentation functions.

$$e_{Aug} = Shad_{PA \rightarrow DT}(M, e_{PA}, H, F) \quad (5.16)$$

Similarly, actions requests that are received by the DT (a_{DT}) can also trigger augmentation functions. Again, there is no direct relationship between the action taken on the DT, as represented by its state, and the execution of the augmentation function. Instead, this process also goes through the shadowing process, which matches the request received on the *DI* with the corresponding augmentation function trigger, thus generating an augmentation event that will serve as the trigger for the augmentation function. This can be used to model a request-response pattern for the execution of augmentation functions such as requesting the predicted next state of the DT. In such cases, since the result of the augmentation does not impact the state of the DT, the result can be directly routed to the external consumer of the DT.

$$e_{Aug} = Shad_{DT \rightarrow DT}(M, a_{DT}, H, F) \quad (5.17)$$

Despite the shadowing might not be required to validate the result of an augmentation function triggered by an action request, it is still enforced to handle the incoming requests. This design aims to decouple the available implementations of the augmentation functions from the actions presented on *DI* and handle this transparently from the consumer perspective. For example, there might be several augmentation functions implementing a given functionality and it would be up to the shadowing process to determine which one is best suited to respond to a specific request depending on the current context of the DT (e.g. the state prediction functionality could be implemented using different specialized models depending on some conditions in the state of the DT).

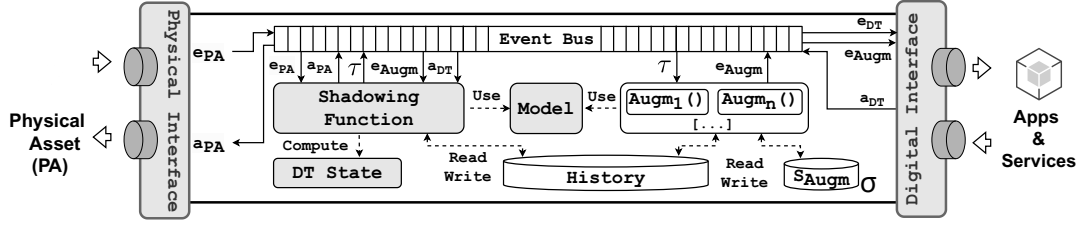


Figure 5.7: Schematic Event-Driven architecture of a DT to support Augmentation function management and execution. A central event bus supports asynchronous interactions between the shadowing and augmentation functions. Functions can access the DT models and history as needed.

5.4.3 Engineering Augmentation Functions

Engineering augmentation functions following the proposed model requires defining software architectural patterns that can be practically applied to implement them. Depending on the implementation, it is possible to define *internal* and *external* augmentation. This setup allows for the dynamic addition of new functionalities to the DT, either through external or internal processing.

As Figure 5.7 shows, the core architecture of the DT is centered around a shared **Event Bus**, which serves as the communication backbone, handling events and actions within the DT system. Starting from the *PI*, physical events e_{PA} are processed by the **Shadowing Function** (*Shad*) which computes the DT State S_{DT} based on the received events and the DT Models (M), ensuring that the DT accurately reflects the current state of the PA. The DT State component maintains this state in memory, while the **History** (H) component persists past events and states for reference and analysis.

Augmentation Functions ($Augm_i$) provide additional capabilities or insights to the DT, interacting with M , H , and internal state information σ . The interaction between the *Shad* and the $Augm_i$ is entirely event-driven, based on trigger events τ and e_{Augm} flowing out of the augmentation functions with their results. These results are then sent to the *Shad* for further state computation and to the **Digital Interface** (*DI*) to respond to requests for $Augm_i$ from external services and applications. The *DI*, mirroring the *PI*, handles outgoing events of the DT (e.g., e_{DT} carrying the new S_{DT} or its variations) or e_{Augm} with the results of an augmentation function, and manages incoming actions and requests from digital

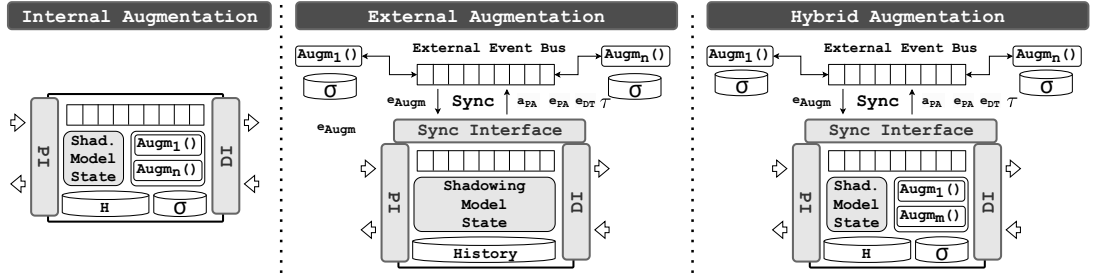


Figure 5.8: Patterns for augmentation functions execution: internally (left), externally (center), or through a hybrid solution (right).

entities as digital actions a_{DT} .

Based on this event-driven model, it is possible to identify three approaches for implementing augmentation functions (Figure 5.8):

- *internal* augmentation, where functions are executed within the DT process;
- *external* augmentation, where functions are executed outside the DT process by means of an external service;
- *hybrid* augmentation, where functions are executed through a combination of internal and external processes.

Internal Augmentation

Augmentation functions are executed *internally*, i.e. within the same operating system process as the DT, thus sharing the same resources allocated to the DT instance (e.g., CPU or GPU computation capabilities). From an external perspective, these functions are perceived as internal functionalities of the DT and only the internal event bus of the DT is used.

This approach ensures low latency and tight integration with the DT core functionalities, as overhead between the DT and the augmentation functions is minimized. For instance, in a manufacturing scenario an internal augmentation function could periodically monitor the operational parameters of a machine physical counterpart. By running within the DT process, this function can quickly react to changes in the machine state, such as detecting abnormal vibrations that might indicate a potential failure (e.g., using signal processing approaches). The

internal event bus would handle the real-time data stream from the machine sensors, feeding the computation of the DT state while allowing the augmentation function to promptly process this data and update the DT.

External Augmentation

Augmentation functions run *externally* as processes entirely separate from the DT, possibly without any knowledge of the DT internal model. Since the functions are implemented on external processes, they can use inter-process communication to synchronize and hence they can be distributed across different computing nodes and resources, for a more fine-grained allocation. This translates either in multi-process deployment of the DT on the same host, or possibly in fully distributed deployment of the DT on a computing cluster.

In the latter scenario, multiple event buses are present. The primary event bus will always be the internal one, while additional external event buses, possibly implemented with different technologies, will be integrated with the internal one. Such integration allows the system to provide triggers, events, and information to the augmentation functions and retrieve their results to synchronize them with the DT logic in a completely transparent manner, both for the PA and external applications.

A *Synchronization Interface (Sync)* within the DT will be responsible for synchronizing the internal and external event buses concerning the events and triggers associated with the augmentation functions. This component must be capable of managing the protocols or formats for mapping between the logic of the internal event bus and the external one.

Since augmentation functions can be external, the state (σ) of an augmentation function will be maintained externally to the DT. For internal augmentation functions, σ will be internal to the DT, but isolated from other DT components and solely dedicated to the augmentation function.

Taking as reference a manufacturing use case, the value of such externalization can be exemplified by considering a DT monitoring various machines on the production line, collecting real-time data on operational status and performance. Augmented features for maintenance prediction and production schedule optimizations

could be delegated to an external data analytics service which, using the external event bus, may process the DT data with advanced machine learning algorithms and send back results to the DT without overloading the DT core system.

Hybrid Augmentation

Augmentation functions run both internally and externally, by combining the strengths of both methods. A synchronization interface ensures communication between the internal and external event buses. Stateful functions can store their state both externally, for those that require external resources, and internally, as a structural element of the DT for internal functions.

This approach is the most flexible and offers significant advantages. For example, in a smart city application, internal augmentation functions can handle real-time traffic monitoring, benefiting from low latency and immediate response times. Meanwhile, external augmentation functions can process long-term traffic pattern analysis for strategic urban planning, leveraging powerful external computational resources without burdening the DT core system. Such hybrid model provides the flexibility to optimize for both real-time and complex, resource-intensive tasks, ensuring that the DT remains efficient and scalable.

5.5 Extending the White Label Digital Twins Framework

The concepts introduced in the previous sections have been realized through a substantial extension and architectural refactoring of the White Label Digital Twins (WLDT) framework, originally introduced in [PMZ21]. Rather than presenting WLDT as a new contribution, this section documents how the framework has been redesigned and evolved the DT engineering principles and lifecycle requirements introduced above.

WLDT is an open-source Java framework available on GitHub⁹, conceived to support the development of DTs with a focus on IoT-enabled applications. The work presented here extends its features by introducing explicit support for

⁹<https://wldt.github.io>

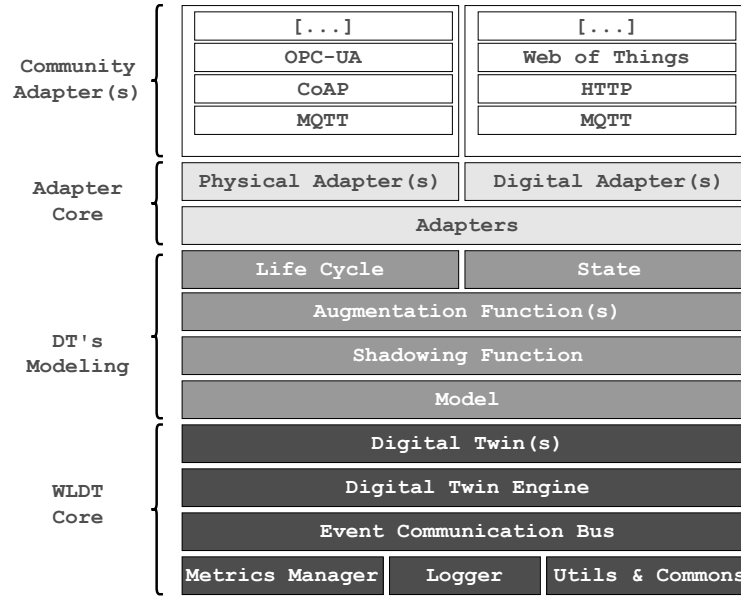


Figure 5.9: Extended WLDT software architecture aligned with the DT engineering model proposed in this thesis.

DT lifecycle management, separation of shadowing and augmentation concerns, and modular interfaces aligned with distributed and microservice-oriented deployments. The extensions are designed to use WLDT as a concrete implementation of the abstract DT architecture and lifecycle model proposed in this thesis, serving as a building block for the development of DTEs.

The redesign of WLDT has been guided by three main requirements derived from the thesis contributions:

- **Simplicity**, to support the rapid engineering of DTs through reusable modules and configuration-based customization;
- **Extensibility**, to enable the concurrent execution of multiple behaviors and the incremental evolution of DT capabilities;
- **Portability and microservice readiness**, to allow DTs to be deployed as independent computational entities across heterogeneous infrastructures.

Architectural refactoring. The extended WLDT architecture, shown in Figure 5.9, is organized around four conceptual layers: a core execution library, a modular DT model, a core shared adapter logic, and a set of community-developed physical and digital adapters. This structure directly reflects the abstract DT architecture introduced in Section 5.1, providing a concrete realization of its main concepts. The core library provides execution control, lifecycle coordination, and internal event-driven communication mechanisms. Supporting services such as logging, metrics, and common utilities improve observability and robustness of DT instances.

Digital Twin core and lifecycle support. A key contribution of this work is the refactoring of the DT core into an explicit *Model* component that governs the evolution of the DT over time. This model controls both the *Shadowing Function* and one or more *Augmentation Functions*, making their roles explicit and separable. The Shadowing Function is responsible for maintaining alignment between the physical asset and its digital counterpart by processing data received from physical adapters and action requests issued through digital adapters. It manages the DT *state* in terms of properties, events, actions, and relationships, and drives the transitions between lifecycle phases as introduced in Section 5.3. The state management component is designed to support transactional updates and maintain a history of state changes, enabling both real-time synchronization and retrospective analysis. The shadowing function determines when a DT can be considered **Bound** and when it reaches a **Synchronized** state suitable for higher-level reasoning and interaction. Augmentation Functions encapsulate additional behaviors that go beyond pure mirroring, such as data aggregation, anomaly detection, or domain-specific processing. This separation enables the incremental enrichment of DT capabilities while preserving a clear boundary between physical synchronization and added intelligence.

Adapters as interaction boundaries. WLDT adopts a modular adapter-based approach to interface DTs with both the physical and digital worlds. Physical Adapters handle the ingestion of data and commands from heterogeneous physical assets, while Digital Adapters expose DT state and capabilities to ex-

ternal consumers through application-specific protocols. Adapters are managed as first-class components with dedicated lifecycles and unique identifiers, allowing multiple adapters to coexist and be coordinated within the same DT instance. Digital Adapters can access both the current DT state and its history, enabling the exposure of richer digital services without embedding protocol-specific logic into the DT core.

Execution and scalability considerations. The extended WLDT framework includes a multi-threaded execution engine capable of managing multiple DT instances within the same runtime. While the current implementation targets single-process execution, the abstraction is intentionally aligned with distributed and microservice-based deployments, paving the way for future extensions where DTs may be deployed across multiple processes or nodes within a DTE.

Overall, the extensions presented in this section transform WLDT from a framework primarily aimed at individual IoT-focused DTs into a more comprehensive framework that explicitly supports the engineering principles and lifecycle management strategies that separate responsibilities of a DT across modular and reusable components.

5.6 Evaluation of the Proposed Approach

In this section, the proposed architectural approach for engineering DTs is evaluated through the implementation of an industrial scenario. This showcase demonstrates the practical application of the concepts introduced in the previous sections and highlights the benefits and challenges of the approach in a real-world context.

5.6.1 Industrial Scenario Description

As a reference scenario, a manufacturing setting where DT are deployed to monitor and manage a production line is considered. The production line is realized through the *Fischertechnik Training Factory Industry 4.0*¹⁰, controlled by PLCs

¹⁰Fischertechnik Industry & Universities: <https://www.fischertechnik.de/en/products/industry-and-universities>

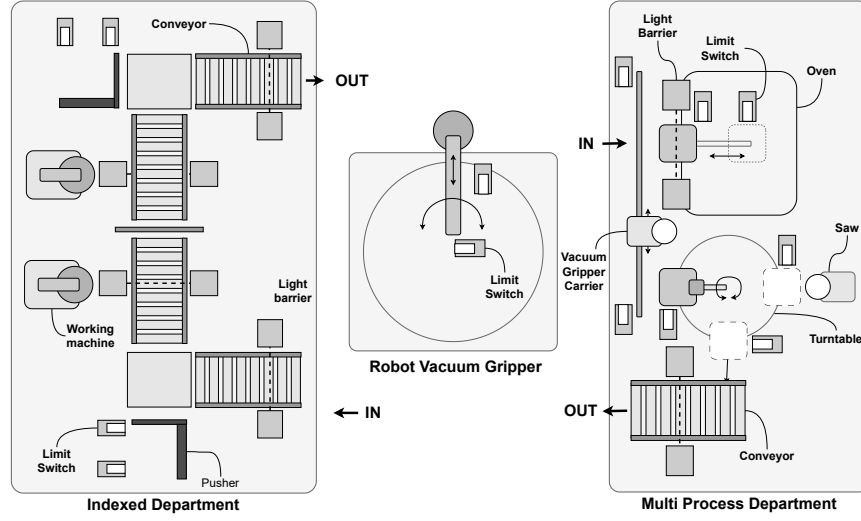


Figure 5.10: A schematic view of the microfactory production line, with the two departments.

that publish real-time data via OPC-UA and integrated with Arduino RP2040 boards¹¹ for accelerometer data collection and processing and communicating over MQTT. The schematic representation of the microfactory production line is shown in Figure 5.10. This microfactory setup provides a controlled yet dynamic environment for testing and refining DT technologies in a realistic industrial context.

In this scenario, pieces are processed along a production line, passing through multiple machines. The first station, referred to as the *Multiprocess Station with Oven*. As shown in Figure 5.10 (right), the module consists of five machines arranged in a flow shop layout: three material handling units (vacuum gripper carrier, turntable, and conveyor) and two transformation stations (oven and saw station). Each machine is equipped with sensors (light barriers and limit switches) and actuators (with two or three operational states). The second station, named the *Indexed Line* (Figure 5.10, left), comprises four conveyor belts, two of which include mechanical processing operations for the workpieces.

The goal of the scenario is to implement, through DTs, a system with the following desired features:

- monitor the state of each machine in the production line;

¹¹Arduino RP2040: <https://docs.arduino.cc/hardware/nano-rp2040-connect/>

- track production performance, including the number of pieces processed and the efficiency of each machine;
- monitor power consumption of each machine to optimize energy usage;
- regulate the production speed of each machine to ensure smooth operation and avoid bottlenecks;
- compute the overall production performance of the entire production line;
- compute the overall energy consumption of the entire production line;
- manage external action requests, at the system level, to adjust production parameters or respond to specific events;

5.6.2 Digital Twin System Architecture

To achieve the desired features, a multi-layered architecture is proposed, consisting of machine-level, department-level, and factory-level DTs. Each machine-level DT processes incoming data on properties and events to generate insights such as production and energy KPIs, while also managing action requests to regulate their individual speed. Additionally, two department-level DTs are created, matching the two different goals of tracking energy consumption and production performance. All DTs are built using the WLDT framework.

The production efficiency is computed at the machine level using the Overall Equipment Effectiveness (OEE) metric¹² calculated from production rate, uptime, and downtime. At the department level, DTs representing the Indexed Departments assess performance by computing Weighted OEE [GGLR17, Nak95], derived from individual machine DTs, and track throughput, defined as processed pieces per unit time. Throughput is determined by monitoring entry and exit events within the department, utilizing relationships captured by the associated DTs.

Figure 5.11 shows the resulting DT system architecture for the *Multiprocess Station with Oven*. Each machine of the station is represented by a dedicated DT which connects to the physical asset through a PhA that interacts with the PLC

¹²OEE: <https://www.oee.com/>

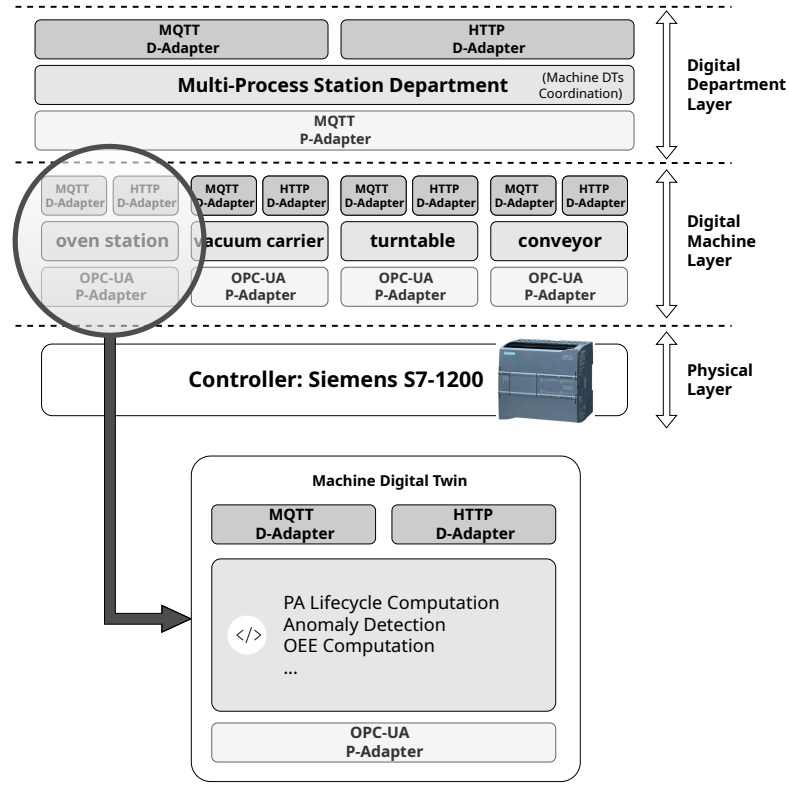


Figure 5.11: Multiprocess Station Digital Twin architecture overview.

iva OPC-UA. At the department level, the DT of the *Multiprocess Station with Oven* aggregates data from the machine-level DTs, computing overall performance metrics and exposing department-level actions.

Figure 5.12 illustrates the relationships among the various DTs in the Multiprocess Station. Relationships between machines and DTs (e.g., *previous*, *following-machines*, *is-part-of*) are statically defined in this setting and used by department-level DTs to track and compute throughput and total pieces processed.

Similarly, Figure 5.13 illustrates the DT ecosystem architecture for the *Indexed Line Department*. Again, each machine is represented by a dedicated DT connected to the physical asset through PhAs. In this setup, machines integrate the communication of the PLC via OPC-UA and Arduino boards via MQTT. Finally, a factory-level DT aggregates data from both departments, computing the overall production performance and energy consumption.

Machine Level DTs expose a digital interface through DiAs. An MQTT digital

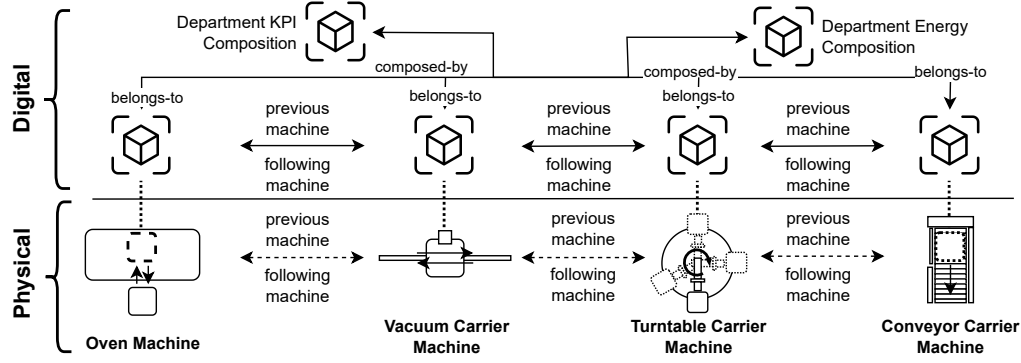


Figure 5.12: Relationships between DTs in the Multiprocess Station serve the purpose of tracking the physical relationships between different machines at the digital level.

interface is used to connect machine-level DTs to the department-level DT, enabling composition between DTs in a hierarchical structure. All DTs also expose an HTTP digital interface which serves as the primary access point for external applications and services. The HTTP interface can further be described using WoT TDs, enabling seamless integration with other WoT-compliant systems and applications as discussed in Section 5.2.3.

5.6.3 Reducing Complexity through Reusable Adapters

Notably, the physical and digital adapters that implement MQTT, OPC-UA, and HTTP protocols, along with WoT over HTTP for serving TDs, are reusable across different instances. These adapters, through configurable parameters, enable effective communication and integration within both digital and physical environments.

Upon startup, each DT connects to its data source, publishes the PAD of the physical asset, and interacts with the PLC or Arduino to collect and send data, enabling digitalization and interoperability among heterogeneous equipment.

Figure 5.14 schematically illustrates the evolution of data and representation from the physical to the digital realms through the adoption of the proposed DT architectural approach, PAD integration, and WoT TDs used as DTDs on the digital side, specifically for the output conveyor of the Indexed Department production line (Figure 5.13). On the physical side, data is structured using OPC-

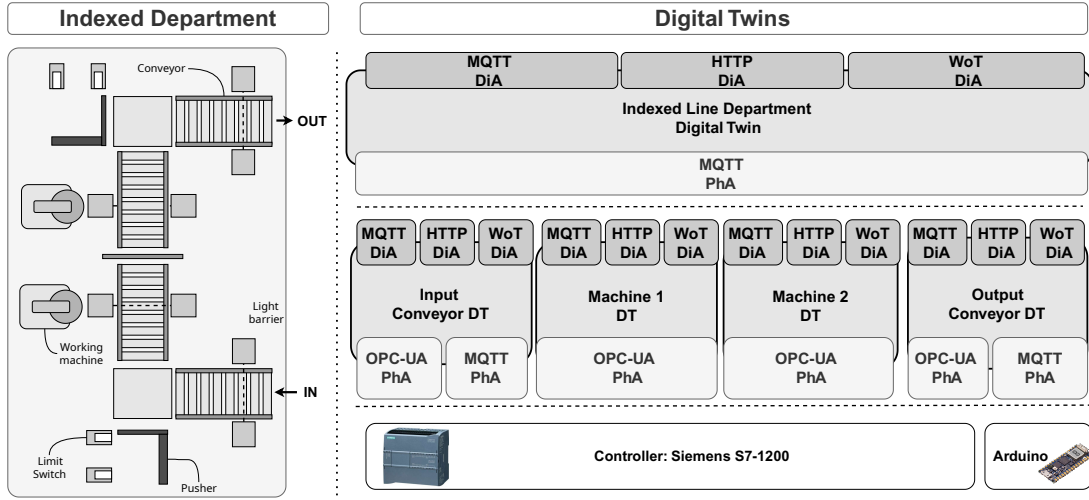


Figure 5.13: The Digital Twin ecosystem architecture of the Indexed Department, showing the different Physical Adapter (PhA) and Digital Adapter (DiA) of each DT alongside a schematic representation of the conveyors.

UA with a hierarchical organization, containing telemetry data and action methods mapped to specific data types. Accelerometer data is transmitted over MQTT on a specific topic with a JSON payload containing axis acceleration values.

The PADs act as an intermediary, translating data from the PT into a format suitable for the DT core, decoupling it from the complexity of interacting with the Physical Twin (PT) and enabling discovery, understanding, and utilization of available data and methods. Using the information from the PADs, the DT model computes the DT's state with target properties, events, and actions, based on either one-to-one matching of physical characteristics or augmented by combining multiple physical properties into computed DT properties or events, such as computing the value of the OEE property or triggering anomaly detection events based on accelerometer data.

Finally, the DiA of the DT leverages WoT TD to describe the DT's data and functionalities through a standardized, interoperable, and machine-readable approach. The TD specifies how these can be accessed via protocols and data formats. This approach enables effective interoperability from the physical to the digital, using a modular and decoupled structure through DTs in the target industrial use case.

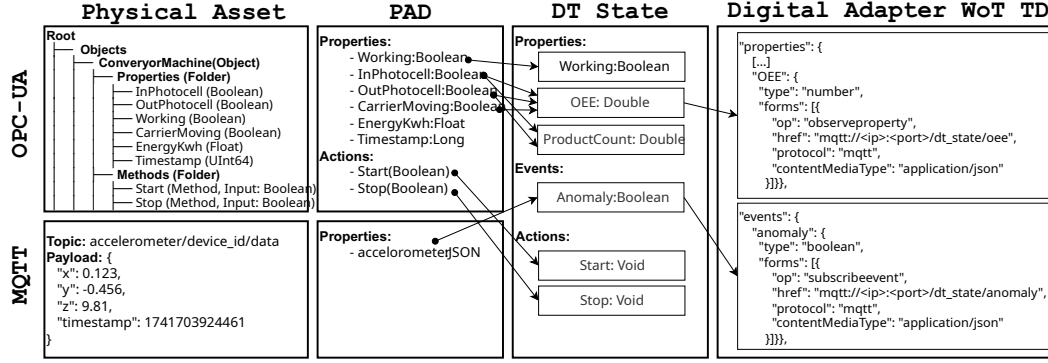


Figure 5.14: The transition from physical to digital representation via the PAD, the DT State, and the DTD for external consumers.

This deployment aims to measure the impact of digitization and responsibility decoupling by comparing the system complexity in scenarios with and without DT adoption and also with respect to the different DT's architectural layers. This allows to assess the benefits of a modular, structured DT-based approach, particularly in terms of interoperability and heterogeneity management.

To quantify how effectively the proposed approach of PhA and DiA manages cyber-physical heterogeneity, the *Cyber-Physical Complexity Index (CP-CI)* proposed in [LMPZ23, LPM⁺24] is adopted as a measure of complexity of the system.

The CP-CI quantifies the perceived complexity of a cyber-physical application based on:

- *Required Protocols (p)*: communication protocols needed for device interaction;
- *Communication Patterns (c)*: interaction models (e.g., Publish/Subscribe, Request/Response);
- *Data Formats (d)*: required data representations or serialization methods;
- *Interaction Points (n)*: modules or services involved in data exchange; and
- *Aggregation Points (a)*: levels of abstraction for physical data (e.g., merging data from multiple machines).

Each criterion is weighted with a *Criteria Importance Factor (CIF)* from 1 (low) to 3 (high), indicating its impact on development, deployment, and maintenance.

Focusing on DT interoperability, the CP-CI is extended to include both inbound and outbound interfaces, essential for bridging the physical and cyber layers with differing requirements. This extension enables a more precise assessment of the complexity in managing interoperability across system boundaries. The CP-CI was applied not only to the entire DT but also to its internal layers: *PI*, *Core*, and *DI*. These results were compared to those of a monolithic external application that directly handles business logic and interoperability, bearing the full complexity, highlighting the advantages of the modular DT design. High importance ($CIF = 3$) was assigned to managing heterogeneous data formats (d) and aggregation points (a), as they are crucial for creating interoperable data models. Medium importance ($CIF = 2$) was given to protocol diversity (p) and interaction with multiple entities (n), which become more significant as systems scale. Low importance ($CIF = 1$) was assigned to multiple communication patterns (c), as their concurrent use is standard in distributed applications.

The graphs presented in Figure 5.15 display the CP-CI values obtained across different levels of deployment: a single machine, the indexed department, and the entire factory. For the machines analyzed, the involved protocols (p) and communication patterns (c) are both set to 2, as MQTT and OPC-UA are used with Pub/Sub and Request/Response interactions. On the digital side, both MQTT and HTTP are employed. Regarding data formats (d), each machine manages 2 distinct formats, accounting for both PLC data and Arduino accelerometer information. Furthermore, the interaction points (n) and aggregation points (a) are both 2, reflecting the aggregation and interaction with two different sub-entities for each machine and its corresponding DT.

The results for single machines (Figure 5.15a) highlight the critical role of the *PI* in the DT architecture. As a decoupling layer, the *PI* isolates physical-world complexities from the DT core, managing priorities like protocols, data formats, interaction points, aggregation strategies, and communication patterns. This allows the DT core to focus on uniform data, processed through the interface via PAD and payload transformations, without dealing with heterogeneous physical entities. Consequently, the core interacts only with its internal communication

5.6. EVALUATION OF THE PROPOSED APPROACH

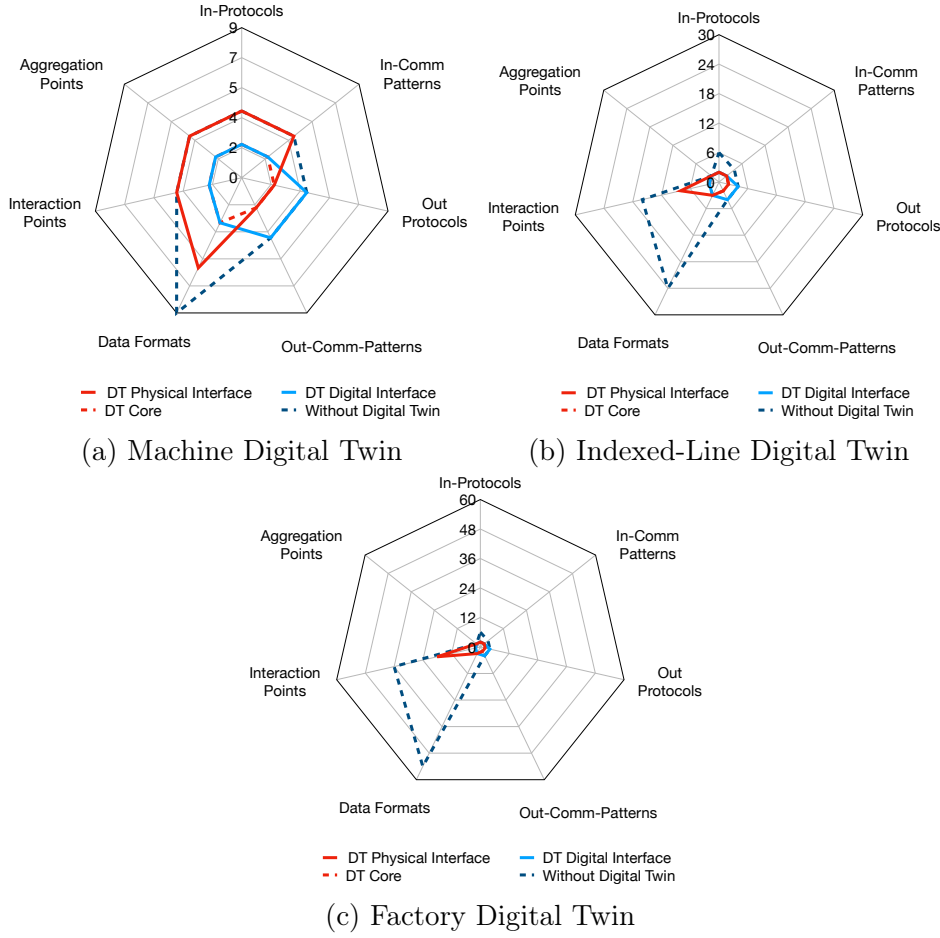


Figure 5.15: Cyber-physical complexity computed on different levels of the hierarchical DT system implemented for the microfactory, showing reduced complexity compared to an approach without modular DTs interfaces.

pattern and consistent interaction points, regardless of the physical environment's configuration. A similar approach is applied to the *DI*, which is decoupled from physical complexities and adapts to external digital systems, ensuring modularity and reuse.

In contrast, systems without a DT must embed all heterogeneity management into a single application, burdening it with business logic, protocol diversity, and data format interoperability. This increases complexity and reduces scalability, as reflected by the complexity index parameters.

As system architecture scales from individual machines to departments (Fig-

ure 5.15b), or entire factories (Figure 5.15c), protocol diversity stabilizes while the number of interaction points and data formats increases due to the need for managing heterogeneous components. The complexity measure shows that systems at the department or factory level experience a significant rise in complexity, driven by interaction points, aggregation nodes, and data format diversity.

These results emphasize the benefits of the modular and interoperable approach presented in section 5.2. By encapsulating complexity within reusable *adapters* and using interoperable representations, developers can simplify application development. In contrast, monolithic systems struggle with managing interoperability and system evolution as integration requirements grow.

5.6.4 Lifecycle Awareness and Coordination

To demonstrate the applicability of the proposed lifecycle synchronization model introduced in section 5.3, each machine-level DT handles the synchronization between the LP_{PA} and the corresponding LP_{DT} . Furthermore, an anomaly detection algorithm runs within each DT, enabling it to identify and report issues that the PLC alone might not detect, thus dynamically computing the LP_{PA} when necessary (e.g., when a piece does not reach a reference photocell on the machinery within a predetermined time and an anomaly is consequently detected).

The department-level DT of the Multiprocess Station incorporates coordination logic that adjusts machine behavior based on their current phases, ensuring seamless station management.

The machine-level DTs operate based on data produced by their physical counterparts. Leveraging these temporal data, it is possible to model the lifecycle of the physical object within the DT lifecycle as shown in Figure 5.16. When the DT lifecycle reaches the Synchronized state, it accurately mirrors the corresponding phase of the physical machinery's operation. These phases are inspired by the reference structure for industrial machine phases defined by ISA-95 [Int10] aligning DTs with recognized industry standards, enhancing their applicability and effectiveness in real-world industrial settings.

To accurately define the phases of the physical machinery, it is essential to specify the transition conditions between these phases by leveraging the raw data

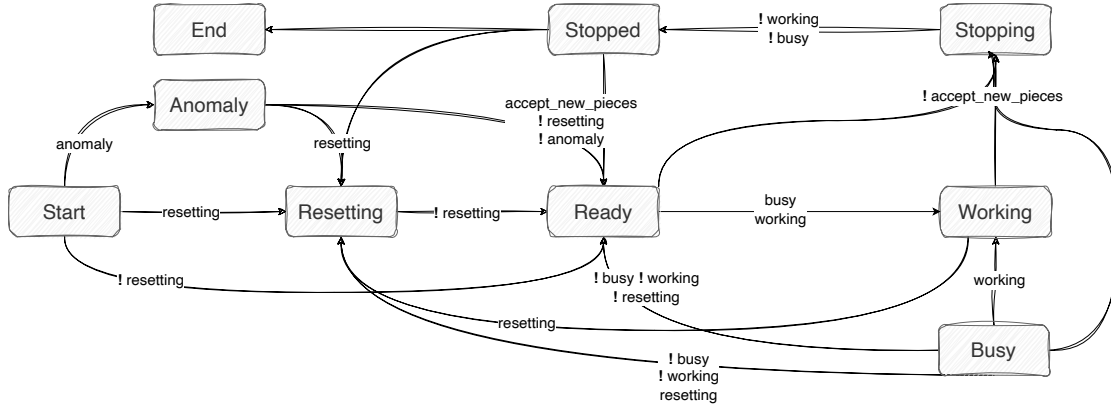


Figure 5.16: Lifecycle of a PA in the target industrial scenario, highlighting the different phases that the DT needs to account for when synchronized.

and telemetry received from sensors and actuators via OPC-UA from the PLC. The DT processes this data to compute and maintain the synchronized lifecycle, ensuring a precise and real-time reflection of the physical object’s operational state.

Department-level DTs enable hierarchical coordination not only between machines within the same station but also across different stations. For example, coordination between the *Multiprocess Station* and the previously mentioned *Indexed Line* can be simplified by establishing consistent lifecycle phases across machines. Without this approach, coordination among multiple DTs requires using low-level physical information and telemetry, making the process less intuitive.

As illustrated by Algorithm 1, high-level control logic can be implemented directly leveraging information encapsulated by LP_{DT} mapping the phase and the transitions of the associated physical counterparts LP_{PA} . Specifically, the algorithm outlines the steps the department-level DT can take to halt production on the multiprocess station. By leveraging this additional layer of abstraction, the department-level DT no longer relies on raw sensor data for control operations, enabling a more declarative and effective coordination process, and distributing responsibilities across different DTs. The composite DT leverages the PA lifecycle phases of individual machines to define the overall lifecycle of the entire multiprocess station. This provides a comprehensive overview of each station’s operational behavior, enabling more effective and intuitive system management.

Algorithm 1 Multiple DTs Stopping Procedure

```

invoke(OvenDT, STOP)
while OvenDT.phase is not STOPPED do
    wait(OvenDT.phase is not WORKING)
end while
invoke(GripperDT, STOP)
while GripperDT.phase is not STOPPED do
    wait(GripperDT.phase is not WORKING)
end while
invoke(TurntableDT, STOP)
while TurntableDT.phase is not STOPPED do
    wait(TurntableDT.phase is not WORKING)
end while
invoke(OutputConveyorDT, STOP)

```

5.6.5 Performance Evaluation

To validate the WLDT framework, the DTs for the Multiprocess Station with Oven were containerized using Docker, and deployed on an Edge node with an Intel i7 2.4GHz processor and 32 GB of RAM. Three key metrics for each DT for a total of 10 minutes (600 seconds):

- **CPU Usage:** the percentage of CPU resources utilized by each DT instance over time;
- **Memory Consumption:** the amount of memory (in MiB) used by each DT instance during its operation;
- **Network Traffic:** the cumulative inbound and outbound network traffic (in Kbit/sec) generated by each DT instance.
- **OEE Computation:** the Overall Equipment Effectiveness calculated by each DT to assess machine and station performance.

Figure 5.17 shows the results. Analyzing CPU usage, there were no significant spikes during the production cycle. Even during the most computationally demanding moments, CPU utilization remained around 30%, indicating a stable and efficient execution. The memory usage chart shows a similarly stable pattern,

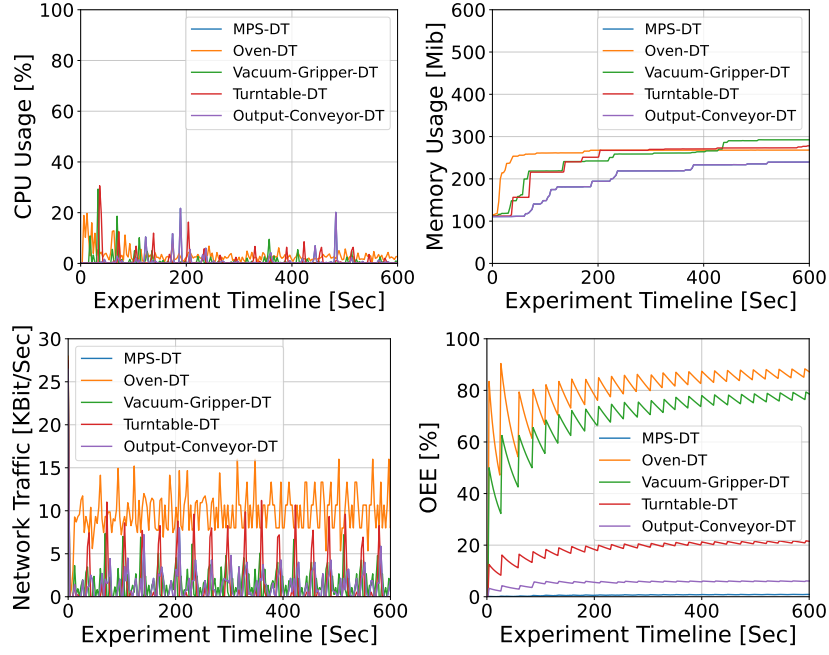


Figure 5.17: Performance indicators of the Multiprocess Station DTs: CPU usage percentage [%], Memory usage [Mib], Network usage [KBit/sec] and OEE [%]

with DT instances settling around an average usage of 300 MiB, demonstrating consistent memory behavior over time. As for network traffic, values remained low—typically around a few tens of Kbit/sec. Noticeable traffic peaks correspond to active processing phases, where machines and their respective DTs exchange larger volumes of data. Finally, the OEE graph highlights the performance of industrial machines computed by each DT. The Oven, which is the first machine in the station and requires the longest processing time, shows the highest OEE, as it remains under constant workload. Conversely, the output conveyor exhibits a lower OEE, as it receives fewer parts to process due to the upstream bottleneck created by the Oven.

This experimental setup aimed to demonstrate the effectiveness of WLDT-based DTs in an industrial setting. The collected metrics confirm that DT instances can operate efficiently on standard commercial Edge nodes, handling CPU, memory, and network constraints while also supporting domain-specific computations such as OEE. These results validate the potential of the WLDT framework

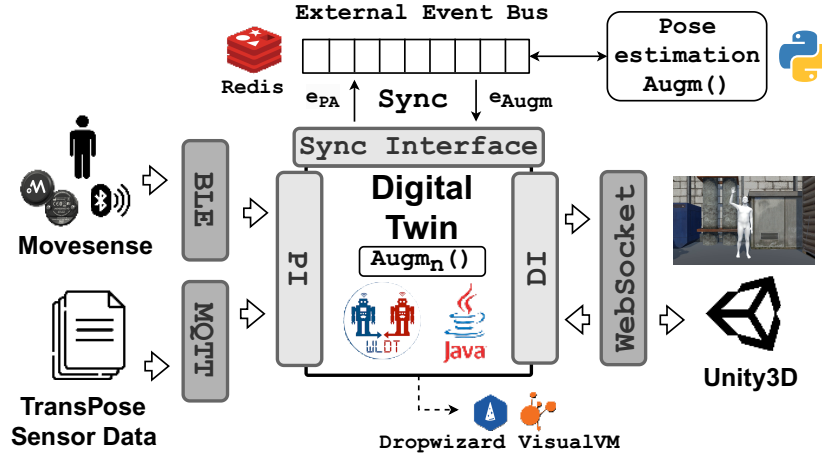


Figure 5.18: Experimental setup for evaluating the design of augmentation functions for a pose-estimation task in a DT of a human operator.

for practical DT development and deployment in real-world industrial scenarios.

5.6.6 Implementing Augmented Digital Twins

A different experimental evaluation was conducted to validate the benefits of the modular design of augmentation proposed in Section 5.4. The reference scenario involves the real-time monitoring of an industrial operator’s movements within a manufacturing environment through a DT whose state is augmented with pose estimation capabilities computed from inertial sensor data.

Figure 5.18 illustrates the experimental setup designed to evaluate the proposed augmentation function patterns within a DT. The DT uses both physical devices and emulated traces, and embeds a machine learning model to estimate pose of the operator which is then visualized in a Unity3D¹³ environment connected to the DT. This setup showcases how internal and external augmentation functions can be implemented in practice and what are the implications on the DT performance.

The DT acquires data from a Movesense Active sensor¹⁴: a wearable device featuring a 9-axis Inertial Measurement Unit (IMU) with accelerometer, gyroscope, and magnetometer capabilities, along with ECG and heart rate monitoring. This

¹³Unity: <https://unity.com/>

¹⁴Movesense: <https://www.movesense.com/movesense-active/>

#Augm.	13Hz	52Hz	104Hz
1 function	2.55ms	6.58ms	10.76ms
3 functions	2.69ms	6.54ms	8.42ms
5 functions	2.79ms	6.79ms	11.79ms

Table 5.2: Average AF execution time with fixed activation period (5s) varying sensor sampling frequency and number of simultaneous functions.

sensor, connected via Bluetooth Low Energy (BLE), allows for continuous tracking of physiological and motion-related data essential to represent the operator's state. Since the sensor can be configured to measure and send data with different sampling rates (13 Hz, 52 Hz, and 104 Hz) it was possible to test the DT responsiveness under different conditions. For the reported experiments, the DT has been deployed on a machine with an Intel i5-13600KF processor (5.1 GHz, 14 cores, 20 threads) and 32 GB of RAM and an RTX 4070ti GPU (12 GB GDDR6X).

Performance of Internal Augmentation Functions

To evaluate the performance of the augmentation functions, execution times, CPU usage, and memory consumption on the workstation were monitored. Execution times were measured by testing various configurations. The augmentation functions are implemented *internally* in Java as periodic functions activated at a fixed period. Each function computes the average of accelerometer values received from the Movesense sensor since the last activation.

In the first experiment, the activation interval of the functions was fixed (5 seconds). Measurements of function execution time were taken using different sensor sampling frequencies (13, 52, and 104Hz) while varying the number of concurrently running functions (1, 3, and 5 functions). Table 5.2 presents the results from this experiment. The experimental evaluation of augmentation functions confirmed that even when multiple augmentation functions coexist within the same DT instance the DT maintains good performance.

In the second experiment, the sampling frequency was fixed at 13 Hz. In this setup, both the number of functions and their activation intervals were varied (1, 5, 10, and 30 seconds). The results from this experiment are detailed in Table 5.3. The DT performance is still robust, with minimal increases in execution times

5.6. EVALUATION OF THE PROPOSED APPROACH

#Augm	1s	5s	10s	30s
<i>1 function</i>	1.70ms	2.05ms	4.17ms	12.64ms
<i>3 functions</i>	1.57ms	2.15ms	3.54ms	13.92ms
<i>5 functions</i>	2.37ms	2.20ms	5.93ms	11.1ms

Table 5.3: Avg AF execution with fixed frequency at 13Hz varying number of functions and activation period.

#Devices	13Hz	52Hz	104Hz
<i>1 Device</i>	3.60ms	16.29ms	32.59ms
<i>3 Devices</i>	7.70ms	46.61ms	81.76ms
<i>4 Devices</i>	18.56ms	62.06ms	111.71ms
<i>5 Devices</i>	23.12ms	75.68ms	104.42ms

Table 5.4: Average AF execution time with multiple devices connected, varying sensor sampling frequency.

even as the number of functions and activation intervals varied. This confirmed the system ability to handle multiple augmentation functions efficiently.

A third experiment was conducted to further examine execution times with up to five Movesense sensors connected to the DT simultaneously. The function activation frequency was fixed at 5 seconds, while the number of connected devices and sampling frequencies were adjusted. Table 5.4 summarizes the outcomes.

To evaluate the resource consumption of the augmentation functions running on the DT, a fourth experiment was performed using a single augmentation function with a 5-second activation interval. In this setup, both CPU usage (as a percentage) and memory consumption (in MB) were measured across the three different sampling frequencies available with Movesense sensor (13, 52 and 104Hz). The results for CPU and memory usage are summarized in Table 5.5. The results showed that the DT maintained efficient resource utilization across different sampling frequencies, further validating the architecture ability to support high-frequency data streams and complex computations even with multiple data sources and physical devices connected and managed by the DT. These experiments collectively demonstrated that the feasibility of the proposed event-driven architecture for DT augmentation functions.

The architecture successfully supports augmentation of DT functionalities, pro-

Data Freq.	CPU%	Mem [MB]
13Hz	0.72%	158MB
52Hz	0.98%	157MB
104Hz	1.16%	156MB

Table 5.5: Average AF CPU and memory usage with one function at a 5-second activation period and varying sensor sampling frequency.

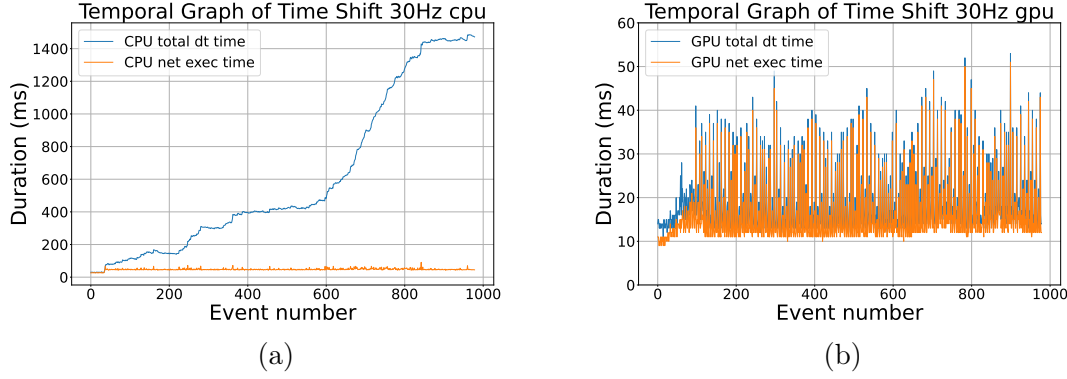


Figure 5.19: Pose estimation latency computed within the DT compared to standalone execution, using the CPU (a) and using the GPU (b).

viding timely insights and maintaining good performance, even under varied and demanding conditions.

Pose Estimation with External Augmentation

For the implementation of the pose estimation DT feature, the augmentation function is implemented externally using a Recurrent Neural Network from the TransPose project [YZX21] using accelerometer data from six inertial sensors positioned on the operator's body to compute its position. As shown in Figure 5.18 the original dataset of the project is used to emulate traces that are sent with a 30Hz frequency to the DT through its physical interface using MQTT. The augmentation function operates externally, running in parallel with the DT. Data exchange occurs via a Redis event bus, and once the position is determined, the DT shadowing function publishes the result through the digital interface using a WebSocket to apply the inferred pose on a 3D model of a person in a Unity3D environment for visualization purposes.

The latency between data transmission and the reception of the estimated pose by the Unity3D application is measured to assess the performance of the external augmentation function. First the neural network is executed using only the CPU, and then with GPU acceleration, to evaluate the impact of consciously allocating resources to augmentation functions.

Both the neural network model and the DT are executed on the same workstation. Figure 5.19a illustrates the total latency introduced by the DT and the model compared with the latency associated to the same function executed outside the DT. With CPU-only execution, the overall delay increases as the model cannot keep up, causing event queue buildup and increased latency. As shown in Figure 5.19b, with the allocation of GPU resources the delay between transmission and reception remains stable over time, with most latency now attributed to model execution rather than DT processes.

This experiment demonstrates that the proposed model for the augmentation functions effectively supports developers in distributing the computational load on dedicated resources, thus improving the overall DT performances.

In conclusion, the experimental evaluation demonstrates that the proposed event-driven architecture for DT augmentation functions is both feasible and efficient. The internal, external, and hybrid augmentation strategies offer flexibility, enabling real-time data processing and complex computations. The architecture supports high-frequency data streams and provides timely insights, making it suitable for different applications.

5.7 Final Remarks

This chapter presents contributions to the engineering of DTs focusing on key challenges in the modeling and implementation of DTs as active components in an IoT system, capable of integrating heterogeneous data sources, managing their synchronization with the PA lifecycle and provide augmented functionalities.

Despite the target scenario used for validation being an industrial manufacturing setting, emulated through a lab-scale production line, the proposed architectural framework and patterns are general and can be applied to other domains. Indeed, the WLDT framework, implementing the functionalities described in this

chapter, has been used in different contexts such as smart-cities and healthcare.

The chapter addresses the research question:

RQ1 How to engineer DTs which integrate heterogeneous data sources and provide an up-to-date uniform representation of the mirrored assets? The modular architecture grounded on the concept of PhAs and DiAs effectively decouples the complexities of physical and digital environments from the core responsibilities of the DT. This enables integration of heterogeneous data sources by means of reusable components, simplifying the development of DTs and promoting interoperability. Furthermore, the lifecycle management managed at the DT level ensures that domain-specific synchronization requirements can be validated and enforced, providing applications with accurate and timely information about the state of the PA. Finally, the modular approach to DT augmentation allows for the flexible integration of additional functionalities, either internally or externally, without impacting the core shadowing responsibilities of the DT.

Chapter 6

Digital Twin Ecosystems

*This chapter contains contributions from the
following works I authored:
[GBCR24, GBCR26]*

As introduced in Section 2.7, the concept of Digital Twin Ecosystem has been proposed to address the need of representing application scenarios that involve multiple PAs that are interconnected and interdependent.

While conceptual models lay out a foundation for DTEs, they leave an operational gap towards their practical implementation. Additionally, real-world deployments are, in many cases, not greenfield: the integration of existing DT infrastructures is not only desirable but often essential. Therefore, arguably, DTEs implementations must account for the challenges that arise when integrating DTs of different entities, belonging to possibly different stakeholders, and potentially built using different technologies.

In this chapter, the concept of Digital Twin Ecosystems is further elaborated by providing a characterization in terms of their functionalities and the architectural patterns that emerge from the analysis of existing practice. This characterization will serve as a foundation for the discussion on the research challenges in the context of practical realizing DTEs which are addressed in this thesis in the following chapters.

6.1 Characterizing Digital Twin Ecosystems

DTEs emerge as a natural evolution of the DT concept, driven by the need of breaking out of the silos of individual DTs to better represent the interactions that occur in the physical world among different PAs which can rarely be considered completely in isolation.

This is particularly relevant when taking the perspective of using DTs as system-wide abstractions to design CPS. For the scope of this thesis, the following definition of Digital Twin Ecosystem is adopted:

A Digital Twin Ecosystem is a dynamic set of Digital Twins, each modeling a Physical Asset, whose meaningful relationships within a target context make it valuable to consider their collective evolution to accurately represent the mirrored portion of the physical world.

This definition highlights three key aspects of Digital Twin Ecosystems:

- **Dynamic set of Digital Twins:** DTEs are not static constructs; they can evolve over time as new DTs are added or removed, reflecting changes in the physical environment or system requirements.
- **Meaningful relationships:** The interactions and dependencies among the constituent DTs are crucial. These relationships can be functional, spatial, temporal, or based on data exchange, and they define how the DTs influence each other within the ecosystem.
- **Collective evolution:** The value of a DTE lies in its ability to represent the collective behavior of its constituent DTs. This holistic view enables new forms of interactions that support processes that consider the interdependencies among multiple PAs.

From a functional perspective, there are two roles that DTEs users can assume:

DTE Managers may need to oversee the creation and evolution of the ecosystem. From such management perspective, being a *dynamic* set of DTs, members could join or leave the ecosystem at any time for different reasons. These include

the PAs lifecycle (e.g., decommissioning) or application-specific inclusion criteria adopted when modeling the ecosystem (e.g., entering or exiting a geographical area). To support this dynamism, DTEs should provide operations to *add* and *remove* DTs. These can be public or restricted to administrators, and invocations can be either manual or automatically triggered by monitoring processes (or the DTs themselves) when detecting relevant changes in the physical environment. A consequence of this dynamicity is the need to easily *discover* which DTs are part of the ecosystem. DTEs may hold an index of all (currently) registered DTs, each *uniquely identified* and described with metadata to support fine-grained discovery. A key piece of information to store is the mirrored PA identifier, allowing tracking of both the digital and physical components of the ecosystem. Since multiple DTs may represent the same PA for different purposes, this information becomes especially valuable. Moreover, to keep track of a DT evolution, it should be possible to *update* metadata over time.

DTE Consumers represent either human users, applications, or intelligent agents which may be interested in observing the state of the registered DTs and exploiting their services. An ecosystem should then offer functionalities to access individual DTs and ecosystem-wide services. This includes the ability to retrieve the current *representation* of the state of all DTs and their relationship at a given time. As the representation of the whole ecosystem could be large and difficult to navigate, consumers should also be able to *query* the ecosystem, enabling them to selectively access and aggregate relevant data across multiple DTs. This supports the derivation of insights that would not be easily discovered by accessing each DT individually. Given the dynamic nature of DTs continuously updating their representation of the corresponding PAs, DTEs should support *observation* patterns—possibly selectively through continuous queries [BW01]—to track changes of DTs and of the whole ecosystem over time. Finally, *navigation* by following DT relationships is a fundamental feature of DTEs, complementing the other interaction patterns. This allows users to explore and discover information progressively. Moreover, relationships may cross ecosystem boundaries, offering paths to discover related DTEs.

Table 6.1 summarizes the support for the main functionalities in some state-of-

	Azure Digital Twins	Amazon IoT TwinMaker	Eclipse Ditto	Twinbase [AST21]
<i>DT Add/Remove</i>	HTTP API	HTTP API	HTTP API	GitHub API
<i>DT Relationships</i>	Yes	No ^a	No	Yes
<i>DT Description</i>	DTD + JSON	No ^a	WoT TD + Ditto Thing	JSON/YAML
<i>Self-described API</i>	No	No	WoT TD	No
<i>Query Support</i>	ADT Query Language ^b	PartiQL ^c	No	No
<i>Observe DTs</i>	Req. Integration	Req. Integration	WebSocket	No

^a Amazon IoT TwinMaker uses *entities* as the main concept, defined as parts of a DT so there is no clear description of a DT

^b <https://learn.microsoft.com/en-us/azure/digital-twins/concepts-query-language>

^c <https://partiql.org/>

Table 6.1: Comparison of DTE feature support in state-of-the-art DT platforms based on the documentation as of May 2025.

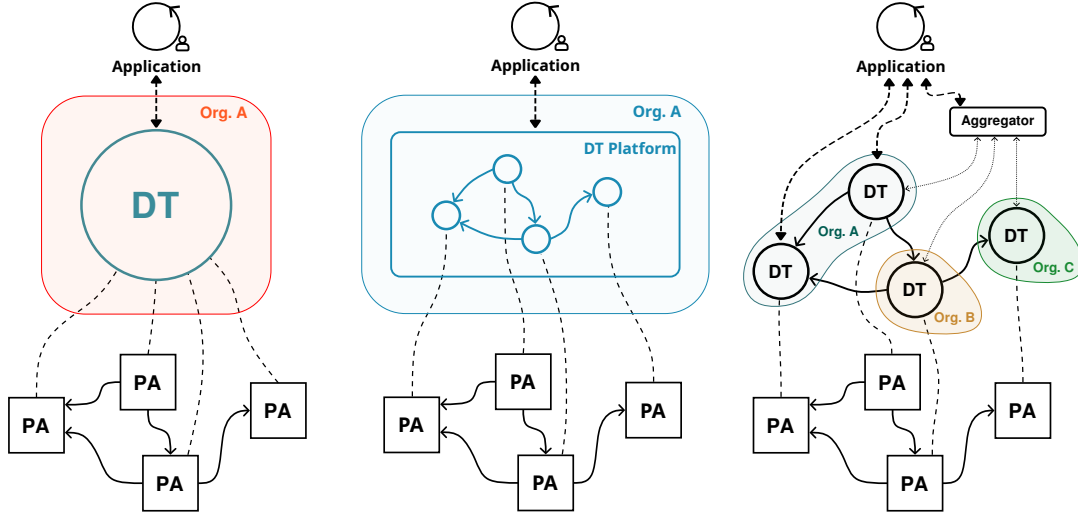
the-art DT platforms. Namely, the table reports how different platforms support the management of multiple DTs, if they provide off-the-shelf support to track relationships between DTs and whether they allow querying information across DTs. The main limitations of the presented approaches are that they require compliance of the DT to platform-specific constraints in terms of:

- protocols and data-formats for the DT to communicate with the platform;
- APIs used by consumers to interact with either the DTs or the ecosystem as a whole (when supported).

This makes DT implementations often tailored to the specific target platform, and limits interaction across multiple platforms since they often propose custom solutions rather than adopting open standards.

6.2 Architectural Patterns

The characteristics and functional requirements identified for DTEs lead to different trade-offs concerning *(i)* the ability to evolve to accommodate changes in



(a) **Monolithic Ecosystem** (b) **Homogeneous DTE** (c) **Heterogeneous DTE**

Figure 6.1: Three architectural approaches for DTEs: in 6.1a a single DT models the whole scenario; in 6.1b a single platform is used to build and deploy all the DTs in the ecosystem; in 6.1c maximum flexibility is allowed, possibly reusing existing DTs under a common interface. Squares represent PAs, circles represent DTs and colored shapes define boundaries between different organizations that may adopt different technological stacks.

the domain of interest, *(ii)* loose coupling of the member DTs, and *(iii)* implementation of the ecosystem functionalities. These are discussed in this section, presenting three architectural approaches emerging from the state of the art of DTE research and technologies (Figure 6.1).

Monolithic Ecosystem

The most straightforward approach is to model the entire targeted context as a single, monolithic DT. Due to the fuzzy definition of a PA, even complex entities (e.g., an entire city) can serve as valid physical counterparts of a single DT, with some models explicitly capturing multiple internal components of such entities. Examples of monolithic DTs include modeling a single DT for entire roads [KSI23], hospital buildings [PZY⁺20], or smart grids [DSH21].

The main benefit of this approach is its simplicity in deriving the overall system architecture: all physical-world data can flow into a single system built with a

consistent technological stack (Figure 6.1a). A monolithic DT also offers strong modeling capabilities and facilitates services such as running simulations on a single model of the entire ecosystem.

The drawbacks of the approach are on the management side. Representing dynamic scenarios involving multiple heterogeneous assets with high fidelity can be challenging, and even minor updates—such as adding new entities or modifying the behavior of existing ones—may require modifying the entire DT. This approach is not suited for rapidly evolving ecosystems or contexts involving multiple stakeholders. Relying on a single DT limits technological diversity, forcing the creation of a new, unified model.

Homogeneous DTE

A DTE can be considered a *Homogeneous* DTEs when the approach relies on a general-purpose platform that supports the definition of multiple DTs (Figure 6.1b). The DTs are hence homogeneous in their modeling and implementing technology as they are created on the same platform. The native support for DTEs enhances adaptability to the evolving physical world as it is simple to add and remove DTs or update models. Furthermore, the modeling and technological homogeneity make implementing ecosystem services relatively easy, similarly to the *monolithic* approach. Differently, though, it is possible to highlight the individuality of the involved entities and their corresponding DTs, which may independently process the relative data flows, achieving better separation of concerns.

The dependency on a single platform, however, may introduce vendor lock-in, modeling and organizational constraints. An example of this approach is the Azure Digital Twins platform, which introduced the concept of a *Twin Graph*¹ to explicitly represent relationships among DTs modeled using a uniform Digital Twin Definition Language (DTDL).

Heterogeneous DTE

The third approach is based on introducing a common interoperability layer on top of technologically heterogeneous DTs. This approach enables the reuse of existing

¹<https://learn.microsoft.com/en-us/azure/digital-twins/concepts-twins-graph>

DTs while ensuring uniform access and navigation for consumers (Figure 6.1c). DTs are implemented as standalone software components, each with specific models and technologies, and each exposing interfaces that can be used by applications directly or through intermediary aggregators.

This approach is well-suited for dynamic, open DTEs where DTs are developed by various stakeholders since the individual components are loosely coupled. Furthermore, DTs are not limited to being part of only one ecosystem, granting an additional degree of flexibility.

To maintain interoperability, though, each DT must be mapped to a shared metamodel to represent it within the ecosystem and expose a shared interface to enable seamless interaction. This can lead to a potential loss of information and expressivity, as with any model conversion. However, direct access to the original DT remains possible when needing to use specialized services not easily mapped in the shared model.

Compared to the previous approaches, where ecosystem functionalities can be directly supported by the implementing technology/platform and benefit from modeling uniformity, implementing ecosystem services in heterogeneous DTEs is inherently more complex. This can be achieved through either fully distributed techniques—such as polystores [DES⁺15] or federated and link-traversal queries [SHH⁺11, QL08, BKZ⁺21]—or by introducing a middleware (as shown in Figure 6.1c) aggregating DTs and implementing the required functionalities. This doubles as a way to set a clear boundary for operations, allowing a clear definition of which DTs belong to an ecosystem.

6.3 Heterogeneity and Trade-offs

The trade-offs in the different approaches to engineer DTEs arise from the necessity of managing several DTs as a cohesive system. In general, the set of DTs belonging to the same ecosystem may be *heterogeneous* for various reasons. To better characterize this heterogeneity, it is possible to extract and analyze a set of orthogonal dimensions along which DTs may differ.

Table 6.2 briefly reports the identified dimensions through an analysis of the relevant literature on DTEs. Findings demonstrate that DTEs are usually envisioned

Heterogeneity Dimension	Description
Organization	DTEs may digitalize large-scale contexts involving PAs that belong to different stakeholders and organizations that may need to collaborate [THXL24]
Domain	DTEs may include assets from different application domains and hence require the integration of different domain vocabularies such as in the case of a smart city ecosystem that may span across the energy, mobility, ecology, and financial domains [DZS21]
Purpose	The DT paradigm is employed for a variety of purposes [BBD ⁺ 18] leading to very different DTs having to coexist in an ecosystem. Examples range from monitoring to simulation, what-if analysis, and planning.
Modeling	DT models may vary based on the nature of the corresponding PAs with one DT possibly having more than one model within itself [QTH ⁺ 21]. For example, while a manufactured product may be effectively represented with a three-dimensional model, using differential equations may be more suited for a physics system [BBD ⁺ 18]
Technology	The spread of the DT paradigm led to a variety of supporting technologies [QTH ⁺ 21], often creating closed vertical technological silos [DBB19]. In complex real-world contexts where legacy is the norm, it is natural to assume that different technological stacks may coexist in the same DTE

Table 6.2: Dimensions of DT heterogeneity that characterize DT ecosystems.

in contexts that either involve multiple stakeholders (*organization dimension*) and a diverse set of assets (*domain dimension*) that may require different techniques to be described effectively (*modeling dimension*). Furthermore, DTs are employed for a variety of use cases, ranging from monitoring to simulation and forecasting (*purpose dimension*). All these factors naturally result in technological fragmentation (*technology dimension*), which is particularly evident in the current DT technology landscape with solutions either tailored to specific application domains [DBB19], or providing a general-purpose platform for handling different DTs.

As an additional factor of complexity, different stakeholders may be interested in different views of the same PA, ending up independently developing multiple DTs to fulfill distinct business goals [MLC20]. Ecosystems can then either serve a unifying purpose, connecting different views of the same PA, or exist in parallel to serve specific perspectives of the same domain.

Heterogeneity influences the design of DTEs. Namely, monolithic and homo-

geneous ecosystems work best when DTs are aligned for what concerns the organization and modeling dimensions, as it might be easier to align the development to common guidelines. Differently, ecosystems involving multiple stakeholders, and spanning across different application domains, might require additional degrees of flexibility that only a heterogeneous approach can grant. Such flexibility comes at a price: implementing ecosystem services is easier—and possibly even supported by off-the-shelf solutions—when dealing with a homogeneous set of entities. Heterogeneous DTEs must instead rely on a carefully designed interoperability layer, adapting different DTs to a common interface.

6.4 Challenges of Digital Twin Ecosystems

As outlined in the previous section, due to its multi-dimensional nature, heterogeneity should be regarded as an inherent source of complexity in DTEs that can not always be avoided. This work focuses on addressing the challenges that arise when dealing with heterogeneous DTEs, as they represent the most general and challenging case for what concerns the design of ecosystem functionalities and the integration of existing DTs infrastructures.

Two main challenges are identified and explored in the following chapters:

- **Interoperability of Heterogeneous Digital Twins:** enabling interaction across DTs built using different technologies, models, and purposes. This includes defining a common metamodel to represent DTs within the ecosystem and implementing mechanisms to translate between different DTs interfaces and the shared interface exposed by the ecosystem. Addressing this challenge will answer **RQ2**, with a proposal for the engineering of heterogeneous DTs as discussed in Chapter 7.
- **Deployment and Runtime Management of Digital Twin Ecosystems:** orchestrating the deployment, configuration, and management of DTs within the ecosystem. This involves handling the dynamic nature of DTEs, including adding and removing DTs in a complex cyber-physical infrastructure. Addressing this challenge will answer **RQ3**, with a proposal for the runtime management of heterogeneous DTs as discussed in Chapter 8.

There are other significant challenges in the context of DTEs which are not directly addressed in this thesis.

These include aspects such as security and privacy of the data exposed by DTs, as well as trust and governance between multiple stakeholders. Although it is true that building DTEs can be effective even when considering a focused application use case with limited stakeholders, open DTEs spanning across multiple organizations can support novel forms of collaboration and value creation [Ken21]. Addressing these challenges requires a broader sociotechnical perspective, that involves pushing towards open data and open standards to foster collaboration and interoperability at a larger scale [AKP24].

Chapter 7

Interoperable Digital Twin Ecosystems

*This chapter contains contributions from the
following works I authored:
[GBCR24, GBCR26, GBCR25]*

Answering **RQ2**, this chapter presents a proposal for the engineering of *Heterogeneous* DTEs (Chapter 6). Motivated by the design of the Web and its success in creating interoperability among digital services—more recently, in IoT systems through the WoT (Chapter 3)—the chapter explores whether and to what extent hypermedia principles and Web standards can be applied to design and implement DTEs. The result of this investigation, refines and extend the ideas originally presented in [RCM⁺22a], combining the WoDT conceptual model with the design rationale of the Web to build a *Hypermedia Web of Digital Twins (HWO DT)*.

This approach offers a practical implementation strategy for DTEs, enabling interoperability at both the DT and ecosystem levels, and facilitating uniform interaction across DTs regardless of the underlying technological stack.

The chapter presents the HWO DT conceptual integration of hypermedia principles and Web standards in DTE, and the prototype implementation of a supporting set of tools that demonstrate the feasibility of the approach.

7.1 A Hypermedia Web of Digital Twins

The main driver of the proposed approach is to support seamless integration of existing DTs, regardless of their underlying technologies and with minimal overhead. Hence, the core idea is to hide the heterogeneity of DTs behind a *uniform interface* built with Web protocols and standards, and supported by an explicit semantic layer that allows DTs to provide a uniform description of both their state and their features and services. Thus, the name *Hypermedia Web of Digital Twins* (*HWoDT*), aims to emphasize the hypermedia-driven nature of the approach, following the HATEOAS principle of the REST architectural style [Fie00].

Such an interoperability layer enables navigation and seamless interaction with heterogeneous DTs. To complement it with additional ecosystem-level functionalities, the idea of a *WoDT Platform* is introduced as both a scope boundary defining ecosystem membership and an aggregation layer enabling consumers to query, observe, and exploit services across distributed DTs.

As illustrated in Figure 7.1, the resulting architecture is composed of DTs: (i) created using heterogeneous technologies, (ii) implementing the uniform interface through *adapters*, (iii) connected by relationships that reflect the physical ones, (iv) and that are aggregated into DTEs by being registered to one or multiple WoDT Platforms.

7.1.1 A Uniform Interface for Heterogeneous DTs

The first step towards integrating heterogeneous DTs is to look into how each individual DT can expose a uniform interface, compliant with the requirements of the HWoDT. Achieving standard semantic representations of DTs is an open challenge for the DT community. In this work, adopting a pragmatic approach, the benefits of uniform semantic representations in building interoperability are demonstrated through a proposal grounded on the HATEOAS principle of the Web architecture: each DT must be able to provide consumers with both *data* and *affordances*—i.e., action possibilities—through hypermedia representations. This leads to distinguish two logically different representations:

- a *Digital Twin Description (DTD)*, inspired by the WoT Thing Description,

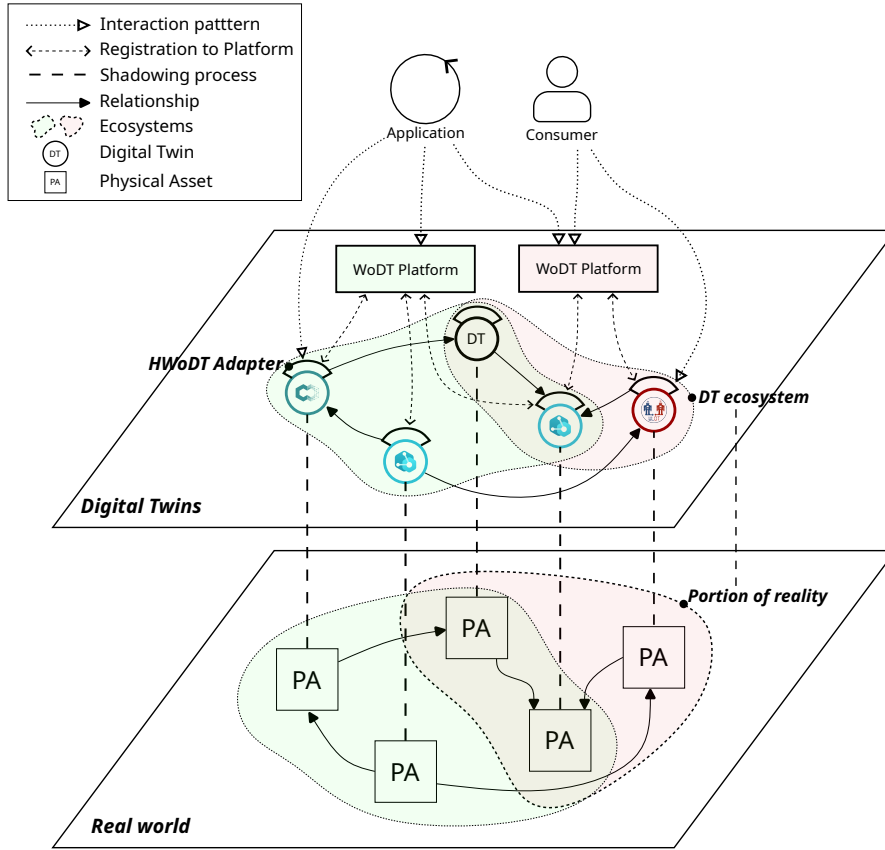


Figure 7.1: The Hypermedia WoDT scheme, in the real world PAs (squares) belonging to different organizations and domains are connected by relationships. In the digital world, cross-domain ecosystems of DTs (circles) mirror different portions of reality, supported by the WoDT Platform. Consumers (top) can interact with the ecosystem and its DTs directly, or through the platform.

to hold metadata and affordances, enabling consumers to understand the DT model, as well as which services the DT exposes and how to access them;

- a *Digital Twin Knowledge Graph (DTKG)* representing the live state of the DT, semantically encoding domain knowledge and supporting state observation, querying, and discovering relationships between DTs.

Each DT is further required to comply with a set of standard *interaction patterns* to allow consumers to uniformly obtain and manipulate such representations for all DTs in the HWOdt.

The Digital Twin Description

The DTD serves the primary purpose of providing management metadata about a DT, registering it within the ecosystem and describing the exposed interactions¹.

A primary concern is *identification* of the DT and the associated PA to unambiguously distinguish them from other elements of the ecosystem. While PAs may use domain-specific identifiers (e.g., a serial number), each DT in a HWoDT is identified with a persistent, globally unique URI. This ensures identification and accessibility of DTs as Web resources throughout their whole lifecycle.

Furthermore, to support navigation from a DT to the ecosystem it is part of, the DTD also links to the URI of the ecosystem—in the prototype implementation this is a URI served by the WoDT platform to identify the DTE.

Other relevant metadata in the DTD concerns the model used to represent the PA at the digital level. The WoDT metamodel (Section 2.7)—closely aligned with the WoT TD—can describe the DT in terms of which properties, relationships, events, and actions are available to DT consumers.

Finally, the DTD presents a description of the API that consumers can use to interact with the DT, using hypermedia controls to describe protocol bindings for the exposed interaction patterns. The DTD, is thus aligned with REST principles as it presents both data and controls to consumers using self-describing messages.

Even if the DTD may evolve with new properties or interactions when a DT gets updated, it remains mostly static as it describes the DT’s identity metadata and software interface, not its real-time state.

The Digital Twin Knowledge Graph

As a DT main responsibility is to provide an up-to-date representation of the PA state, the Digital Twin Knowledge Graph (DTKG) complements the DTD by representing the live state of the DT with a KG.

The DTKG by means of RDF triples, represents:

- current property values;

¹Documentation on the structure of the DTD schema is available on GitHub <https://github.com/Web-of-Digital-Twins/dtd-conceptual-model>

- current relationships with other DTs;
- context-dependent available actions.

Events, generated by the DT are excluded from this representation as they are non-persistent information handled via subscriptions.

By using RDF, the DTKG can represent knowledge about the DT's state with explicit semantics using domain-specific ontologies to support a common interpretation of DT data. Additionally, following the Linked Data principles [BHBL23], DTs linking to other DTs through relationships supports navigating across the ecosystem in a distributed KG.

Digital Twin Interaction Patterns

Other than generating the necessary DTD and DTKG each DT in a HWoDT ecosystem is required to implement a Web API for consumers. This enables direct use by any Web client.

First, the API should expose methods to retrieve the two DT representations presented above. As the main function of a DT is to represent the PA state over time, when dereferencing the URI which identifies the DT with an HTTP GET request the DT must return the current *snapshot* of the DTKG.

Technically, since a DT qualifies as a *non-information resource*² the standard Web practice is returning a 303 (See Other) status code with the *Location* HTTP header set to the Uniform Resource Locator (URL) of the DTKG information resource. The URL of the DTD is linked in an HTTP Link Header with a custom relation type `dtd` in the response of the DTKG GET request. In this way, consumers can follow links between resources and access both DT representations with standard Web interactions as shown in Figure 7.2a.

Since the DTKG evolves over time to reflect the PA state, DTs are required to support a subscription API to observe such changes (e.g., via WebSockets, long-polling, or WebSub [PG18]). Modeling the DTKG as a Web resource also enables basic historicization using protocols like Memento [dSNS13], if the DT acts as a Memento TimeGate.

²For a discussion on information vs. non-information resources see <https://www.w3.org/TR/cooloris/> and <https://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html>

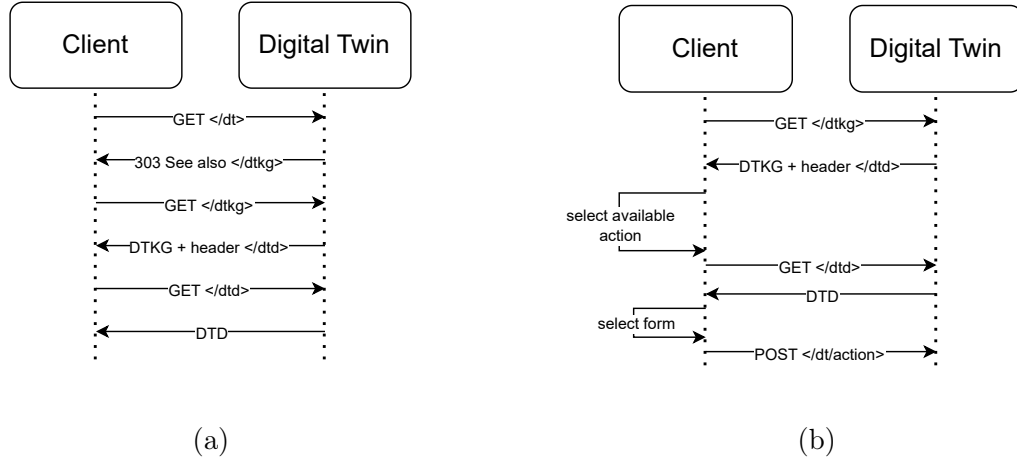


Figure 7.2: Interaction patterns to (a) access the DTD and DTKG and (b) invoke available actions on a DT through the uniform interface and standard Web patterns.

Finally, the API may expose DT actions that can be invoked by external consumers. The DTKG shows which actions are currently valid, while the DTD describes how to request their execution to the DT. Consumers rely on both to interact and invalid actions should fail (e.g., with HTTP 403), ensuring consistency with the DT model. Figure 7.2b illustrates the interaction pattern to invoke an action on a DT, by first checking its availability in the DTKG and then following the affordance described in the DTD to execute it.

7.1.2 Ecosystem Services in the HWoDT

In the proposal for realizing the HWoDT ecosystem functionalities described in Chapter 6 are implemented through a middleware: the *WoDT Platform* aggregates data from registered DTs and exposes an API that consumers can use to manage and interact with the ecosystem as a whole.

Managing the Hypermedia Web of Digital Twins

The HWoDT proposal assumes that DTs are existing, independently running software entities which implement the uniform interface presented in Section 7.1.1.

Hence, the DT creation is handled by developers with their technology of choice, and adding it to an ecosystem is simply a matter of registering it to a WoDT Platform. The platform exposes an API that either managers, an external service or a DT itself, can use to submit a DTD to register a new DT to the ecosystem. The DTD is then stored in a registry, which further allows it to be updated or removed if the DT leaves the ecosystem. Consumers can also use the registry API to discover which DTs are in the ecosystem and to filter them based on metadata in the DTD—e.g., they can discover if multiple DTs in the ecosystem model the same PA filtering on the PA identifier.

These simple operations are sufficient to manage the ecosystem. The DTD is used in the registration process as it provides the necessary metadata to assess membership and compliance requirements as well as the API description that the platform can use to interact with the DT upon registration. Once registered, the platform uses the API description within the DTD to access and observe the DTKG. It can also notify successful registration, enabling the DT to record the platform URI in its list of ecosystems updating its DTD accordingly.

Exploiting the Hypermedia Web of Digital Twins

By observing each individual DTKG, the platform can cache the latest update and aggregate them to create a global DTE KG that can be exploited to implement ecosystem-level services for consumers. The DTD and DTKG of each DT are stored within the KG, allowing queries to easily retrieve all available information about a DT. Each update of a DTKG is processed and merged in the global KG so that current representation of the whole ecosystem is always available to consumers.

The DTE is a non-information resource and has a URI which, when dereferenced, returns the global DTE KG³. Consumers can also *query* the DTE KG through the WoDT platform SPARQL endpoint. This enables the extraction of selected information from the current state of all DTs in the ecosystem with a standard query language.

³As the ecosystem KG can be very large, returning the whole thing can be impractical. Alternatives include returning a partial representation with links to the DTs caches

Finally, consumers may want to observe the evolution of the ecosystem KG. The platform must then support an API to receive subscription requests and send updates to observers. Given the large nature of the DTE KG, consumers might not be interested in receiving *all* updates. Hence, techniques for RDF stream processing [BBC⁺09] might be more effective to support selective observation.

7.2 A Prototype Framework for the HWO DT

To support the prototyping of heterogeneous DTEs based on the HWO DT proposal, a set of tools is implemented aiming at the integration of existing DT technologies. The HWO DT framework is open source and available on GitHub⁴.

The software distribution includes a prototype *WoDT Platform* implementation and *adapters* to implement the HWO DT uniform interface for DTs developed with Azure Digital Twins, Eclipse Ditto, and the WLDT framework (Section 5.5). The technology choice is meant to be representative of the state of the art as they differ in terms of functionalities, namely Azure Digital Twins is a cloud platform, Ditto is an open-source platform, and WLDT allows developing and deploying DTs as standalone software processes. This section presents the overall architecture (Figure 7.3) and its implementation.

7.2.1 A WoT-compatible Digital Twin Description

The DTD is a central component in the design of the uniform interface of the HWO DT. Its design is inspired to the WoT TD, as an API description of the functionalities offered by a DT.

Although the WoT explicitly reference DTs in its architectural patterns, the TD alone does not support all the metadata that are needed in the proposed DTD. Hence, using WoT-compliant mechanisms, TDs are extended to include the relevant metadata in order to qualify as a functional DTD for a prototype implementation of the HWO DT. Namely, the TD is extended using a custom *WoDT vocabulary*⁵ which defines concepts to implement the DTD and the DTKG, as well

⁴<https://github.com/Web-of-Digital-Twins>

⁵<https://github.com/Web-of-Digital-Twins/wodt-vocabulary>

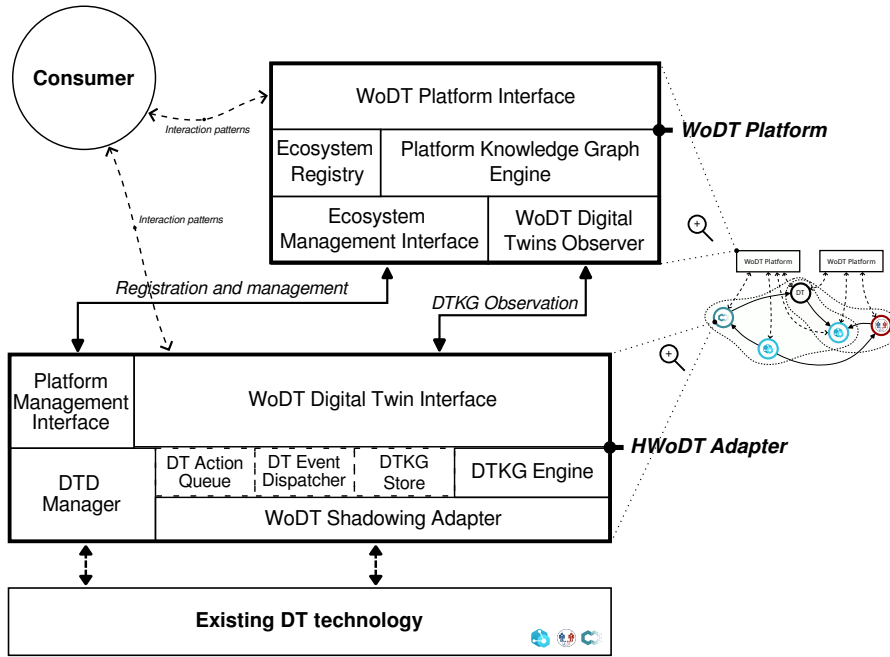


Figure 7.3: Overall abstract architecture of the HWoDT framework, showing the main functional modules and their main interactions. On the top, the WoDT platform manages the ecosystem of DTs by registering them through their DTDs, observing their DTKGs, and exposing the DTE KG to consumers. On the bottom, existing DTs can implement the HWoDT uniform interface through the HWoDT adapter and be integrated with the platform.

as HTTP Link Headers relation types used in the interactions of the HWoDT (see Section 7.1.1). Listing 7.1 shows a generic Thing Model (TM) [MKK23] that can be used to implement a valid DTD. The DTD must include:

- the PA identifier,
- a link to the DTKG,
- an `observeallproperties` affordance to subscribe to updates of the DTKG.

Listing 7.1: The Thing Model that Thing Descriptions must implement to be recognized as a valid Digital Twin Description.

```

1 {
2   "@context": [
3     "https://www.w3.org/2022/wot/td/v1.1",

```

```
4      {
5          "wodt": "https://raw.githubusercontent.com/Web-of-Digital-Twins/
              wodt-vocabulary/refs/heads/main/core.ttl#"
6      }
7  ],
8  "@type": "tm:ThingModel",
9  "title": "{{DT_TITLE}}",
10 "version": {"model": "1.0.0"},
11 "id": "{{DT_URI}}",
12 "wodt:physicalAssetId": "{{PA_ID}}",
13 "tm:optional": [
14     "/properties/availableActions",
15 ],
16 "properties": {
17     "availableActions": {
18         "@type": "wodt:AvailableActions",
19         "title": "Available actions",
20         "readOnly": true
21     }
22 },
23 "links": [{
24     "rel": "wodt:dtkg",
25     "href": "{{DTKG_URL}}"
26 }],
27 "forms": [{
28     "op": "observeallproperties"
29 }]
30 }
```

The alignment with the WoT is strategic for the DT community as reusing the existing standard ensures compatibility and favors the development of application mashups. The DTD can be served by the *DI* of a DT to be discovered and consumed by existing WoT clients (see Section 5.2). Clients that have the capability to interpret the WoDT vocabulary can then exploit the additional metadata to interact with the DT as part of a HWoDT ecosystem.

7.2.2 HWoDT Adapters

DTs in a HWoDT ecosystem must all adhere to the uniform interface, regardless of the technology used to implement them. Adapting existing DT technologies to the HWoDT interface requires aligning data and interaction patterns. To show that this alignment is achievable with configurable reusable components a set of *adapters* is developed to integrate some mainstream technologies within a HWoDT

ecosystem. All adapters support HTTP interactions and WebSocket-based DTKG observation. The abstract architecture followed by all adapters is shown in Figure 7.3, bottom.

Azure Digital Twins Adapter

Azure Digital Twins is Microsoft's domain-independent Platform as a Service (PaaS) DT solution, supporting the management of multiple DTs connected within a *twin graph*.

Creating DTs in Azure Digital Twins requires defining their model using the *Digital Twin Definition Language (DTDL)*, a custom JSON-LD format to specify properties, relationships, commands (actions) and telemetry (events) that describe a *type* of DT. DTDL models can then be used to create DT instances. Although the model closely aligns with the WoDT model, the support of events and actions is partial⁶ and relationships are limited to linking DTs within the same Azure Digital Twins instance, which hence defines a closed homogeneous ecosystem.

The Azure Digital Twins adapter is implemented as a middleware, connecting to an Azure Digital Twins instance and mapping DTs to the HWoDT uniform interface. The adapter can be configured to select which DTs needs to be mirrored and specify the mapping from DTD properties and relationships to produce the DTKG. Figure 7.4 shows the adapter architecture and the necessary components to connect to an Azure Digital Twins instance. The following Azure service pipeline must be set up to link the Azure Digital Twins instance to the adapter:

- *Azure Event Grid* to capture and forward Azure Digital Twins events;
- *Azure Function* () to fetch the current DT state and send it to the adapter via SignalR on every new event;
- *Azure SignalR* to deliver DTs state updates to the adapter.

The adapter can be deployed either within the Azure cloud or is designed to automatically retrieve the DTDL models, stored on Azure Digital Twins, and

⁶As of June 2025, commands can be defined but not invoked automatically, while telemetries are not used within Azure Digital Twins <https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models>

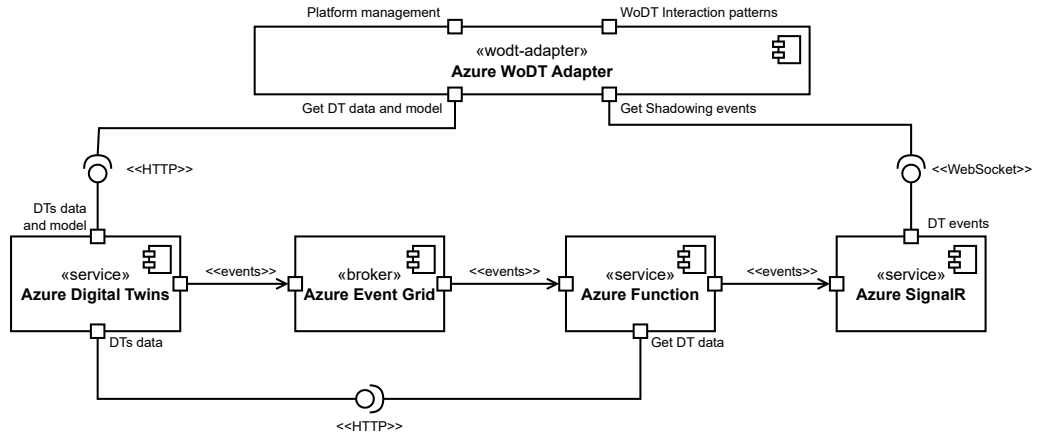


Figure 7.4: Main components of the Azure Digital Twins adapter, implemented as a standalone middleware that maps specific instances of DTs hosted on an Azure Digital Twins instance.

convert them in valid DTDs for each DT instance. The adapter waits for SignalR events to receive DT state updates generated by the Azure Function and convert them into DTKGs updates.

Eclipse Ditto Adapter

Eclipse Ditto, is an open-source DT platform from the Eclipse Foundation. It abstracts each PA as a *Ditto Thing* and offers a Web-based layer for interacting with IoT devices through their DTs. A Ditto instance can manage multiple DTs, using a metamodel with:

- *Attributes* for static metadata,
- *Features* which can group data (properties) and functionalities (messages).

Although lacking native support for relationships, actions, and events, these can be modeled using attributes (for links), consumer-to-DT messages (for actions), and DT-to-consumer messages (for events). Models can be defined via a custom JSON format or a (set of) WoT TM.

Eclipse Ditto is implemented with a microservice architecture. Despite Ditto being open-source and allowing the development of custom extensions, the proto-

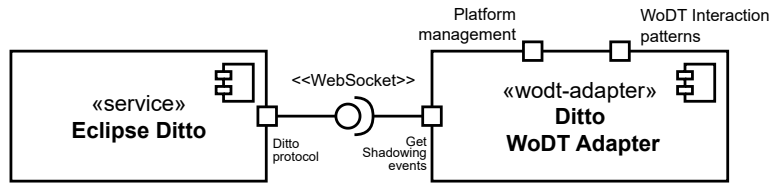


Figure 7.5: Main components of the Eclipse Ditto adapter, implemented as a standalone middleware that maps a specific instance of a DT hosted on the Ditto platform.

type adapter is implemented as a custom external middleware that can be deployed to map one Ditto Thing to the HWoDT uniform interface. The middleware leverages Eclipse Ditto’s native WebSocket interface to retrieve data and performs two key transformations: converting the WoT TD exposed by Ditto into a DTD, and serializing DT data into a DTKG. This process relies on a configurable mapping between Ditto features and their corresponding RDF representations.

White Label Digital Twins Adapter

The *White Label Digital Twins (WLDT) framework* (Section 5.5) enables the development of DTs as standalone software components deployable across cloud or edge environments. Specifically, the adapter is implemented as a DiA, used in the WLDT architecture to represent digital interfaces through which the DT can expose data for consumers. The framework internal metamodel is already aligned with the HWoDT, so no additional effort is required to map concepts. The adapter is provided as a reusable Java library, which WLDT developers can easily import into their projects and then configure the mapping towards the HWoDT uniform interface leveraging the implementation of the necessary API to manage the DTD and DTKG.

Implementing Custom Adapters

The description of the adapters implemented in the prototype HWoDT framework should be useful for any DT developer wishing to implement a custom adapter for their own DT or DT technology. The idea of the HWoDT is grounded on the fact that implementing such custom adapters is relatively straightforward and that

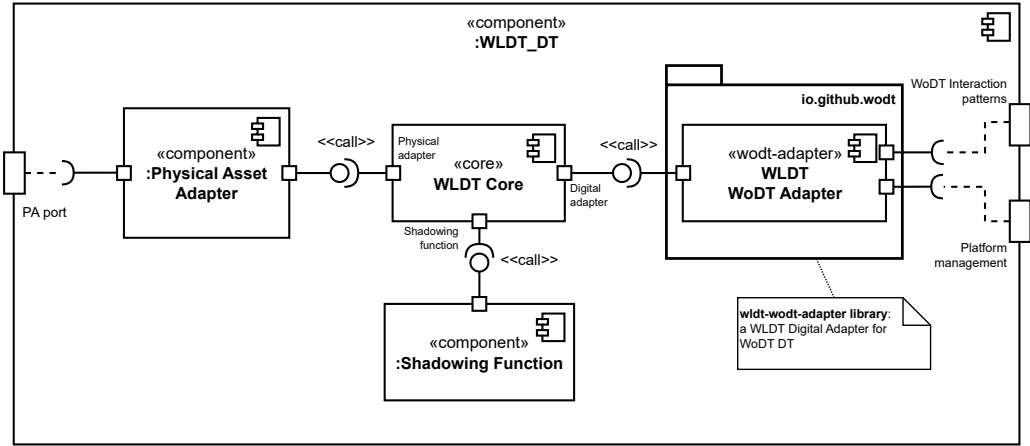


Figure 7.6: Main components of the WLDT adapter, implemented as a Digital Adapter and illustrating its integration with the other components of the WLDT framework.

the little effort spent in developing (or even less in configuring) them can bring benefits in the integration with the HWoDT ecosystem.

To summarize, the steps to develop a custom adapter are:

1. understand how to map the concept of the original DT model into the WoDT metamodel;
2. understand how to represent the state of the DT with an evolving semantic representation for the DTKG;
3. implement a module that can reactively produce the DTKG whenever the DT state is updated;
4. implement a module that can serve the DTD over HTTP, alongside the mandatory API for DTKG observation.

7.2.3 The WoDT Platform

A prototype of a WoDT Platform is implemented in Kotlin. The platform implements the modules of the abstract architecture shown in Figure 7.3, top as shown in Figure 7.7. Namely, the platform offers multiple interfaces for consumers: an

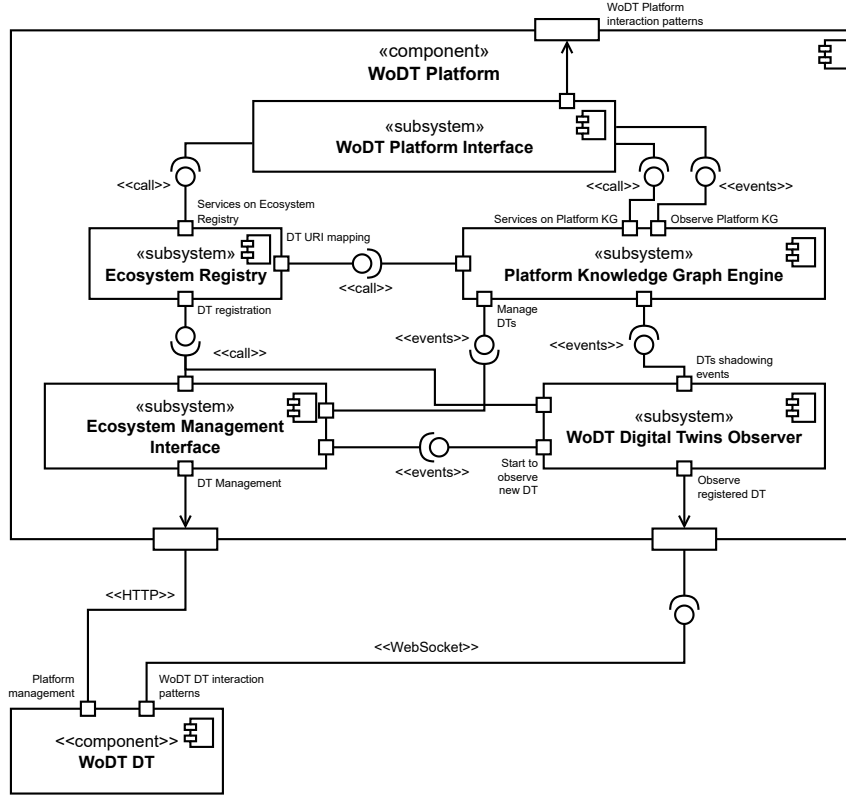


Figure 7.7: WoDT Platform modules and interfaces, represented using components and connectors.

HTTP API for DTE management (Ecosystem Management Interface) and one for KG access with a SPARQL endpoint for read-only queries, as well as a WebSocket endpoint to observe ecosystem updates (WoDT Platform Interface).

The platform can process the WoT-based DTD described above to register DTs. When a registration requested is submitted, the platform:

Step 1 locates the DTKG observation form (**observeallproperties**) and starts observing the DT for updates;

Step 2 notifies the DT of successful registration to let it update its DTD with the platform URI;

Step 3 maps the DT URI to a local cache URL;

Step 4 observes DTKG updates and merges them into the DTE KG stored in memory with Apache Jena.

The current prototype supports the observation of DTKG through WebSockets for **Step 1** and implements **Step 2** sending a request to a hardcoded `/platform` HTTP endpoint on the DT.

Consumers can interact with the DTEs through the WoDT platform either through (possibly repeated) SPARQL queries to retrieve the information they need or by observing the whole KG through WebSockets. In the prototype, all observers receive the whole KG whenever there is an update so that consumers don't need to have specific update handling logic.

7.3 Comparison with Related Works

This section compares the HWoDT proposal with related approaches that target the integration of DTs in ecosystems. The aim of this comparison is to highlight the similarities and differences with existing solutions, discussing advantages and limitations of existing approaches with respect to the HWoDT.

The HWoDT proposal is not meant to compete with existing standards or framework, but rather to either complement them or provide an alternative approach to DT interoperability. The main principle of the HWoDT is to promote interoperability through the adoption of REST principles and Semantic Web technologies, given their proven effectiveness in enabling integration across heterogeneous systems on the Web.

7.3.1 HWoDT and Digital Twin Standards

The HWoDT tackles the relevant issue of enabling DT interoperability targeting the creation of *heterogeneous DTEs*. This is an open challenge in the DT research community (see Section 2.6), and several interoperability frameworks are being proposed to streamline the integration of DTs [Bar24]. Among the most relevant standards are OPC-UA, AAS, AML, and the W3C WoT. Table 7.1 summarizes a comparison of these standards across selected aspects.

7.3. COMPARISON WITH RELATED WORKS

	AAS	OPC-UA	AML	WoT	Semantic Web
Scope	Asset digital representation	Device data / M2M	Tool interoperability	Web-scale access	Cross-domain knowledge
Data model	Submodels + semantic IDs	Hierarchical node graph (address space)	Hierarchies (CAEX, CAD, PLCopen)	Thing Description (JSON-LD)	Triple Graphs (RDF / OWL)
Semantics	Structured submodels, semantic references	(Sub)Types and Relationships	Role-based, structural	JSON-LD semantics, ontology-aligned	General-purpose ontologies
Querying and Data access	REST API	Browse address space	File access	SPARQL (if RDF-mapped)	SPARQL / Web browsing

Table 7.1: Comparison of interoperability frameworks across selected aspects.

The comparison highlights how existing interoperability frameworks address complementary layers of DT integration. AAS focuses on structured digital representations of assets through modular submodels, OPC-UA provides a secure communication infrastructure centered on hierarchical address spaces for device-level data exchange, and AML supports interoperability among engineering tools via structured XML-based hierarchies. The W3C WoT promotes Web-scale integration by describing devices through TDs, while Semantic Web technologies provide a general-purpose, ontology-driven approach to cross-domain knowledge representation based on graph models and query languages such as SPARQL. Together, these standards span different scopes, data models, semantic mechanisms, and data access paradigms, reflecting the multi-layered nature of DT interoperability.

Given their different nature, these standards are better viewed as complementary rather than competing. Integration efforts already aim to leverage their respective strengths: OPC-UA has been integrated with Web technologies [CSS19], operations in AAS [Pla19] and affordances in WoT⁷ can have OPC-UA protocol

⁷<https://profiles.opcfoundation.org/workinggroup/97>

bindings, and AML specifications can be linked to AAS for a coherent view of assets across engineering and operational layers [DRH19, WZM18]. In this landscape, Semantic Web technologies – as adopted in the HWoDT – can be layered on top of existing standards to complement their capabilities.

The Semantic Web excels at heterogeneous knowledge integration under a uniform interface (Section 3.3), enabling expressive queries and ontological reasoning across domains. Alignments between Semantic Web technologies and industrial standards have therefore been investigated, highlighting similarities between, e.g., OPC-UA [MWD19, PPRK19] and AAS [BCB21, Pla19] data models, in order to benefit from graph-based semantics and reasoning capabilities.

Following this trend, the HWoDT proposes a hypermedia layer that integrates existing DTs, regardless of whether they are implemented using industrial standards. Since most interoperability standards emerged from the Industry 4.0 movement and primarily target industrial domains, they leave a gap when modeling entities and concepts across heterogeneous domains. The HWoDT addresses this limitation by relying on Web ontologies that span a wide variety of application areas and can be seamlessly combined within a unified KG.

Providing a synchronized representation of the asset state is a defining feature of DTs. The HWoDT, designed from a DT-centric perspective, requires each DT to continuously produce and update a representation of its observable state in the DTKG. This enables consumers to query an up-to-date view of the entire ecosystem and observe state changes. Although similar synchronization can be implemented by AAS servers (e.g., Eclipse BaSyx⁸) retrieving data from OPC-UA endpoints, such approaches remain limited by the expressiveness of the underlying APIs and the lack of hypermedia-driven navigation. Trends in the DT engineering community show that service-oriented and micro-service architectures are becoming increasingly common in DT implementations [FBB22], making the HWoDT a natural fit for these modern architectural patterns. Web-based approaches describing APIs with ontologies are emerging, but they are often limited to specific domains and use cases (e.g. [LJJ20]), and not yet adopted as a general solution for DT interoperability as proposed in this thesis.

Leveraging the HWoDT, developers can focus on application logic while ab-

⁸<https://eclipse.dev/basyx/>

stracting away the complexity of heterogeneous DT integration. This yields advantages similar to those promoted by the WoT approach: *(i)* the application layer remains stable and unaffected by the introduction of new DTs; and *(ii)* the DT layer can evolve or replace underlying implementations without impacting upper layers, as long as the exposed interface is preserved.

The WoT is a source of inspiration for the HWoDT and the proposal in this thesis aims to extend the WoT principles to the specific requirements of DTs and their ecosystems. Related approaches also exploit the WoT in the context of DTs with similar goals. A semantic extension of the WoT ontology is presented in [GCB⁺24] proposing to use TDs to integrate DTs in the construction domain, extending it to capture the five-dimensional model of DTs [QTH⁺21]. The WoT is also the foundation of WoTwins [STM⁺22] a framework to generate automatically DTs starting from existing WoT things, modeling their behavior and enabling prediction features. These works highlight the relevance of the WoT as a standard for the future of DT applications, despite its limitations in capturing fully the nature of a DT in its entirety which may require extensions and development of custom ontologies aligned with the WoT model.

A fundamental difference with the (semantic) WoT compared to DTE that this thesis exposes, however, lies in the seamless browsing of the DTE through a KG that explicitly captures the observable state of DTs. In a W3C WoT application, even if TDs were semantically annotated and stored in a KG, selecting *things* based on their current state would require issuing requests to retrieve the value of each property of each thing. The WoT does not natively address the DT-specific requirement of continuously shadowing the asset state, nor does it specify how to manage dynamic relationships among things, limiting the expressiveness of cross-asset queries.

The HWoDT leverages the intrinsic synchronization capability of DTs through the DTKG, simplifying interaction and enabling direct, state-aware navigation. Without such a unifying abstraction, consumers interacting with heterogeneous DTs would need to individually retrieve, interpret, and merge their state representations to resolve a query (Figure 7.8).

Overall, the HWoDT is compatible with and complementary to existing DT interoperability efforts, positioning itself as an additional semantic and hypermedia

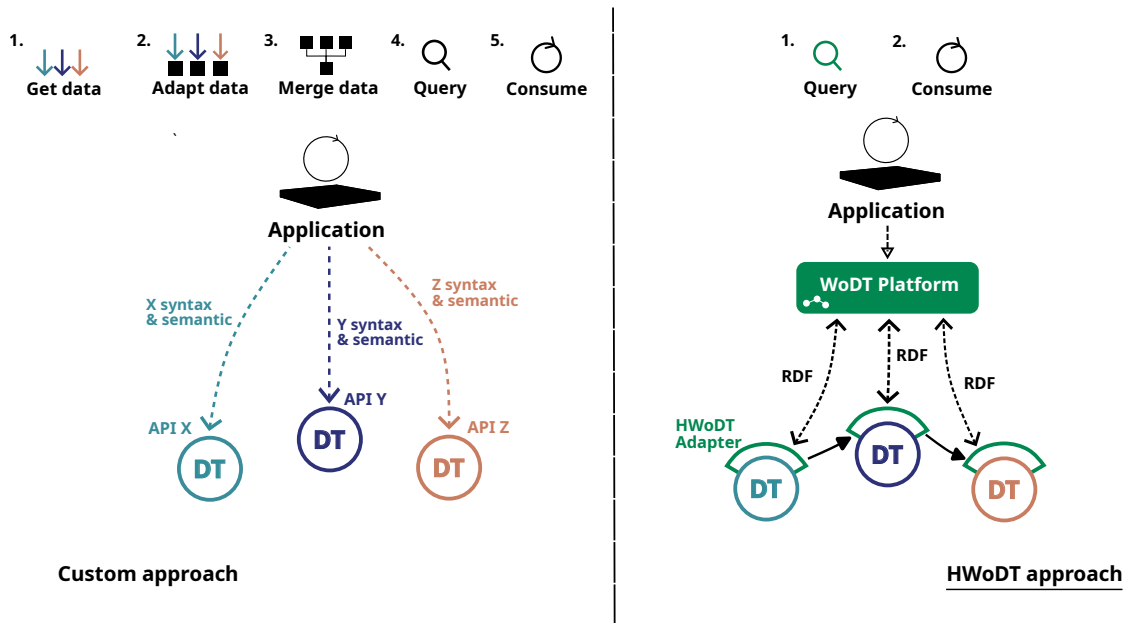


Figure 7.8: Comparison of the steps required to perform a “query” across heterogeneous DTs versus with the HWoDT approach, which abstracts heterogeneity through a uniform interface.

layer that integrates DTs implemented with—or without—industrial standards, supporting:

- integration with DTs beyond the industrial domain,
- advanced semantic querying with reasoning,
- Web-based navigation and direct access to up-to-date asset representations, and
- subscription to changes in the DT state.

7.3.2 HWoDT in the Landscape of DT Platforms

The HWoDT approach offers a solution to implement open DTEs on top of heterogeneous DT technologies. This contrasts with other DT platforms, which typically support homogeneous DTEs and allow multiple DTs to be defined within the same instance, but constrain all of them to specific technological choices.

To frame the positioning of HWoDT within the broader DT ecosystem, Table 7.2 compares a selection of representative platforms among reference DT technologies (Section 2.3) that support the construction of DTEs along key dimensions.

These comparison dimensions adopted are chosen to highlight differences between the different approaches, both in terms of design purpose, structural and operational characteristics, which are critical for analyzing how platforms support the construction of DTEs. The main comparison dimensions are:

- **Primary Role:** indicates whether the platform functions primarily as a *registry* or a *runtime* for DTs.
- **Deployment:** the technological and operational model used to deploy the platform itself.
- **Governance:** whether the platform adopts a centralized or decentralized management of DTs. Centralized governance typically implies that DTs are created and managed within the platform, while decentralized governance allows for independently created DTs to be registered and integrated.
- **DT Distribution:** the approach used to deploy DT instances across the system. Namely, this dimension distinguishes between platforms that require DT instances to be deployed within the platform itself (instance-based) and platforms that allow DT instances to be distributed and managed independently, with the platform serving as an integration layer.
- **Platform Coupling:** whether DTs exist only within the scope of the platform (bound), must be aware of being part of the platform (aware) or can be completely platform independent, regardless of the deployment model. This dimension is critical to understand the level of openness and flexibility in integrating DTs from different sources.
- **DT Representation:** the supported representation format used to describe and catalog DTs within the scope of a platform.
- **Synchronization Management:** whether and how DT state management is handled by the platform or must be implemented externally.

Table 7.2: Comparison of Digital Twin Ecosystem Platforms Along Key Architectural Dimensions

Comparison Dimension	Azure Digital Twins	Eclipse Ditto	WLDT [PMZ21]	DT-as-a-Service [TGM+23]	TwinBase [AST21]	HWoDT Platform
Primary Role	Registry	Registry	Runtime	Runtime	Registry	Registry
Deployment	Cloud (PaaS)	Microservices	Containerized	Microservices	Git-based	Containerized
Governance	Centralized	Centralized	Centralized	Centralized	Decentralized	Both
DT Distribution	Distributed	Distributed	Instance	Instance	Distributed	Distributed
Platform Coupling	Aware	Aware	Bound	Bound	Aware	Independent
DT Representation	Custom (DTDL)	Standard (WoT)	Any	Any (Service-defined)	Custom (DT Document)	RDF (DTD + DTKG)
Synchronization Management	External (Azure Services)	External (IoT middleware)	Internal (Shadowing)	Internal (Service Orchestration)	External (Git)	Internal (DTKG Observation)
DT Relationships	Explicit	Implicit (Attributes)	Explicit	Explicit	Explicit	Explicit
Data Retrieval	Query (Custom Language)	Query (Resource Query Language)	Any (Adapters)	Any (Service-defined)	API	Query (SPARQL)

- **DT Relationships:** whether relationships between DTs are explicitly modeled or implicit.
- **Data Retrieval:** the mechanisms provided for querying or accessing DT data.

The selection is not meant to be an exhaustive collection of DT development platforms for which we refer to other surveys [GMGL24b, LPT⁺22, MYH⁺22], but rather a representative sample of the state of the art, focusing on platforms that may support the construction and management of multiple DTs. Accordingly, the comparison includes the following platforms analyzed as follows:

Azure Digital Twins⁹ is a cloud-based PaaS solution that supports the definition of multiple interconnected DTs instances with an explicit notion of relationships between them. Azure Digital Twins is selected as a representative of commercial cloud platforms. A similar general-purpose platform is AWS IoT TwinMaker¹⁰, or other industry-specific platforms such as Siemens Insights Hub¹¹, Bentley iTwin¹², and PTC ThingWorx¹³. Azure Digital Twins functions primarily as a registry of DTs, which explains its strong support for cataloging multiple DTs and managing explicit relationships. Its cloud PaaS deployment ensures distributed availability, while centralized governance guarantees full control over DT creation and lifecycle. DT instances are platform-aware, meaning they rely on Azure-specific constructs, and Azure’s custom Digital Twin Definition Language (DTDL) is used for representation to support complex modeling, though at the cost of cross-platform interoperability. Synchronization generally relies on external Azure services, and data retrieval uses a dedicated query language optimized for the platform.

Eclipse Ditto¹⁴ is an open-source microservices-based registry for IoT DTs. Ditto is selected as a representative of open-source platforms that integrate with

⁹<https://azure.microsoft.com/en-us/services/digital-twins/>

¹⁰<https://aws.amazon.com/iot-twinmaker/>

¹¹<https://plm.sw.siemens.com/en-US/insights-hub/>

¹²<https://www.bentley.com/en/products/brands/itwin>

¹³<https://www.ptc.com/en/products/thingworx>

¹⁴<https://www.eclipse.org/ditto/>

existing IoT ecosystems. FIWARE [CSB⁺19] is another similar platform that has been used in the DT context [CMAA⁺22], although more focused on data management and integration rather than DT modeling and lifecycle management. Ditto's role as a registry reflects its focus on managing DT metadata rather than executing their behavior. Ditto is implemented as a microservice application that can be deployed in various environments, allowing integration into diverse IoT infrastructures. Governance is centralized within the Ditto instance, while DTs can be deployed as distributed components implementing the platform-aware synchronization process through IoT middlewares. Representations use the custom *Ditto thing* model, but can also expose WoT TDs for standard compliance, DT relationships are implicit, inferred from attribute structures. Data retrieval is supported through the Resource Query Language ¹⁵.

White Label Digital Twins [PMZ21] is a research-oriented framework for developing DTs as standalone software components. WLDT is selected as a representative of research prototypes which explores the ability to deploy DTs in a variety of infrastructures and here is considered in its ability to run multiple DTs instances within its multi-threaded runtime (see Section 5.5). Here it is compared to the HWoDT to highlight the complementarity of two contributions of this thesis, with WLDT providing a flexible runtime for DT execution and HWoDT offering a unifying layer for heterogeneous DT integration and ecosystem management. WLDT serves primarily as a runtime for executing DTs, deployed in containerized environments for flexible infrastructure support. Governance is centralized, as DT instances are bound to the runtime, unable to operate independently. The framework allows any representation format, facilitating experimental setups through the modular digital adapters introduced in Section 5.2. Synchronization is handled internally through the shadowing process, relationships are explicitly defined although manually handled by developers, and data access is adaptable via custom adapters, reflecting its research-oriented flexibility.

DT-as-a-Service Platform (DTaaS) [TGM⁺23] is a research-oriented platform with service-level orchestration to manage the lifecycle of multiple DTs in-

¹⁵Resource Query Language Project: <https://github.com/persvr/rql>

stances. DTaaS is selected as a representative of research prototypes which explores the ability to orchestrate and combine services to implement and run DTs instances. A similar approach is taken by the OpenTwins platform [RMD23] which extends Ditto with a composition of additional services. DTaaS operates as a runtime platform orchestrating DTs as services and supporting both the development and execution of DT models e.g., through co-simulation services [LFW⁺16]. Its microservice deployment supports flexible service composition, while governance remains centralized to control service-based DT lifecycles. DT instances are bound to the service runtime, and representation is service-defined, allowing custom but platform-specific models. Synchronization occurs internally through orchestration, relationships are explicitly modeled in service definitions, and data retrieval is determined by implemented service APIs.

TwinBase [AST21] is a Git-based registry promoting Web standards and interoperability. TwinBase is selected as a representative of research efforts that support the management of multiple DTs with a decentralized approach to governance following the idea of a *Digital Twin Web* that is close to the vision pursued in this thesis. TwinBase’s design emphasizes registry functionality with decentralized governance, allowing independently created DTs to be registered and integrated. Its Git-based deployment supports distributed DTs that must be aware of the platform to update their representation. DT representation uses custom DT documents (currently as YAML documents), synchronization relies on Git versioning, and relationships can be explicitly defined linking DTs within the platform. Data access is provided via APIs, reflecting the platform’s focus on openness, interoperability, and adherence to web standards, but limiting complex interactions and query capabilities, reflecting the more static nature of the platform compared to the other approaches.

HWoDT Platform The HWoDT platform is the approach proposed in this thesis to implement hypermedia-based DTEs on top of heterogeneous DT technologies. The HWoDT platform functions primarily as a registry of DTs, supporting the management of multiple DTs instances and their relationships within a unified ecosystem. It is deployed in containerized environments, which allows flexible

distribution across heterogeneous infrastructures. Governance can be managed as centralized to limit the ability to register DT instances or decentralized to allow more flexible participation of DTs in the ecosystem. Regardless of the governance model, DTs themselves are fully independent, meaning they can exist and operate outside the platform’s direct control as long as they adhere to the uniform interface, which can be implemented as an external adapter as demonstrated in Section 7.2. The only platform-aware behavior required from DTs is to expose the URI of the platform in their DTD to enable discoverability of ecosystems the DT is part of. HWoDT uses RDF-based representations combining DTD and DTKG, supporting semantic interoperability and explicit modeling of relationships between DTs. Synchronization is managed internally through the DTKG observation mechanism which ensures that the platform maintains an up-to-date representation of the state of all registered DTs. This inversion of control compared to other platforms, where the platform actively retrieves updates from DTs, allows for the integration of independently developed DTs and simplifies the synchronization process. Data retrieval is performed via SPARQL queries, enabling rich semantic access.

Compared to the other platforms, this positions HWoDT as a bridge between centralized commercial clouds like Azure Digital Twins, open-source registries like Ditto, research runtimes like WLDT and DTaaS, and decentralized approaches like TwinBase. It combines their strengths – support for multiple DTs, cross-platform relationships, semantic expressiveness, and operational flexibility – while overcoming typical limitations such as platform lock-in, constrained representation formats, and lack of independent DT deployment. In the broader DT ecosystem, HWoDT thus occupies a unique position as a heterogeneous, open, and interoperable platform, capable of supporting both experimental research and operational deployments across diverse infrastructures.

7.4 Prototype Performance Evaluation

A performance evaluation of the prototype is carried out to quantify the performance implications introduced by the additional layer introduced by the HWoDT to implement DTEs. Figure 7.9 schematically shows the communication between components in a HWoDT-based system. ① is the latency between PA and DT, ②

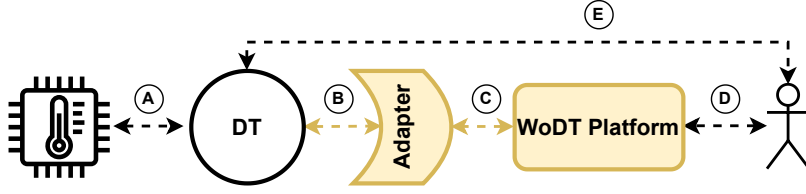


Figure 7.9: Latency in an HWoDT-based system. From left, to right, the PA, the DT, the HWoDT adapter, the WoDT platform and the client. Each connection between components (labelled with letters) may introduce latency.

is the latency between the DT and the HWoDT adapter, $\textcircled{C}$ is the latency between the adapter and the platform, while $\textcircled{D}$ is the latency between the platform and the client. In a scenario where the DT is used without the platform $\textcircled{E}$ would be the latency between the DT and the client. This shows that the additional potential latency loss in a HWoDT system is given by $\textcircled{B} + \textcircled{C}$.

Accordingly, the following evaluation assesses the performance of the HWoDT platform prototype, thereby characterizing the overhead incurred when integrated atop existing interoperability standards. Even though performance optimization was not a primary focus in the development of the HWoDT platform prototype, it can still be useful to make some considerations on the performance of the HWoDT approach. To do so an experimental setup is used to measure the latency $\textcircled{B} + \textcircled{C} + \textcircled{D}$ when interacting with the platform and compare it with $\textcircled{E}$.

7.4.1 Experimental Setup

The experiment is designed to stress the performance of the platform in a resource-intensive scenario, and measure the delay perceived by consumers wishing to interact with DTs and the ecosystem as a whole. A synthetic workload is used because—differently from a real-world IoT dataset which may include data spikes over time—the scenario allows us to have a fully controllable environment to scale the traffic and evaluate the response of the platform under different loads.

A DTE of a network of temperature sensors is implemented, with DTs reporting an update every second using DTs implemented with the WLDT framework, running within the same process. Each DT generates a random temperature value between 0 and 100 every second, with a random initial offset of up to one second

to avoid perfect synchronization of the DTs' data streams. The DT state includes a timestamp marking when the state was computed by the DTs. The different interaction patterns of the DTE are tested:

- with clients *observing* all the DTs receiving all the DTs state notifications;
- with clients periodically *querying* the DTE to retrieve all sensors that have a reported temperature greater than 50.

The key metrics collected are the *freshness* of the information, which is computed as the difference between the time a state notification is received and the timestamp of the message (or the greatest timestamp when observing the DTE KG). Additionally, the time taken to perform a query on the ecosystem retrieving the state of all DTs matching a condition is measured.

The experiments have been carried out running the DTs the WoDT platform and the client applications on the same machine equipped with a 13th Gen Intel(R) Core(TM) i7-13700H CPU and 32 GB of RAM, using the OpenJDK 21.0.7 runtime. Data is collected by analyzing logs of the client applications.

All DTs have been implemented with the WLDT framework. The framework supports defining DT implementations and uses a plugin-based architecture to expose different interfaces. This has been useful to implement different interaction patterns for the same DT.

Measurements are repeated with 50, 100 and 200 DTs, and performing operations with 1, 50, 100 clients each observing all DTs and querying the entire DTE.

7.4.2 Results

This section briefly reports results of the performance evaluation focusing on the different interaction patterns of the WoDT platform prototype and comparing them with baseline direct interactions with individual DTs. The results are summarized in Figure 7.10, showing two bar plots reporting the average values of the two metrics collected in the experiments. Results confirm that the overhead introduced by the HWoDT platform is not negligible, but still acceptable for many DT

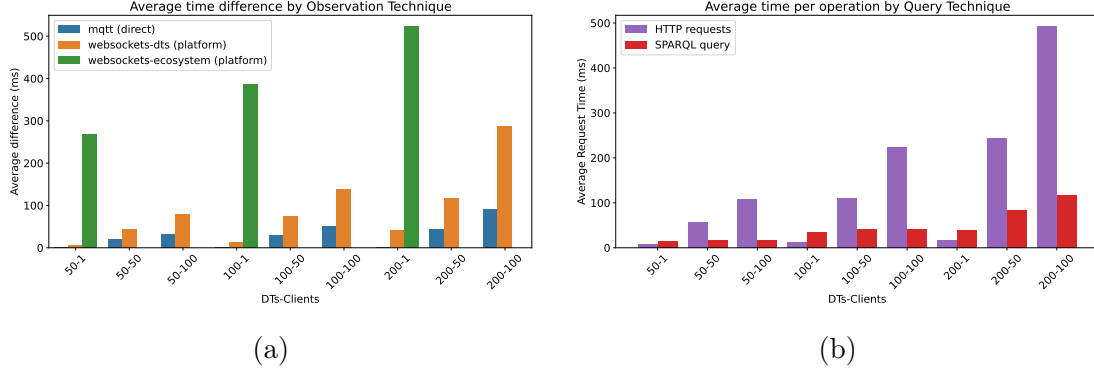


Figure 7.10: Performance indicators with different loads of DTs and clients working on the ecosystem: (a) average time difference when observing DTs either directly or through the WoDT platform, and (b) average time per operation when querying the DT ecosystem either through the WoDT SPARQL endpoint or directly retrieving the state from each DT API.

applications, especially when considering the benefits of having a uniform interface to interact with heterogeneous DTs, especially when considering the ability to query the whole DTE KG.

Digital Twins Observation

The HWoDT adapter sends updates of the DTKG on a WebSocket connection to the platform. Clients are set up to connect to the platform and observe each DT through a WebSocket connection. This aims to measure the delay introduced by the communication between the adapter and the platform, before getting to a final consumer. As the platform also supports observing the full DTE KG, a client also observes directly the KG updates over a WebSocket connection ((B)+(C)+(D)). This is compared against direct observation of the DT, which is implemented by sending each state update to an MQTT broker. Clients connect to all topics `dt/+state` to observe all DTs ((E)).

Whenever a message is received, the *freshness* metric is computed as the difference between the timestamp of the message and the time the message is received using the system clock (which is synchronized as all processes run on the same machine). For the DTE KG as a new state update is generated for each change of each DT, the greater timestamp among all DTs is taken to compute the freshness.

Figure 7.10a shows the average time difference with the different observation techniques. As expected, directly observing individual DTs is more efficient, as there is virtually no overhead. Users of the WoDT platform should prefer this whenever possible, following the affordances of DTs in the DTD. In all scenarios, the delay grows with the number of DTs. This is especially evident when considering the observation of the whole DTE KG, which is the least efficient. Analyzing the performance of the prototype it was possible to deduce that this is caused by the serialization of the whole KG to stream it at every update, which is computationally intensive. This is also the reason why the figure avoids reporting the observation of the ecosystem KG with more than 1 client, as the results are significantly worse than the other options, even with just 1 client. Performances are expected to be less drastically impacted when using smarter KG update notification strategies—such as sending incremental changes (e.g., as in [RAA⁺18])—at the cost of requiring more complex clients, capable of applying the changes to reconstruct the full state.

Repeated Querying

Clients can query the DTE through the SPARQL endpoint of the WoDT platform. This is compared with performing an interrogation directly on all DTs to retrieve the current state and then filtering it on the client side which is implemented by having all DTs expose a simple HTTP endpoint `GET /state` to retrieve the current state of the DT (similarly to what is shown in Figure 7.8) For each client, requests are sent every second, ideally, since DTs send updates at the same frequency, this would be a realistic way to monitor the sensor network in our scenario.

The overall time the operation takes is measured by computing the time of sending the request and the response for the SPARQL query, and the time of performing all HTTP requests and then filtering for the direct interaction. The time of performing SPARQL queries ((B)+(C)+(D)) is compared against the time taken by a client to perform an HTTP request to retrieve the state of each DT and filter them on the client side (E). Figure 7.10b shows that the average time taken to perform a query is significantly less than requesting the state on each DT, especially when the number of DTs in the ecosystem grows. Notably, when

running only 1 client, the overhead of the computation of the platform KG makes using queries less effective.

Comparison of Interaction Patterns

Results indicate that, with large numbers of DTs, using SPARQL queries becomes a more scalable and effective approach for observing the evolution of the ecosystem over time, comparable with direct observation of all DTs. While this approach may not capture every individual update from each DT, it reliably provides the latest available state at the time of each query. In this experimental setting, for instance, querying every second is realistically sufficient to capture almost all updates, as the client would match the update frequency of the sensors. This trade-off favors scalability and responsiveness, especially in dynamic settings where maintaining subscriptions for all DTs would be resource-intensive or impractical.

The centralized implementation of the platform makes it a potential bottleneck. Specifically, interacting with the DTE KG is costly because the graph gets updated by all DTs updates. This is a reasonable trade-off as interacting with the DTE provides advanced functionalities that would be impossible or very difficult to replicate simply by interacting with individual DTs. Nevertheless, it will be interesting to explore optimization techniques for future improvements of the platform, as well as decentralized approaches to provide ecosystem services that would make the system less reliant on the platform itself.

7.5 Final Remarks

This chapter presents a proposal for the engineering of heterogeneous DTEs grounded on principles and technologies from the Web and the Semantic Web. The chapter addresses the research question:

RQ2 How to engineer DTEs which integrate heterogeneous DTs and support applications in interacting with them? The proposal of the HWoDT addresses the challenges of integrating heterogeneous DTs by means of a uniform self-described interface that allows the representation of DT interaction possibili-

ties and metadata (through the DTD) as well as the up-to-date state of the DT (through the DTKG). The services and interfaces offered by the HWoDT platform allow consumers to seamlessly interact with the DTE, abstracting away the heterogeneity of the underlying DTs. Consumers can either observe the evolution of the DTE KG or perform expressive queries to retrieve information through SPARQL. The prototype implementation of the HWoDT platform and the set of adapters for representative DT technologies demonstrate the feasibility of the approach in integrating DTs developed with different technologies through a shared semantic layer that can complement existing interoperability standards.

Chapter 8

Operational Management of Digital Twin Ecosystems

This chapter contains contributions from the following works I authored: [BBM⁺ 24]

This chapter investigates **RQ3** by exploring the challenges in the operational management of DTEs that involve the management of multiple DTs from a deployment and runtime perspective. Several DTs need to be orchestrated on a cyber-physical computing infrastructure, ensuring that DTs are able to interact with the PAs they represent and maintain the target level of performance and synchronization. This problem is further exacerbated when considering that DTE can involve DTs developed for different platforms, and computing infrastructures that span the whole *edge-cloud continuum*.

Motivated by these challenges, this chapter introduces the concept of a *Digital Twin Continuum (DTC)*, a middleware that aims to facilitate the operational management of DTEs by providing a unified interface to manage the deployment and runtime of DTs across heterogeneous DT platforms and computing infrastructures. This proposal complements the HWoDT approach presented in Chapter 7, which focuses on the interaction with DTE, rather than their operational management.

The chapter analyzes the challenges in the operational management of DTEs, and presents a proposal for an architecture of the DTC middleware. The functionalities are supported by a set of *descriptions* that can inform the DTC about the characteristics of the DTs and PAs involved in the DTE and support stakeholders

in having a clear understanding of the system state at any stage.

8.1 Operational Challenges

An orthogonal aspect of complexity when developing a DT is understanding its non-functional requirements that allow for considering the DT effectively synchronized. The concept of *entanglement* [MLC20] has been used to define the level of synchronization of the DT with the PA and can be measured with metrics that include network latency and computation time on both the physical device and the computing node that is running the DT [BBF⁺24b].

In recent years, with the increased availability of computing resources at the edge of the network, the concept of edge-cloud continuum has emerged as a paradigm to distribute computation across a variety of computing nodes, obtaining benefits in terms of latency and optimizing the overall workload of an IoT system [RCVA22]. Like other IoT system components, DTs can also be deployed within the continuum [BBF⁺24a] with trade-offs in terms of entanglement, cost and resource availability. The advantages of this approach, though, are offset by additional complexity in the operational management of the software system. When it comes to DTs, this means choosing the suitable computing node to deploy the DT software, to ensure the Quality of Service (QoS) requirements are matched at runtime. This can be challenging due to the cyber-physical nature of DTs. Additionally, when several DTs are employed as parts of an ecosystem sharing the same computing infrastructure, it should be possible for operators to easily navigate the continuum and observe performance metrics to understand the potential impact of deploying new DTs in the system and eventually reconfigure it to balance the overall workload.

On the development side, the heterogeneity of DT platforms poses additional challenges as they may constrain the ability to deploy a DT across the edge-cloud continuum. Indeed, while open-source platforms can be adapted to run on a generic infrastructure, proprietary ones are usually Platform as a Service solutions, bound to the cloud infrastructure. Developers tasked with implementing a DT must hence evaluate the suitability of a given DT platform, taking into consideration both the functional (i.e., meta-model, features, services) and non-functional (i.e., latency,

computing power, privacy) requirements.

Since these requirements may vary significantly depending on the PA being modeled it is realistic to assume that a complex DT-based system may include DTs developed on different platforms. This has surfaced challenges in the management of the development and deployment of DT-based systems which may need to guarantee the QoS requirements of DTs developed with heterogeneous technologies and platforms while managing a shared pool of heterogeneous computing resources.

The proposal of the DTC aims to address the following goals:

- **reducing deployment complexity** of a DT-based system integrating DTs developed for different platforms sharing the same computing infrastructure.
- **facilitating the interaction of different stakeholders** with a DT-based system enabling easy access of relevant information over the development-to-deployment lifecycle of DTs.

8.2 Phases and Stakeholders

The process of developing and managing a DTE involves several critical phases, each essential for the effective development, discovery, selection, deployment, and operation of DTs. These phases abstract the development-to-deployment lifecycle of DTs in a structured approach which is independent of the target application domain or specific technological choices.

Identifying these phases helps shape the requirements for the DTC and understand the roles of the stakeholders that are involved in each phase. Figure 8.1 summarizes the main phases and the stakeholders involved in the development and management of DTs in a DTC that are described in the following.

DT Software Capabilities Discovery & Selection As a starting point, every DT system is originating from the needs of a cyber-physical context. This usually involves one or several *PA Owner* who are the stakeholders that possess the knowledge about the PAs and the goal that requires PAs to be modeled and digitalized [MSW24]. The *PA Owner* is also in charge of granting access to the

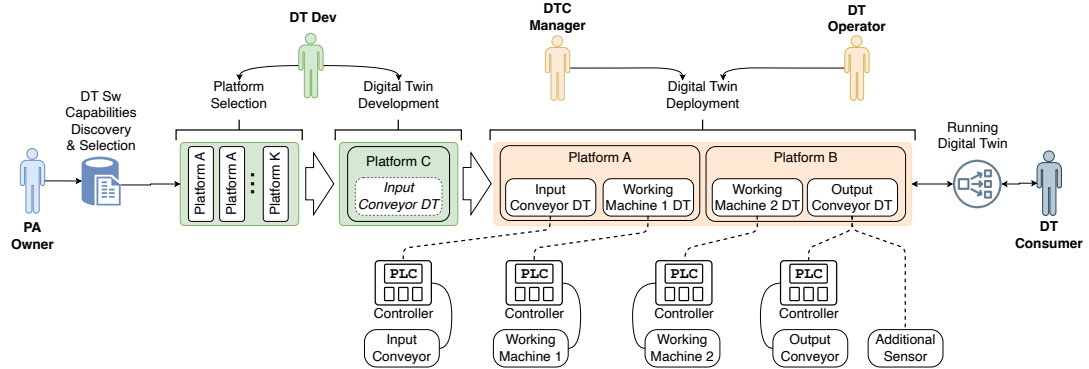


Figure 8.1: Stakeholders involved in different phases of the development and management of DTs in an ecosystem.

PA data and sensors. The first phase hence focuses on identifying DT capabilities that align with specific application and use case requirements. This phase involves defining the functionalities of the DT, associating it with the corresponding physical asset category, specifying communication protocols, and detailing its properties, events, relationships, and available actions. Notably, an asset might have multiple owners. For instance, an industrial machine is produced by a manufacturer and acquired by a company to employ it in one of its facilities. The same machine is then owned simultaneously by the manufacturer who may have control of telemetry data and be interested in monitoring the machine to offer predictive maintenance services, by the company who might want to keep track of all the machines across facilities and by the facility manager who might monitor the state of operational of the machine in the specific production line it is employed. All the owners may have access to different data and model the same asset with different target goals.

DT Platform Selection and Development A *PA Owner* may commission a *DT Developer* to implement the DT for the PA. The developer will implement a DT using a target technological stack, that depends on their expertise and the availability of the computing infrastructure that is available for the deployment of the DT. In this phase it is necessary to ensure that the chosen DT requirements can be effectively implemented on a specific platform. This phase involves defining platform-specific configurations, including implementation details, required config-

uration parameters, and technical requirements. Notably, the *PA Owner* is usually the (virtual) owner of the computing infrastructure on which the DT will be eventually deployed. When the DT is developed as commissioned by the *PA Owner* this may influence the technological choices of the *DT Developer*. A possible future may envision DTs implementations for specific kinds of assets made available to owners of such assets either publicly or commercially. These would be implemented either by manufacturers or a community of *DT Developers*. Such reusable DTs would then be implemented with a given technological stack and need to be deployed on the available resources of the *PA Owners*, in order to be configured and connected to the locally available PAs. Similarly, such reusable DTs could be offered as-a-service and made available only as instances deployed on commercial platforms. This adds a further layer of complexity to the management of the computing infrastructure running the possibly different DTs in an organization.

Digital Twin Deployment Once a DT is developed (from scratch or reusing existing implementations) the next goal is to run it on a computing infrastructure. This step involves a *DT Operator* who have knowledge about the DT requirements and can configure the computing infrastructure to make sure that the DT is deployed correctly and gets access to the PA data streams (this can be the same person as the *DT Developer* in a typical Dev-Ops fashion). Deploying a DT can be a challenging task, involving several steps that depend on the complexity of the deployment infrastructure and the requirements of the DT. This phase is crucial to ensuring that all deployment conditions are met, including availability of deployable artifacts, and correct setup of communication configurations, enabling interaction with both physical and digital entities. In a computing continuum, a DT may be deployed at different levels either on the edge, fog or cloud depending on the admissible trade-offs between network latency and computing power requirements. Notably, DTs might move across this continuum for a variety of reasons, either being connectivity requirements with the PA (and thus move horizontally in the continuum), or drops in the quality of service that require either scaling to the cloud or moving closer to the edge (and thus moving vertically in the continuum). Finally, the same PA could be connected to different DTs replicas, serving different use cases [MLC20]. Ideally, the DT Operator can delegate a re-

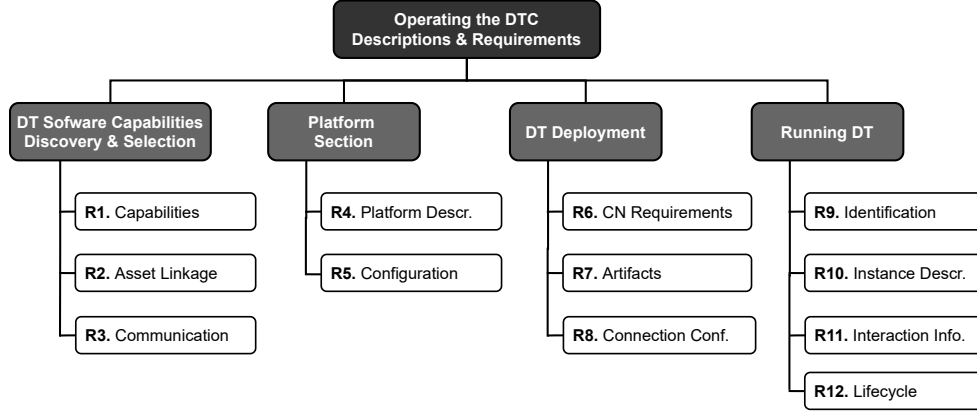


Figure 8.2: Requirements associated to each phase of the DT development-to-deployment process.

quest to a *DTC Manager* who has access to data about *other* DTs running on the same infrastructure and hence deploy the DT in order to use resources effectively. While a *DT Operator* is fundamentally interested in monitoring the activity and managing one or more DTs, the DTC Manager is instead interested in monitoring the activity and managing the computing infrastructure on which DTs are being deployed.

Running Digital Twin The last phase starts once the DT is up and running. In this phase, the DT must be identified, described, and reachable by *consumers*. This phase is critical for maintaining an accurate and up-to-date representation of the DT instance, managing its interaction protocols, handling lifecycle transitions, and enabling real-time state monitoring and interaction with the DT. The ability to monitor information about running DTs is essential for ensuring synchronization with physical assets, supporting advanced digitalization processes, and facilitating interoperability within complex cyber-physical systems. In this phase, generic *DT Consumers* may interact with the deployed DT instance. Such consumers can either be human users or other systems and applications, leveraging the DT features for their purposes. The *DT Consumers* are inherently also DTC consumers, as they may interact with the DTC to discover which DTs of which PAs are available, where to find them and how to interact with them.

Figure 8.2 summarizes the *information* requirements for managing DTs across

these phases, namely what kind of information is necessary to support stakeholders in each phase, and ensure that the phase can run successfully. These requirements guide the design of *descriptions* that can be used to inform the DTC about the characteristics of the DTs to deploy and manage, in a platform-agnostic way.

8.3 Key Elements and Descriptions

To address the challenges in the operational management of DTEs, the DTC relies on a set of key elements and their structured descriptions that provide a structured way to represent and manage the various components involved in a DTE.

At the core of this descriptive framework is the *Physical Asset Schema* (PAS), which serves to uniquely define a specific *type* of industrial asset—such as a robotic arm, conveyor system, or CNC machine. The PAS encapsulates essential meta-data that describes how assets of this category are expected to interact within a digital environment. This includes supported communication protocols (e.g., MQTT, OPC UA, HTTP), standardized API definitions, and more broadly, the configuration of interfaces and interaction models required for monitoring, control, and integration across systems. It is important to note that the PAS does not contain instance-specific values such as IP addresses, port numbers, or credentials. Instead, it provides a structured description of the expected capabilities and modes of interaction for devices of a given type. This is similar to the concept of *Thing Model* in the context of the WoT [MKK23] which only focuses on capabilities and abstracts completely from protocols and implementation details. The PAS, instead, includes this information, as it is more closely related to the physical devices and not on their virtual *Thing* representation.

This schema is essential for the DTC to understand how different PA conform to specific interaction patterns and how to configure communication with them.

Building upon the PAS, the *Physical Asset Instance* (PAI) represents a deployed instance of an asset as defined by the PAS. While the PAS defines the abstract capabilities and expected interaction patterns of a category of assets, the PAI provides the concrete, instance-specific configuration required for operational integration. This includes network information (such as IP addresses and ports), authentication credentials, and protocol-specific settings like MQTT topics

Entity	Description	Req.
Physical Asset Schema (PAS)	Defines the abstract capabilities and interaction patterns of a specific type of industrial asset (e.g., robotic arm, conveyor system). Provides essential metadata for communication protocols, API definitions, and interface configurations without specifying instance-specific values like IP addresses or credentials.	R1, R2, R3
Physical Asset Instance (PAI)	Represents a deployed instance of an asset as defined by the PAS. Includes instance-specific configurations like network settings, authentication credentials, and protocol-specific settings. Facilitates the runtime communication between the DT and the physical asset.	R1, R2, R3
Digital Twin Schema (DTS)	Defines the capabilities, behaviors, and structural components of a Digital Twin, linking the digital representation to its corresponding physical asset. Specifies properties, events, relationships, actions, and fidelity between the physical and digital counterpart.	R1, R2, R3
Digital Twin Package (DTP)	Defines the platform-specific implementation of the DTS, including code, configuration parameters, and dependencies required for deployment. Ensures the fidelity and communication requirements of the DTS are met during deployment.	R6, R7, R8
Digital Twin Instance (DTI)	Represents a running instance of the Digital Twin, including metadata for orchestration, monitoring, and lifecycle management. Tracks software lifecycle states and exposes relevant runtime metrics. Ensures the connection between the DTC and the deployed DT instance.	R9, R10, R11, R12
Runtime Platform (RP)	Describes the underlying computing infrastructure and environment where the Digital Twin is deployed. This includes details about the hardware, software, and network configurations that support the DT's operation.	R4, R5

Table 8.1: Overview of DTC Entities: Characteristics, Responsibilities, and Associated Requirements

or OPC UA node identifiers. These configurations are essential to enable runtime communication between the DT and the physical asset, supporting dynamic discovery, secure interfacing. As such, the PAI forms the operational bridge that links the abstract asset description to real-world deployment contexts, ensuring that the DT can monitor and interact with its physical counterpart in a precise and reliable manner. The WoT TD can be used for this scope as it includes the necessary information to describe how to connect to a specific device instance.

Following the same principles and responsibilities of the PAS, the *Digital Twin Schema* (DTS) defines the capabilities and behaviors of a DT independently of any concrete implementation. It establishes a reference to the corresponding PA category, ensuring semantic linkage between the physical and digital representations. The schema outlines the structural components of the DT, such as its properties, observable events, defined relationships, and available actions. It also specifies the target *fidelity* of the DT, capturing the mirrored functionalities of the physical counterpart within a given context and under specific application goals.

The *Digital Twin Package* (DTP) represents instead the software implementation of a given schema on a target DT platform. The package includes all platform-specific implementation artifacts required for deployment and execution. It incorporates the executable code, configuration parameters, and platform dependencies, ensuring compliance with the DTS's fidelity and communication requirements. It also provides detailed bindings for the declared communication protocols—e.g., how MQTT is used to interface with physical assets and how HTTP APIs enable interaction with external digital services.

When a package is deployed within the DTC, it gives rise to a *Digital Twin Instance* (DTI). This instance represents the active, running version of the DT within the operational infrastructure and includes all metadata necessary for orchestration, monitoring, and software lifecycle management. The DTI maintains references to its originating schema and the platform on which it is deployed, along with essential access information such as IP addresses, ports, and authentication tokens. It also exposes relevant runtime metrics—including CPU, memory, and network utilization—as well as any endpoints required to access its services and functionalities. In addition, the DTI tracks and exposes the DT connectivity information tracking whether the DT is connected to the PA. These lifecycle states

pertain strictly to the operational status of the deployed software component and are independent of the internal logic or functional behavior of the DT itself (differently from the synchronization lifecycle discussed in Section 5.3). Functional state and domain-specific behavior remain under the responsibility of the DT and are communicated through its defined interfaces in the digital space (e.g., through the DTKG if considering a HWoDT ecosystem Chapter 7)

With reference to the requirements identified in Figure 8.2, Table 8.1 summarizes the key elements and their associated responsibilities in supporting the operational management of DTEs. The integration of the PAS, PAI, and DTS effectively addresses the core requirements **R1.Capabilities**, **R2.Asset Linkage**, and **R3.Communication**, which are focused on the representation of features of the PAs and DTs, the relationship that links a DT to a specific PA, and the necessary communication protocols. These descriptions support the *DT Software Capabilities Discovery & Selection* phase, enabling stakeholders to explore available PAs and their corresponding DT types based on defined capabilities and interaction patterns. Additionally, the DTP supports requirements **R6.Computing Node Requirements**, **R7.Artifacts**, and **R8.Connection Configuration**, which pertain to platform-specific aspects to support the creation of DT instances within the DTC and can hence support the *Digital Twin Deployment* phase.

The DTI is responsible for addressing the requirements **R9.Identification**, **R10.Instance Description**, **R11.Interaction Information**, and **R12.Lifecycle**, which in the *Running Digital Twin* phase support stakeholders in the identification of the DT instance, the discovery of its status and interaction descriptions, as well as its connectivity with the PA during operation.

R4.Platform Description and **R5.Configuration** are supported by the *Runtime Platform* (RP) description. Differently from the other descriptions, these are tied to the computing infrastructure and the target platforms on which the DT can be deployed. **R4**, includes platform-dependent attributes such as the platform type, supported communication protocols, and data formats, enabling the DTC to match DTPs to the correct platform. Additionally, **R5** plays a crucial role in establishing the necessary parameters and operational requirements specific to the platform and implementation. Proper configuration ensures that the DT functions optimally by aligning its settings with platform constraints, communication

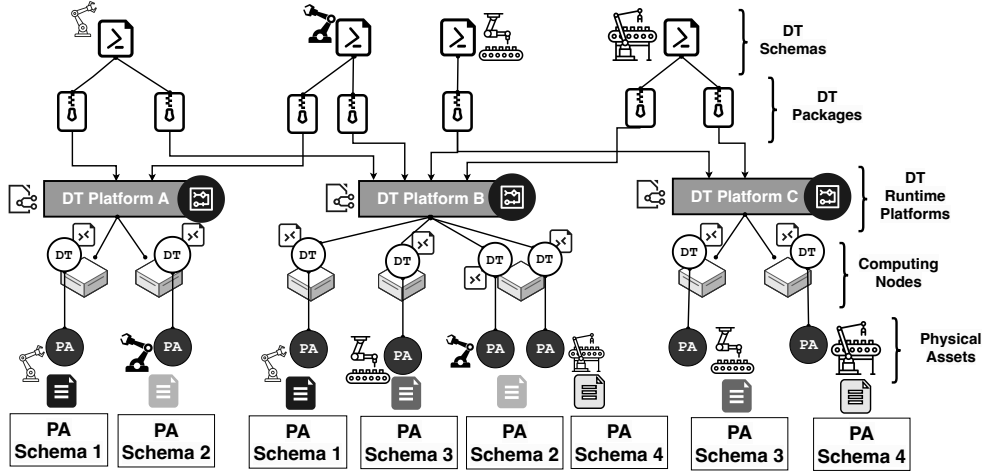


Figure 8.3: Key elements and their descriptions depicted within a schema of the DTC. PAs instances (bottom) are described by schemas and each have a DT instance mirroring them, running on possibly different platforms depending on which DT package among the ones implementing the correct DT Schema (top) is used to deploy the DT.

protocols, and performance expectations.

Together, these components support stakeholders to seamlessly identify and discover managed PAs, understand their capabilities, and determine the availability of corresponding DT packages for deployment. This structured approach ensures that the DTC facilitates efficient asset discovery, aligns with system interoperability needs, and supports dynamic integration across diverse environments. Figure Figure 8.3 visually summarizes the key elements and their descriptions within the context of the DTC, illustrating how they interrelate to support the operational management of DTEs.

8.4 Architecture of the Digital Twin Continuum

The aim of the DTC is to create a new open-system perspective of connected, interoperable and pervasive DTs modeled and engineered to create an effective cyber-physical abstraction layer. This requires, first, facilitating the execution of DTs on a set of DT platforms. Second, by exploiting the advantages of a broad compute continuum of cloud and edge resources, to deploy the DTs on the *best*

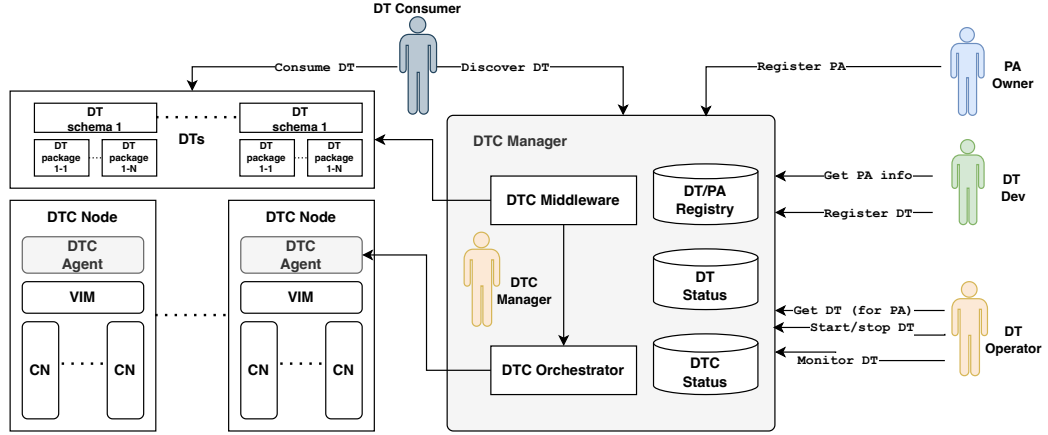


Figure 8.4: Functional Architecture of the DTC with the main components and the interactions with stakeholders. On the left, DTC nodes abstracting the interaction with physical computing resources. On top of them, DT packages belonging to a given schema are deployed as DT instances. The DTC Manager (right) is composed by a middleware and an orchestrator. The first is responsible for managing the registry of DTs and PAs, while the second is responsible for orchestrating the deployment of DTs across the DTC nodes.

Compute Node (CN) available to guarantee application requirements.

8.4.1 Functional Overview

The architecture of the DTC is depicted in Figure 8.4 and includes DTC functional blocks (grayed boxes), infrastructure modules, DT artifacts, and DTC stakeholders. From a high-level perspective, the DTC allows DT operators to expose a collection of M DTs for use by DT consumers. Each DT is described by a schema, which in turn is associated with a number of DT packages (up to N), enabling deployment on different platforms. New DT schemas and packages can be dynamically onboarded by DT developers.

In the DTC, we distinguish between *compute nodes (CNs)* and *DTC nodes*. A compute node represents an individual unit of computation (e.g., a VM, a bare-metal server, or a container host) where a DT instance is executed. A DTC node instead abstracts a collection of compute nodes orchestrated by the same Virtual Infrastructure Manager (VIM), similarly to how a MEC node in ETSI

MEC represents a set of compute hosts under a common VIM. Typical examples include Kubernetes clusters or OpenStack-managed virtualized clusters.

Within each DTC node, the DTC Agent is responsible for managing and monitoring the execution of DTs on the underlying compute nodes. The VIM handles low-level resource management, while the DTC Agent exposes the control and monitoring capabilities needed by the DTC Manager to orchestrate DT execution and placement across the continuum. Above these components, the DTC Manager coordinates all DTC Agents to provide a unified operational interface over the entire infrastructure.

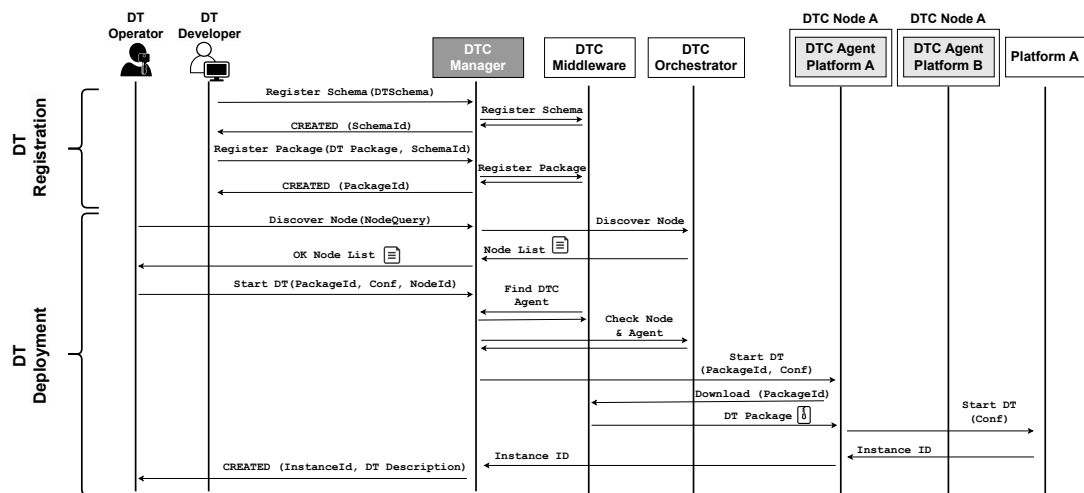
DT packages are used to create and execute DT instances on top of DT platforms. Each instance runs on a specific compute node inside a DTC node. DTs are deployed across the compute continuum by first selecting the appropriate DTC node (i.e., the cluster orchestrated by a VIM) and then, within it, the compute node that will host the instance. This separation enables the DTC Orchestrator to move and replicate DTs not only across compute nodes but also across different DTC nodes, depending on fidelity requirements, resource availability, and performance considerations.

More in detail, a DT operator requesting the creation of a DT corresponding to a given schema contacts the DTC Manager, which is responsible for identifying suitable platforms and corresponding DT packages, based on fidelity and connectivity requirements, and for configuring communication and computing resources accordingly. The DTC Manager comprises two logical components: the DTC Middleware and the DTC Orchestrator. The Middleware acts as the entry point for all stakeholder requests and configures DTs directly when possible. The Orchestrator is responsible for instantiating DTs by interacting with DTC Agents to configure the necessary resources through the underlying VIMs. This separation enables the DTC to support heterogeneous underlying infrastructures while providing a unified management layer.

8.4.2 Interaction Example

Figure 8.5 presents an illustrative example depicting a typical interaction scenario among the key components of the DTC architecture. Assuming that the PA has

8.4. ARCHITECTURE OF THE DIGITAL TWIN CONTINUUM



been already registered in the DTC providing the corresponding PAS and PAI descriptions, the figure illustrates the sequence of interactions that occur when a generic user aims to register and deploy a DT for a specific PA.

The subsequent step entails registering or creating a DT Package linked to the specific schema. The interaction flow follows is similar to the one above, with the DTC Manager receiving the request and the DT Middleware registering the package and associating it with the corresponding SchemaID.

Internally, the DT Orchestrator engages in the discovery process, returning a list of available nodes that match the user's specified criteria, such as the support for package deployment and the support for target hardware requirements (e.g., a GPU for the execution of specific DT's functionalities). The selection of nodes could be automated based on predefined policies. For simplicity, in this example, the user chooses the preferred node from the list of available options.

Upon reviewing the list of available compute nodes matching the target package and schema, the user can proceed by sending a start request to the DTC Manager. This request includes the specific package ID associated with the target schema, along with the starting configuration of the twin which include the identifier of the PAI to which the DT will be linked together with the node ID indicating where the DT should be deployed. The DTC Manager first verifies compliance of the DT package with the platform and the selected PAI, then it checks for the availability of the designated DTC Agent managing the specified node.

Subsequently, it forwards a start request to the target DTC Agent managing the selected platform and node. Here, the DTC Agent checks for the presence of the target DT Package and downloads it if necessary. Once the package is available, the Agent initiates and starts the target DT on the designated platform. Finally, the Agent receives the instance ID of the running DT from the platform and communicates it back to the DTC manager. Subsequently, the DTC manager replies with the instance ID to the user, along with the current description of the running DT instance.

This interaction flow exemplifies the collaborative efforts of the DTC components in facilitating the registration and deployment of DTs, ensuring a seamless experience for users while abstracting the underlying complexities of the infrastructure. By leveraging the descriptions introduced in Table 8.1, the DTC effectively manages the lifecycle of DTs, from registration to deployment, while ensuring compatibility with the underlying platforms and compute nodes.

8.5 Proof of Concept Implementation

A preliminary analysis, implementation, and experimental evaluation of the core functionalities of the DTC have been conducted using a prototype based on Eclipse

Ditto and WLDT as reference platforms. The primary objective of this implementation is to demonstrate the feasibility of the DTC and obtain initial insights by leveraging the microservice capabilities of the White Label Digital Twin (WLDT) library. The DTC's functional modules (Figure 8.4) are implemented in Python, featuring a *master API* that orchestrates overall behavior and a set of *DTC agents* that manage specific computing nodes and interface with underlying DT platforms: namely, WLDT and Eclipse Ditto for this experimental setting. These components expose structured RESTful APIs, enabling standardized interaction with the DTC core services. Monitoring and performance tracking across distributed nodes are facilitated through Prometheus, serving as a centralized time-series database, while local agents collect metrics on individual machines. Observability, visualization, and analysis are supported via Grafana dashboards. For the experimental evaluation, the supporting infrastructure – including Virtual Machines (VMs), networking, and container runtimes (Docker and Kubernetes) – was assumed to be preconfigured.

The Edge and Cloud infrastructures are associated with two real Proxmox clusters and deployments. The Edge node is realized as a server within the same laboratory as the microfactory used as the source of PA data, hosting the industrial Edge nodes. The microfactory setup is the same used in Section 5.6.1. Conversely, the Cloud component is realized using a second Proxmox cluster external to the laboratory network and connected via the Internet, ensuring that delays and latencies are aligned with non-local communication scenarios. On the respective Proxmox clusters, Linux Ubuntu Server 20.04 VMs were deployed, upon which the containerization environments were installed for the execution of all required applications, including Ditto, WLDT, and the monitoring and logging layers. Specifically, the Edge infrastructure utilized a VM provisioned with 2 CPUs, 2 GB of RAM, and 20 GB of storage, while the Cloud infrastructure was realized on a VM featuring 8 CPUs, 8 GB of RAM, and 500 GB of storage.

The **first experiment** examines the behavior of two DTs deployed across two platforms: WLDT at the edge and Ditto in the cloud. Rather than focusing on communication delays, this experiment aims to demonstrate how the DTC orchestrates state computation and ensures synchronization across multiple DT instances operating on heterogeneous platforms. Two observers, one on the edge

and one in the cloud, monitored the propagation of PA and DT states. In the first configuration (Section 8.5), both DTs run the same model version. The results confirm that the DTC effectively deploys the two instances which are getting data from the same physical asset and compute the state, with the expected delay difference between edge and cloud platforms. To each PA state update (green), the edge DT (WLDT) computes the new state (blue). Similarly the cloud DT (Ditto) receives the updates and computes its state (orange). The DTC ensures that both DTs are correctly deployed and synchronized, demonstrating its capability to manage multiple DT instances across heterogeneous platforms.

In the second configuration (Section 8.5), the cloud-based DT is switched to a different, coarser, DT schema (V2) that employs a longer time window for state aggregation (receiving updates from the PA every seconds and computing the state every 5 samples), while the edge DT continues operating with the original high-granularity schema. The DTC seamlessly manages this update, ensuring the cloud DT is correctly deployed and integrated without disrupting the ongoing edge operations. This illustrates the DTC's capability to orchestrate heterogeneous DT versions.

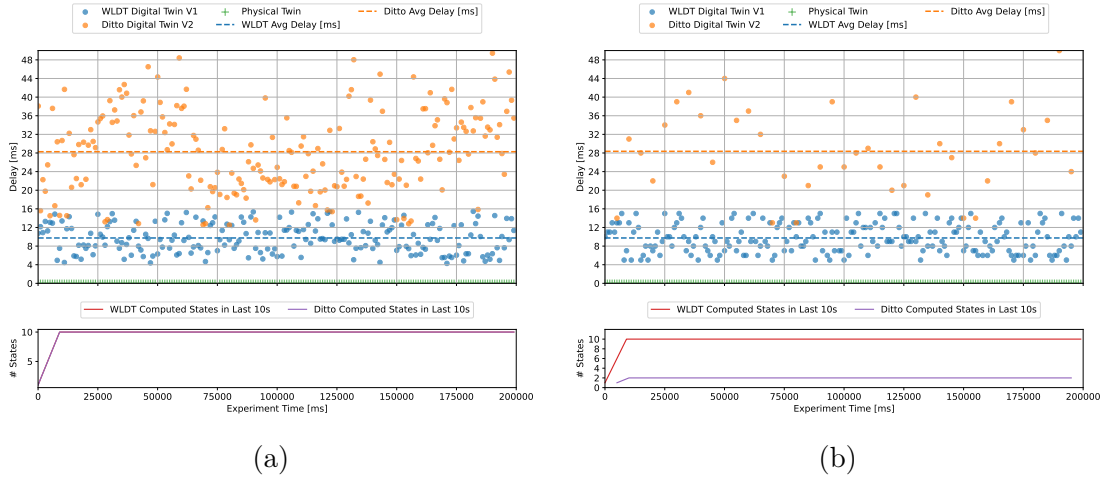


Figure 8.6: Comparison of end-to-end DT state computation delays across edge (WLDT) and cloud (Ditto) platforms. (a) shows the same DT model being deployed on both edge and cloud, while (b) shows a coarser DT model being deployed on the cloud. Below each figure, the number of DT states computed in the last 10 seconds, to highlight the difference in the update frequency of the two DTs.

The first experiment was engineered to address a realistic operational and architectural challenge facing stakeholders in industrial DT deployments: the management of asymmetric data granularity and synchronization requirements across heterogeneous Edge and Cloud environments. This asymmetry reflects a core business trade-off: high data frequency is strategic at the Edge to support time-critical applications, such as real-time predictive maintenance and rapid feedback loops, demanding maximum responsiveness. Conversely, a lower synchronization rate is often sufficient for the Cloud layer, typically supporting historical analysis, aggregate reporting, or non-critical dashboards. This selective reduction in frequency – a direct response to the stakeholder need to save bandwidth, storage costs, and processing load in the Cloud – is a key driver. The DTC is validated in this context by demonstrating its capability to deploy different DT schemas across distinct platforms.

The **second experiment** evaluates the scalability of the DTC prototype in a realistic DT ecosystem deployment involving multiple DTs that work together to digitalize the microfactory. This scenario directly addresses a key stakeholder challenge: the need for a phased and controlled deployment where visibility into the factory floor is progressively increased.

The deployment evolved incrementally over the experimental timeline, as shown in Figure 8.7, which depicts the number of active DTs on the edge and cloud platforms. Specifically, eight DTs were first launched on the edge, followed by the activation of four additional DTs in the cloud, achieving full deployment of all twelve DT types present in the Microfactory. Later, edge DTs are also replicated in the cloud and vice-versa, to simulate a scenario, where replicas are needed to support different use cases and applications. Bringing the total to 24 DTs running across the two platforms.

In this configuration, the edge DTs exhibited extended functional behavior compared to the first experiment.

They implemented more advanced DT capabilities, specifically augmenting the PA digitalization by performing complex tasks such as industrial machine state computation and anomaly detection for blocked products within the production line. Critically, the cloud DTs received their state input indirectly from their corresponding edge DTs, rather than retrieving data directly from the physical assets

in order to give to the edge DTs the responsibilities of the first digitalization points and controlling data flows and granularity. This setup highlights the DTC’s ability to coordinate multiple DT instances across heterogeneous platforms, maintaining synchronization, ensuring consistent communication, and optimizing resource utilization as the system scales.

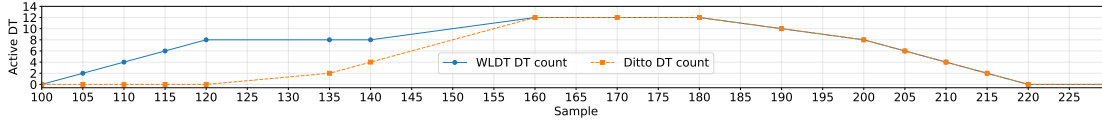


Figure 8.7: Number of active DT instances on the Edge and Cloud platforms during the incremental deployment process. The graph illustrates the gradual increase in the number of DTs over time, reflecting the phased deployment strategy adopted in the experiment.

Performance metrics—including CPU usage, memory consumption, and network traffic—were continuously monitored during this scaling phase. The primary objective is to demonstrate the flexibility and robustness of the proposed DTC system in seamlessly coordinating and orchestrating these evolutionary and incremental deployments in a realistic, heterogeneous environment. It is important to emphasize that Ditto manages multiple DTs within a single centralized platform composed of several core microservices (e.g., connectivity, storage, proxy, query, gateway), whereas WLDT deploys each DT as an independent microservice at the Edge. Consequently, in Ditto, the deployment of multiple DTs affects the overall platform performance, while in WLDT, each DT instance can be monitored independently. This fundamental difference in platform architecture introduces additional complexity, which is transparently handled by the DTC. By abstracting these platform-specific details, the DTC allows application designers to focus on defining schemas, packages, and the desired behaviors to be implemented, without needing to manage the underlying deployment differences. During the experiment, detailed measurements were collected for CPU and memory usage, along with inbound and outbound network traffic, for each active microservice—including the core components of Eclipse Ditto and the WLDT-based DT instances. These metrics provide insights into how system resources are utilized as the number of active DTs increases across Edge and Cloud deployments. The results, illustrated in Fig-

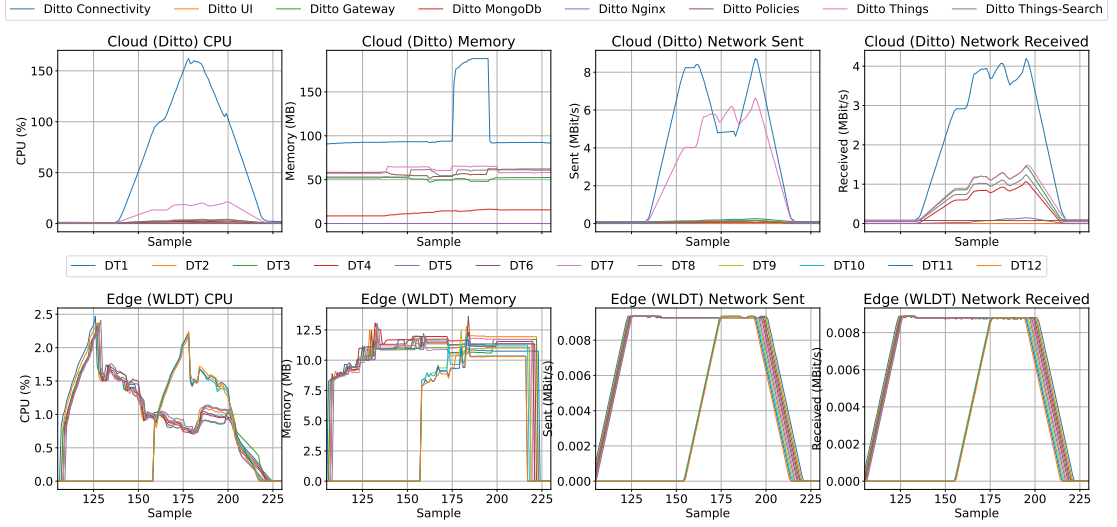


Figure 8.8: Evaluation of the DTC prototype enabling multi-platform DT deployment in the Microfactory. The figure compares resource usage (CPU, Memory, Network Sent and Received) for Ditto in the Cloud (top row) hosting multiple DTs within a single Ditto instance composed of several microservices (Connectivity, UI, Gateway, Nginx, Policies, Things, Things-Search) and WLDT at the Edge (bottom row) deploying individual DTs as microservices (DT1,..., DT12) as the number of active DTs increases.

ure 8.8, reveal the distinct operational phases of the experiment and demonstrate the DTC’s ability to coordinate and manage multiple heterogeneous platforms concurrently.

The obtained results underscore the feasibility of dynamically, modularly, and flexibly evolving an Edge-Cloud Digital Twin deployment within a realistic industrial context, thanks to the DTC’s structured and integrated approach. This successful evolution demonstrates the system’s capacity to accommodate new DTs over time, effectively matching emerging stakeholder requirements and catering to different observers and users of the DTs across local and remote Cloud environments. Crucially, the proposed DTC solution effectively masks the complexities of the various underlying platforms (WLDT and Ditto). Without such an integrated orchestration layer, the deployment, synchronization, and lifecycle management of these heterogeneous, cross-platform DT instances would entail a significantly greater level of complexity and manual effort for developers and system operators.

This abstraction layer allows users to seamlessly select the DTs of interest based on the architectural layer where they need to be deployed, the specific features and functionalities required, and the data sources to be utilized.

Overall, the experiment confirms both the feasibility and robustness of the DTC architecture. The results show that the DTC can seamlessly encapsulate the heterogeneity of underlying Digital Twin platforms, offering a unified and coherent system view across deployment layers. Furthermore, it maintains scalability, observability, and consistent system performance as operational load and deployment complexity increase—demonstrating its suitability for large-scale, distributed industrial scenarios

8.6 Final Remarks

This chapter presents the concept of Digital Twin Continuum as a middleware platform to support the operational management of DTE on a compute continuum. The DTC addresses key challenges that emerge when dealing with the deployment and management of systems that involve multiple DTs, characterized by heterogeneous physical assets, diverse DT platforms, and varying application requirements.

The proposal in this chapter contributes to answering the research question:

RQ3 How to support the operational management of DTEs? By using structured descriptions that capture the essential characteristics of physical assets, digital twins, and runtime platforms, the DTC enables stakeholders to effectively discover, deploy, and manage DT instances across a distributed infrastructure. The descriptions are designed to be platform and application-agnostic, allowing for seamless integration and interoperability among different DT platforms and use cases. The DTC architecture incorporates key functional components, including a middleware layer and orchestrator, which facilitate the interaction with distributed computing nodes through the abstraction of DTC agents. This approach supports different stakeholders in their respective roles, offering a coherent holistic view of the DT ecosystem alongside the development-to-deployment lifecycle of its components.

Part III

Intelligent Applications with Digital Twins and Multi-Agent Systems

Chapter 9

Multi-Agent Systems and Digital Twins

This chapter contains contributions from the following works I authored: [BMM⁺ 25]

This chapter tackles **RQ4** by analyzing how DTEs and MAS can be used synergistically to engineer intelligent IoT systems and applications. Since both abstractions have been used to encapsulate intelligence in IoT systems, it is crucial to clarify their respective roles and responsibilities to avoid confusion and overlapping solutions. This has been partially explored in the literature before (Section 4.5) but not yet systematically addressed from a software engineering perspective.

This chapter aims to fill this gap by providing a comprehensive analysis of the synergies between DTs and MAS. The resulting proposal is grounded in a set of design principles that help system designers decide when to use DTs, when to use MAS, and how to combine them effectively.

9.1 Complementarity of DTs and MAS

Autonomous Agents (AAs) and Digital Twins (DTs) are gaining increasing attention in the literature about the distribution of intelligence in IoT systems and applications (Section 4.5). AAs encapsulate reasoning routines and decision-making criteria tailored to a given goal. Hence, they naturally lend themselves to implement the perception-action feedback loop typically required in the IoT to realize

the “actionable knowledge” paradigm [BSH]. DTs digitalize an entity of interest in the physical world to decouple applications and services from the technical intricacies of interacting with such entities. Also, they may offer additional digital functions (i.e. *augmenting* the behavior of physical entities [MLC20]). Thus, they both naturally fit the IoT context in bridging together the “cyber” and the “physical” parts of a system. Consequently, to date, several approaches and architectures have been proposed employing either one abstraction or the other, or even both combined [MPR22, KC24, PMN⁺24], to enable *distributed intelligence* [RCVA22, AQHAA21].

However, it is not always clear what the *relationship* between these abstractions is at both the architectural and technological level: namely, whether AAs and DTs are interchangeable, (partially) overlapping, alternatives, complementary, or anything else. In those works that propose an integration, it happens in many different ad-hoc ways: for instance, with “multi-agent based DTs” to digitalize complex systems of the real world [PZVN⁺25], or with AAs using DTs as a digital substrate to deal with interoperability [RCM⁺22a]. As a consequence, there is an emerging *fragmentation*: the existence and proliferation of several solutions to the same problems, which have no clear relationships with each other, and that are conceived for specific applications.

In fact, as most of these approaches are proposed in the context of a specific application or intended goal, there is little interest in providing IoT systems and application engineers with general principles to guide their design choices regardless of the specific application domain and/or goal. Arguably, that this lack of clarity is problematic for the broader IoT community, as it leads to a variety of approaches that may tackle similar problems with different solutions, keeping closed vertical silos between partially overlapping communities.

This section briefly reports exemplary works that discuss the application of either AAs or DTs in IoT systems, highlighting their respective contributions to the design of intelligent functionalities. This effort goes towards highlighting the complementary roles of these abstractions in engineering intelligent IoT systems, paving the way for defining design principles to guide the separation of concerns between the two abstractions.

Intelligent Functionality	Representative Papers	Notes
<i>Logic inference</i>	[OC19] [LLS21] [LKR ⁺ 16] [CFG ⁺ 16]	Works applying logic-based approaches in IoT systems to improve explainability and deductive reasoning of the system components.
<i>Adaptive control</i>	[SFZ16] [LSC20] [CMM18] [LKR ⁺ 16] [ZQLL17] [TLWD18]	Works showcasing the effectiveness of adaptive behavior of agents in IoT settings, highlighting properties such as context- and self-awareness as well as planning for dynamic reconfiguration.
<i>Learning of control policies</i>	[RUR ⁺ 22] [YZW21] [LTZ ⁺ 20] [ZTZZ21] [LLC17] [ZMZ ⁺ 23]	Works showcasing applied scenarios of Multi-Agent Reinforcement Learning and Deep Reinforcement Learning to learn cooperation policies.
<i>Simulation</i>	[LN16] [BDD ⁺ 20] [LKR ⁺ 16]	Works describing simulations of IoT scenarios performed with a multi-agent based model to simulate the interactions between autonomous components.

Table 9.1: Selection of literature on AAs exploitation in IoT for delivering different intelligent functionalities.

9.1.1 Use of AAs and DTs in IoT Systems

Agents and MAS have been widely used in many domains primarily as a context-aware model and technology to encapsulate intelligent behavior in *distributed* and *reusable* software components (Chapter 4) This includes the IoT domain, that is a good fit for agents' situated decision-making abilities [SGP⁺20]: sensors and actuators provide the means to perceive and affect the real world (a physical environment), while agents encapsulate the system's *goals* and the reasoning processes needed to bring the system in the desired state of affairs—by closing the feedback loop between sensing and acting. From a distributed system perspective, MAS improves IoT by addressing autonomy and heterogeneity, enabling device and data discovery through semantic service descriptions, fostering trust through reputation and incentives, and facilitating collective intelligence [SC17].

Table 9.1 summarizes a selection of representative works that exploit AAs in IoT systems to deliver different intelligent functionalities.

Similarly to AAs, DTs are increasingly used to encapsulate intelligent functionalities that enhance the capabilities of the PAs they digitalize.

Intelligent Functionality	Representative Papers	Notes
<i>Prediction and Learning</i>	[EAG21] [HFZ ⁺] [QWYG] [QGZ ⁺] [ZWX ⁺ 23] [AKL ⁺ 23] [MJW22]	Prediction of future states of a PA is one of the most common intelligent functionalities delivered by DTs, together with simulation (see below). Such predictions are increasingly delivered via data-driven, machine learning techniques.
<i>Inference</i>	[XSLZ19] [SLV ⁺ 19] [QLZL24] [ZLQ24]	DTs are also used to synthesize novel information either based on historical data gathered about the PA, or by aggregating the data of multiple PAs that are so tightly coupled in the physical world to be worth digitalizing them altogether with a single DT.
<i>Simulation</i>	[BR16] [HB] [ET] [KFL] [DSH19] [WDW ⁺ 23] [QLZL24]	Simulation of alternate configurations or working conditions for a given PA is another typical application of DTs.

Table 9.2: Selection of literature on DTs exploitation in IoT for delivering different intelligent functionalities.

The relationship between DTs and intelligence in IoT systems has been widely explored in the literature (Section 2.4). Table 9.2 summarizes a selection of representative works that exploit DTs in IoT systems to deliver different intelligent functionalities. DTs can leverage data collected from their corresponding PAs to perform various intelligent tasks. The modeling capabilities that are intrinsic to DTs is often associated with simulation, forecasting and anomaly detection functionalities.

9.1.2 Intelligent Functionalities

The literature on distributing intelligence in IoT reports on several intelligent functionalities for which AAs and DTs are being actively used by system designers. Here they are summarized focussing on the *kind* of intelligent function they deliver, regardless of the specific task they accomplish (e.g. time series forecasting vs. fault prediction) and of the specific technique adopted (e.g. regression, SVM, etc.). This categorization is useful in defining what can be considered, “intelligence” in a practical, bottom-up way, stemming from related works in the area without getting trapped in philosophical arguments. Then, it is also useful to establish the *coverage* of such functionalities by AAs and DTs as emerging from Table 9.1 and

Table 9.2.

Prediction there including time series forecasting, recommendations, namely any form of reasoning meant to “guess” new information based on past and current knowledge. In the IoT, common prediction targets are machinery failures, stock levels, resource use, and system states.

Simulation encompassing any form of hypothesizing different states, in the past, present, or future. “What-if”-like analysis falls into this category. In the IoT, it is common to simulate the future states of individual things, for example, to improve the design of a product or a production pipeline. However, complete systems can also be simulated with an added degree of complexity.

Planning that includes any form of synthesis, and *practical reasoning* [Bra87], which is meant to figure out how to achieve some practical effect (on the target system) by properly sequencing available actions. Planning to achieve a given system configuration is common in the IoT, as well as planning to reconfigure after some disruptions.

Inference within which both *epistemic reasoning* [MvdH], that is, synthesizing novel information from known data, or data-driven inference such as pattern recognition and classification. This is perhaps the most common form of intelligence used in IoT, as even simple monitoring and control applications usually infer situations by aggregating different data coming from distributed sources of information. In addition, fault detection and diagnosis can be gathered in this category.

Adaptation instead gathers all the intelligent functionalities aimed at enabling the system to adapt to unforeseen situations that had not been explicitly managed at design time. As in the previous category, this one is quite broad and includes several heterogeneous approaches, ranging from engineered adaptation (e.g. the MAPE-K loop [KC03]) to learning-based methods (e.g. evolutionary approaches [Eib]).

Learning which despite not usually being the primary goal for a functionality, is a valuable tool to implement some of the ones highlighted above and still

poses important requirements on the architecture of a system that wishes to support any form of learning process in one of its components. That includes statistical machine learning, reinforcement learning, and causal structure learning. This category groups functionalities that aim to make the system, or one of its components, learn to do something. In IoT, the most common form of learning employs statistical machine learning to create prediction models, for example.

These high-level categories of intelligent functionalities will guide the analysis of how to distribute them across AAs and DTs in the rest of this chapter.

9.2 Design Principles

In an attempt to clarify the complementary roles of DTs and MAS in engineering intelligent IoT systems, a set of design principles are proposed in this section, to guide the analysis from a functional standpoint, i.e. focusing on the intelligent functionalities to be delivered by the system. The proposed principles can then be used to inform the design of architectural patterns that combine DTs and MAS effectively.

The principles are derived from the quite abstract software engineering criterion of *separation of concerns* given in [MPR22], expanding and refining it along three dimensions to make it more practically useful: *specificity*, *scope*, and *time*. These are meant to be used to analyze the requirements posed by the intelligent functionalities identified in the previous section.

Neither of these dimensions *alone* is sufficient to determine *univocally* which abstraction is best suited for a given functionality. Instead, they all need to be taken into consideration together when approaching the design of an intelligent IoT system as shown in Section 9.5.

9.2.1 Specificity

The first proposed principle is named *specificity*, whose spectrum is depicted in Figure 9.1, and is defined informally as



Figure 9.1: The spectrum of *specificity*: from intelligent functionalities specific to an application goal to general-purpose services.

how much a given (intelligent) functionality is *specifically* serving a given application goal (at one end of a spectrum), rather than being *generally* exploitable by multiple goals and potentially across application scenarios (at the other end).

This aims to help IoT systems and application designers to distinguish between

- features that are strictly tied to the achievement of a specific *application goal*
- features that may be *generically* useful to pursue multiple goals and build other intelligent functionalities and/or applications on top.

This is especially important for designing highly modular systems that can evolve over time.

In IoT systems, for instance, a widespread general-purpose functionality concerns the digitalization of PAs to enable intelligent functionalities such as prediction of that assets’ future states, and simulation of alternate states. Functionalities like these can be considered to be not tied to the application goals, but to the general services offered by the IoT platform, in an open-system, “as-a-service” perspective.

In contrast, functionalities such as fault diagnosis or inference of specific information may be relevant only in the context of the application domain or goal. Another example would be the implementation of adaptive control policies and decision-making strategies: these are likely to be specialized for the system at hand, as they encode the requirements and goals set by stakeholders.

For instance, in a smart building, there could be several control policies introduced to optimize comfort or energy consumption. Those policies will be part of the system due to a specific requirement coming from the stakeholders. At the

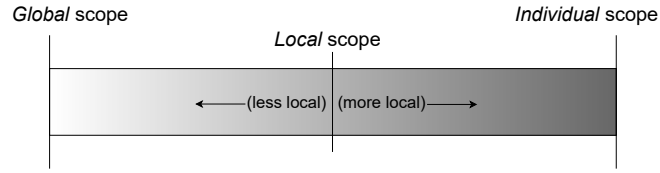


Figure 9.2: The spectrum of *scoping*: from intelligent functionalities requiring knowledge of the whole system, to those confined to individual assets.

same time, the building may also include functionalities that are more general-purpose, such as predicting the future energy consumption of the building based on historical data. This may serve multiple goals, such as optimizing energy use, improving occupant comfort, or reducing operational costs.

9.2.2 Scoping

The second proposed principle, is named *scoping*, and is defined as follows:

to what extent a given (intelligent) functionality requires data from the whole system (at one end of a spectrum) or a single individual entity of interest (at the other end) to deliver its results

This aims to help IoT systems and applications designers distinguish between:

- features that require data, inputs, and interactions with multiple entities of interest in the system at hand (up to *global* information, synthesized at the system level);
- features that require only considering a single asset (the extreme case of *local* information).

Figure 9.2 depicts this spectrum.

In any IoT system, due to its inherent connection with the physical world, there are software components whose main function is to digitalize individual, clearly identifiable domain entities or a specific portion of the physical world. Such software components are coupled with PAs, which set boundaries to the *scope* within which intelligent functionalities operate.

For example, a component might be designed to collect and train on local information only (e.g. prediction of the future states of specific machinery, not others). Conversely, many other functionalities may require a broader scope that goes well beyond such local boundaries. For instance in the domain of a smart home, software components may mirror, and grant access to, the individual devices (e.g. for remote monitoring and control), but there could also be adaptive control routines spanning multiple devices or even multiple rooms to orchestrate a coordinated action (e.g. preparing for the comeback of an inhabitant by activating the A/C, unlocking the doors, raising the curtains, etc.). Accordingly, the scoping principle is meant to suggest that system designers consider where the knowledge and data required to accomplish the intelligent function come from, as well as which entities will be subject to its effects.

It should be noted that the *individual* and *global* scopes are the two extremes of the spectrum depicted in Figure 9.2. Functionalities may require data from multiple entities, but such entities may be confined in a limited space, for instance geographically, or network-wise. Or, it could be the case that data is required from different and distant entities that are anyway somehow easy to access together—think of overlay networks. In this case, the proposed principle is still useful to guide design choices, as it forces designers to clarify and sort out why and how they deem the scope to be worth defining global or local (or even individual).

9.2.3 Timing

The third (and last, for the time being) proposed principle is what named *timing*, and defined informally as

how much a given functionality relies on any one notion of *time* (e.g. physical or logical) to be either explicitly (i.e. *time-aware*) or implicitly (i.e. *time-situated*) available

The concept of “time-aware”, is meant to convey that the functionality requires access to time-related information, either to make it directly observable to consumers of such functionality or to work properly. For instance, time series forecasting (a form of prediction) needs an explicit representation of time to be available, hence it is time-aware.

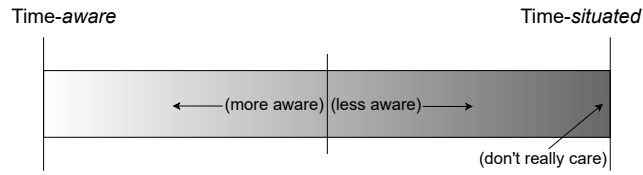


Figure 9.3: The spectrum of *timing*: from intelligent functionalities requiring explicit modelling of physical or logical time, to those that don’t care.

The concept of “time-situated”, instead, conveys that the functionality has some dependency on any one notion of time, but such dependency need not be explicitly captured and managed. For instance, real-time monitoring and control of an asset surely depend on time, but such a dependency need not be modelled explicitly to deliver the functionality. Simply, its existence suffices for the functionality to work properly. There are also some other cases where time does not matter from the designer’s standpoint, as time is not an issue when implementing the functionality—e.g., sending commands to an actuator device. This whole spectrum is depicted in Figure 9.3.

Of course, placing intelligent functionalities within this spectrum cannot be a precise operation, as with the other principles. But some general considerations about the categories of intelligent functionalities can and should be made to aid system designers. For example, prediction, simulation, and diagnosis are likely to require time awareness. How far into the future should an event or state of interest be predicted? How long into the future should a simulation extend? Conversely, when did a chain of faults lead to the present faulty situation? Other functionalities, such as inference, learning, and adaptation, may be less demanding and simply be time-situated. Of course adaptation actions are based on some events that happened, or on some predicted future state, and need to be carried out in the future, but possibly they do not require an explicit representation of time to work properly as simply their sequential ordering is sufficient to eventually react accordingly.

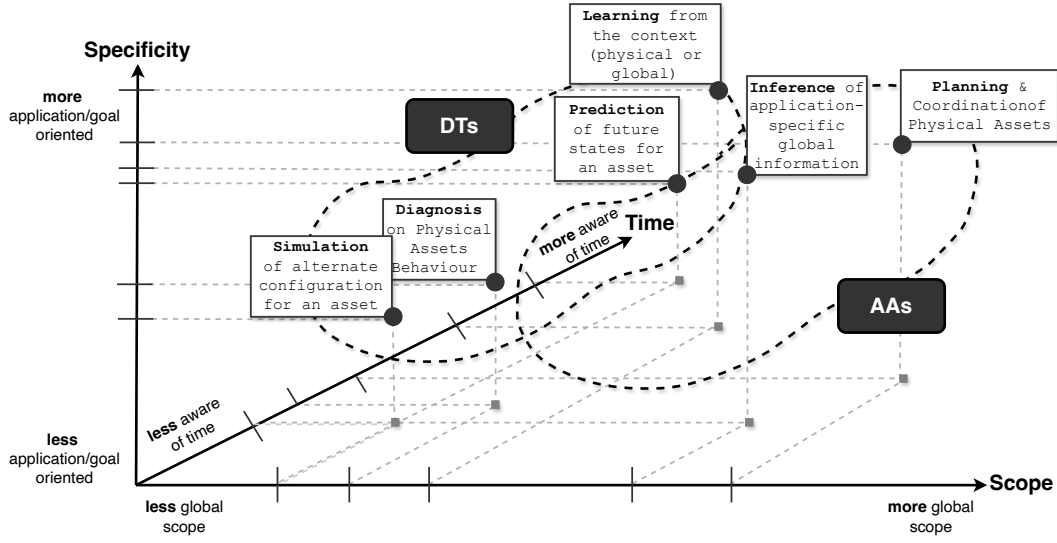


Figure 9.4: Exemplary intelligent functionalities placed within the spectrum of the proposed design principles. AAs and DTs are clustering those functionalities that they better match with, according to such principles.

9.3 Matching AAs and DTs to Design Principles

The design principles introduced in the previous section are here matched against AAs and DTs, to conceptually assess which abstraction is best suited for which end of the spectrum described above for a given principle. Figure 9.4 shows the exemplary functionalities and how they are better encapsulated by AAs or DTs according to the principles.

Considering the *specificity* design principle:

- **AAs**, being by definition *goal-driven* entities, are usually exploited to encapsulate intelligent functionalities that are strictly related to the problem to solve, hence to the domain and objectives of the application.
- **DTs**, instead, are usually exploited with the scope of digitalising assets to provide data-oriented services to consumer applications, *servitisation* being one of their core characteristics.

The principle of specificity, thus, would encourage designers to encapsulate intelligent functionalities closely matching application goals in AAs, and, complemen-

tarily, to model general-purpose services as DTs instead.

Considering the *scoping* design principle:

- **AAs** are not tied to any particular source of information in achieving their goals. Although part of the definition of AA includes a relationship with the notion of an environment in which AAs are *situated* [WJ95], AAs are not limited in any particular way to any given “object” in such an environment. In other words, the application environment (physical or digital) in its entirety is a source of information that AAs can perceive and (possibly) affect—their *global* scope.
- **DTs**, instead, are specifically defined as being *coupled* with the individual asset in the physical world that they are digitally representing (their physical twin). It is thus natural to give DTs a *local* scope, limited to the information available from this physical asset.

Note that the term “scoping” is deliberately chosen instead of, for instance, “locality”, to avoid the misunderstanding that such design principle is inherently tied with some notion of space, such as in a geographical sense or limited by a physical size boundary under which something can be considered local or not.

Scoping simply means that there should be a clear boundary in terms of how many physical assets are involved with the intelligent functionality that designers are considering. If an individual asset is the source of information for that functionality, it could be suited to be encapsulated by a DT. Otherwise, if the information needed belongs to multiple sources, an AA might be more conceptually aligned. However, the boundary of an individual asset can extend to be a rather large entity. In a smart home, for instance, an asset can be an individual device, a room, or the whole house even. Also, nothing prevents an asset from being a composition of multiple others that have a smaller local boundary.

Finally, with respect to the *timing* principle:

- **AAs** do not include any explicit notion of time in their definition. The situatedness feature already mentioned includes the temporal dimension but does not prescribe agents to capture and model time-related aspects. Hence AAs are *time-situated* entities, naturally.

- **DTs**, complementarily, are specifically requested to keep in sync their related PA to provide an updated digital representation in a timely manner. This implies explicitly capturing the time at which events in the PA happened, and modelling the temporal evolution of the PAs. DTs are thus naturally *time-aware*.

Summarizing the above considerations, AAs and DTs can be described as playing the following roles in the design of distributed intelligent functionalities in IoT systems and applications:

- **AAs** are the components of the IoT system that encapsulate the system/application **goals**, hence are aware of such specific goals and strive to achieve them by autonomously deciding **whose other entity** to interact in the whole application domain. AAs are also not particularly tied to any specific notion of time.
- **DTs**, instead, model and encapsulate properties, behaviors, and functions of specific portions of the IoT system or application domain, therefore are **inherently bound to specific assets**, to provide **general-purpose services** to other components or directly to the application. Due to such modelling, DTs must at least account for the specific **notion of time** relevant to the modelled asset.

These definitions are not meant to set crisp boundaries amongst AAs and DTs in any possible use case and scenario. Consequently, there is a degree of flexibility in how these definitions can be interpreted to distribute responsibilities across different components. However, having well-defined criteria that dictate how to exploit AAs and DTs as complementary abstractions helps to identify responsibilities in the design of an IoT system willing to fully leverage distributed intelligence.

9.4 Architectural Integration Patterns

The proposed design principles and their match against AAs and DTs help decide whether an AA or a DT is best suited for a given intelligent functionality. However,

they tell little about how to combine AAs and DTs *synergistically* when intelligent functionality requires it. For instance, because it is positioned within the spectrum of each principle in a way that does not perfectly match all the features of either an AA or a DT, but some of both. Accordingly, this section discusses the several “micro-architectures” for AAs and DTs integration that may arise during the application of our principles.

The term micro-architecture (μ -arch, for short) is used to differentiate from the classic *top-down*, “macro-architecture” perspective adopted in many IoT architectural proposals: what is proposed is a reference architecture for a whole system, a blueprint to adopt in any given IoT scenario (and beyond). In contrast, this work follows a bottom-up approach, where the architecture of the overall system is the result of the composition of multiple μ -arch, depicted in Figure 9.5 by making use of the following components: (i) Physical Assets (“PA” squares), (ii) DTs (“DT” circles); (iii) AAs (“A” circles); (iv) and the intelligent functionalities (the 3D boxes), which may be entire applications or simply one of the many functions needed for the application at hand.

The possible μ -archs are:

- (A) is the most common in the related literature [MPR22, MPR23, RCM⁺22a], and is widely used as a reference architecture for the integration of AAs and DTs. In this pattern, the intelligent functionality to realize does not perfectly match either an AA or a DT. For example, it may have high specificity, require time awareness, and have a scope locally extended to a set of related PAs. Hence, PAs are digitalized by DTs, which offer services to AAs that encapsulate the intelligent functionalities directly serving the applications’ goals. This μ -arch typically arises when an intelligent functionality can be decomposed into an application-specific part and a PA-specific part [MPR22]. The former can combine PA-related information with external data to directly serve application goals and is therefore better encapsulated by an AA. The latter may impose temporal constraints on the PA and be reusable across domains, and is therefore better modeled by a DT.
- (B) represents an edge case in which the intelligent functionality can be directly mapped onto a DT: it has low specificity, its scope is limited to a given PA

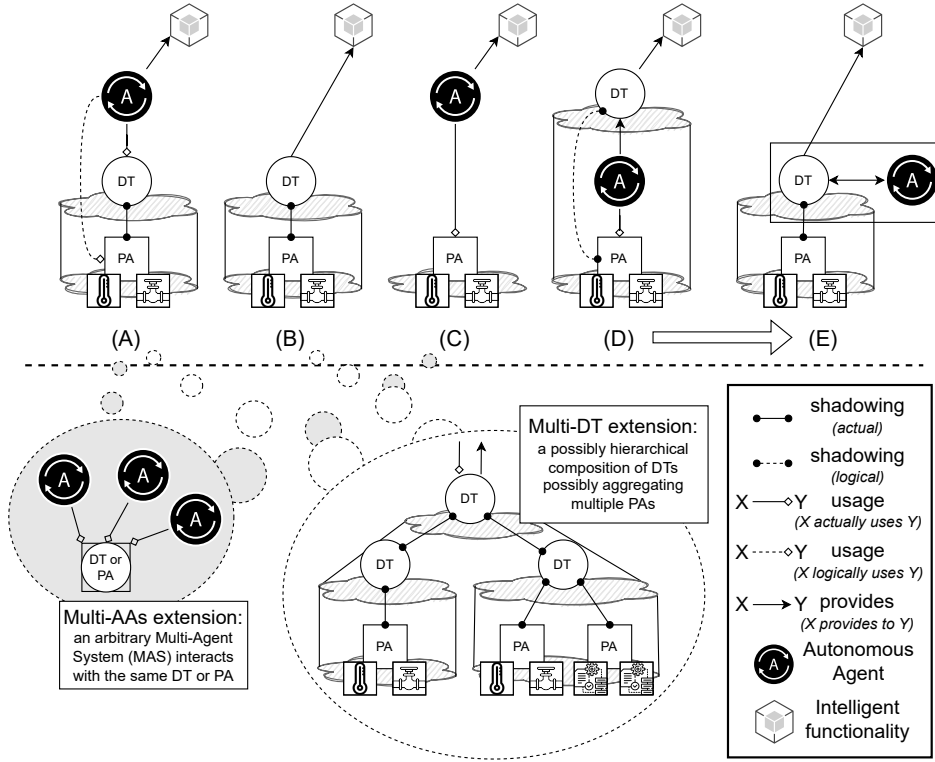


Figure 9.5: Architectural perspective for the synergistic combination of AAs and DTs in distributing intelligence in IoT systems. Each letter denotes a specific *micro-architecture* that may arise from the application of our principles. Such architectures can be combined to give shape to the overall system architecture, in a bottom-up way.

(or a cohesive set thereof), and explicit time awareness is required. In this case, the AA abstraction is unnecessary because DTs are suitable to deliver the functionality alone.

(C) follows the “agentification” paradigm [PH, SGP⁺20]: wrapping any relevant object as an agent to model the domain as a MAS, including physical devices and objects. This is an edge case that requires caution because it can lead to undesired coupling. This pattern is appropriate when the agent’s goal is specific to the asset and the scope of its decision-making is strictly localized. Such an agent should not be intended to provide a digital representation of the asset to other application components (that is the role of a DT), but can

implement, for example, a closed feedback loop on the asset itself. When the functionality's scope expands (e.g., coordination with other agents), μ -arch (A) is more effective in decoupling interaction from asset control via the corresponding DT. With increasing system complexity, this approach often degenerates into either (A) or (E).

(D) is shown as an antipattern and thus is not recommended. This architecture appears, for instance, in literature on agent-based DTs [AGD21, Sta21] and MAS-based DTs [WDD21, PZVN⁺25] when mirroring (or shadowing) of the PA by the DT is performed through an agent. That intermediary role tends to introduce issues, such as delays in the mirroring process [CMS⁺17]. Moreover, digitalizing a PA should not be the responsibility of an AA. A preferable alternative is μ -arch (E), described next.

(E) captures the intent behind (D) more precisely: augmenting or enhancing the capabilities of a DT. In this pattern, the DT remains responsible for digitalizing the PA but interacts internally with an agent to implement or expose augmented capabilities to consumer components. Unlike (A), the agent in this pattern is internal to the DT and is not directly accessible by other system components. This aligns with the DT definition: externally the DT mirrors a PA while adding specific functionalities, and internally it separates concerns between the digitalization process and functionalities better represented by agents (e.g., decision-making, planning, simulation). This pattern goes towards *cognitive* DTs architectures (Section 2.4).

Most of these μ -archs can be trivially extended to multi-AA or multi-DT configurations, where multiple AAs or DTs are used. These cases are exemplified at the bottom of Figure 9.5. Whenever a single AA or DT appears in a μ -arch, that component can be extended into a multi-AA/multi-DT subsystem. For example, the DT of a complex structured PA (a room or an entire building) can be obtained by composing multiple DTs in a hierarchy [JWZ].

It should be noted that these μ -archs can be composed recursively to structure complex IoT systems [WDD21]. For example, a complex DT that digitalizes a hospital ward may be composed of DTs that shadow specific rooms, each in turn

shadowing individual pieces of equipment. Some of these DTs can provide general-purpose services and be assisted by AAs according to μ -arch **(E)**. Those DTs may then be exploited by multiple AAs as in μ -arch **(A)**, each using general-purpose services while providing application-specific functions. By iterating this recursive composition of μ -archs, the overall IoT system architecture emerges in a bottom-up manner, driven by the nature of the intelligent functionalities rather than imposed by a reference architecture, which may fail to capture domain-specific requirements.

The next section illustrates how the proposed design principles and the resulting μ -archs can be applied in a practical case study in the domain of smart manufacturing.

9.5 Exemplary Application of the Principles

In this section, the application of the proposed design principles and the resulting micro- and macro-architectures is exemplified in a smart manufacturing scenario. To illustrate the design process, consider the goal of implementing an IoT system capable of dynamically optimizing production with respect to the overall energy consumption of a manufacturing plant. This goal can be achieved by distributed intelligence systems combining AAs and DTs that implement a set of high-level functionalities such as monitoring, prediction, planning for dynamic task allocation, and adaptation to external factors.

The design process proceeds by first analyzing the functionalities from a domain perspective, decomposing the main goal into finer-grained functionalities that can be implemented directly. Then, the proposed principles are applied to identify the primary characteristics of each functionality and to determine the appropriate combination of AAs and DTs. This analysis provides a practical interpretation of the principles when applied to real-world requirements and validates the role of AAs and DTs as complementary abstractions for designing complex distributed intelligence systems.

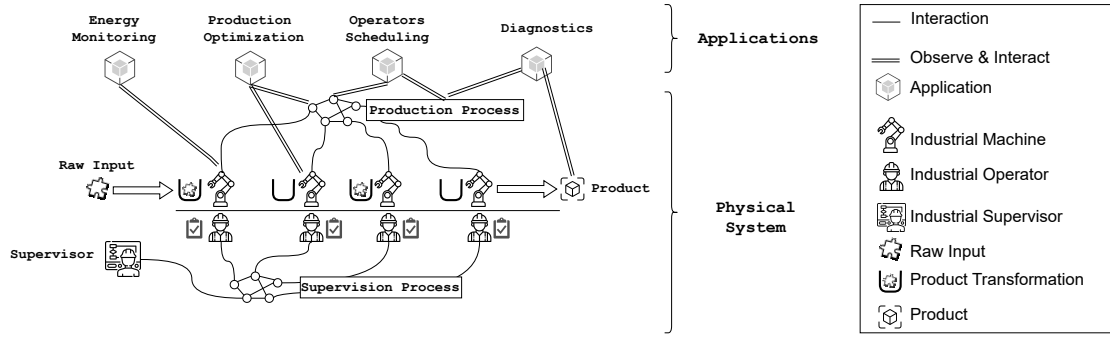


Figure 9.6: A reference Smart Manufacturing scenario with multiple physical entities and a set of applications interested in implementing intelligent functionalities on top of the physical deployment.

9.5.1 A Smart Manufacturing Scenario

A typical production system distributes a diverse array of *machinery*, *operators*, and *processes* throughout the shop-floor environment (as schematically represented in Figure 9.6). These individual components are often organized into *production nodes*, which serve as logical groupings of related machines. Several nodes make up the overall production plant.

Industrial energy monitoring and optimization is a challenging task, especially in a distributed environment. There, such tasks might involve several processes, to selectively turn on different nodes when needed, depending on the external information about the energy cost, while keeping production at an acceptable rate.

As summarized in Figure 9.7, the main target of production and energy optimization is decomposed into three objectives that represent a useful reference to analyse and apply the design principles presented and proposed in this article:

- **Energy Monitoring:** to collect information about the current energy consumption and needs of each production node;
- **Intelligent Coordination:** to dynamically adjust the loads in the production nodes, continuously optimizing the utilisation of resources;
- **Production Optimization:** to make high-level decisions that incorporate data from both individual production nodes and external sources.

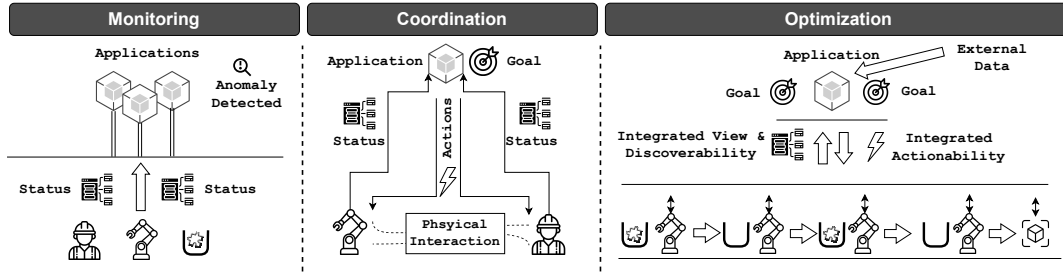


Figure 9.7: Three main objectives that concur to the energy optimization of a Smart Manufacturing plant: Energy Monitoring at the machine level, Intelligent Coordination between robots and operators, and overall Production Optimization based on external factors (e.g. energy cost).

These functionalities are an exemplification of decomposition that can be performed in the design process of an IoT system. In the following, each functionality analyzed towards the definition of components that can implement the IoT system employing either DTs or AAs or a combination of the two.

Energy Monitoring To make dynamic adjustments to the production process based on the energy consumption of the whole plant it is essential to be able to monitor it in real-time through smart metres embedded in the machines. Furthermore, it is essential to collect data from each machine or production node to have fine-grained control on what parts of the plant are consuming the most energy. Monitoring can also include some forms of inference to determine whether the observed consumption is considered regular or is an anomaly, and potentially take corrective actions in such cases to improve safety. Finally, prediction or simulation can be used to forecast what the energy consumption is expected to be, given some production tasks, to improve decision-making.

Intelligent Coordination This objective involves the use of data collected from the IoT system to introduce coordination capabilities to improve both performance and safety at different levels of deployment (e.g., machine-to-machine or machine-to-operator interactions). For example, machine learning models can be trained to detect patterns and, by continuously monitoring and analyzing data streams from machines and operators, the system can automatically understand the context of

the currently performed activities. This is essential to understand the best planning policy to schedule the next tasks as soon as machines or operators are free to perform them. For example, if a portion of the plant is to be shut down for energy saving, the tasks that were queued on those machines should be redistributed across the other active production nodes of the plant.

Production Optimization Digital applications can combine real-time data from the physical world and external sources (e.g., market price and production demand) to dynamically optimize manufacturing processes together with all machines, operators, raw materials, and products. This requires decision-making driven by inference of constraints, as well as means to monitor the current situation to adapt according to the real-time state of the system. For this scenario, it is assumed that the market price of energy is provided by an external forecasting system. According to target business rules, the digital layer can optimize production schedules, resource allocation, and task assignment to maximize efficiency and reduce costs.

9.5.2 System Design with AAs and DTs

Table 9.3 reports the mapping between the intelligence functionalities envisioned, the application of the design principles, and the associated micro-architectural design. Each overall objective is decomposed into specific functionalities and associated with the main *kinds* of intelligent functionalities identified in Section 9.1.2. The analysis leads to the adoption of an AA, a DT, or a combination thereof, for functionality, and a different μ -arch for each objective (as shown in Figure 9.8), choosing from those presented in Section 9.4

Energy Monitoring For this objective, the basic functionality is data collection of a machine’s energy consumption. This functionality has low specificity, since the collected data can be reused for other tasks (e.g., consumption prediction). Because the data are generated by and associated with a specific PA, the functionality has a local (individual) scope. An explicit representation of time is required to produce time series. An anomaly detection functionality analyzing time series is

Overall Objective	Functionalities	Kind	Specif.	Scoping	Timing	Abst.	μ -arch
Energy Monitoring	Data collection	n.a.	General	Local	Explicit	DT	E
	Anomaly Detection	Inference/Prediction	General	Local	Explicit	DT	
	Corrective Reaction	Inference/Planning	Specific	Local	Implicit	Agent	
Intelligent Coordination	Activity Monitoring	Inference/Prediction	General	Local	Explicit	DT	A
	Dynamic Planning	Planning	Specific	Global	Explicit	Agent	
	Task Delegation	n.a.	General	Local	Implicit	DT	
Production Optimization	Decision-making	Inference	Specific	Global	Explicit	Agent	A
	Process Monitoring	Inference	General	Global	Explicit	DT	
	Task Delegation	n.a.	General	Local	Implicit	DT	

Table 9.3: Mapping of design principles to the target use case and broad categories of intelligent functionalities (Specif. = Specificity, Abst. = Abstraction, n.a. = not applicable, pure digitalization function).

also necessary to detect unexpected patterns is expected. The knowledge for this task remains local to the machine, and results can serve general services (e.g., malfunction notifications). An on-the-fly corrective reaction mechanism is introduced to shut down the machine upon detection of a problem. This behavior is specific to the monitoring application goal, operates on local data, and can be triggered by events without requiring an explicit time representation. This analysis indicates the need for a machine DT to collect data and detect anomalies, and an AA to implement the emergency shutdown policy. The resulting micro-architecture corresponds to μ -arch (**E**) in Figure 9.5.

Intelligent Coordination The data required for this functionality is produced by activity monitoring, which has a local scope on the mirrored PA (e.g., an operator) to infer the current activity and to predict its expected duration; therefore, an explicit representation of time is required. Each machine or operator must also expose a digitalized, personalized interface for receiving delegated tasks. Task

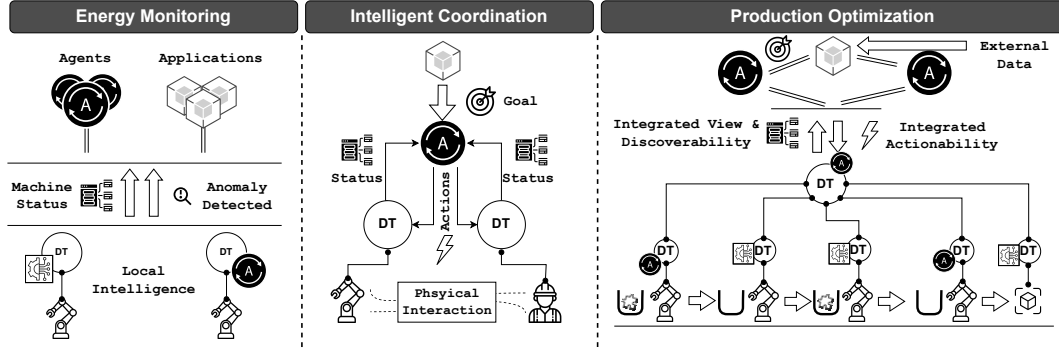


Figure 9.8: Three illustrative instances showcasing the synergistic usage of AAs and DTs as guided by the principles and μ -archs, within the Smart Manufacturing context.

assignment is carried out by a dynamic planning component that operates on aggregated global knowledge obtained from each entity involved in the coordination scenario.

Accordingly, each PA is expected to have a DT that mirrors it and encapsulates activity-monitoring and task-delegation functions tailored to the mirrored entity. Planning can be implemented by an AA that maintains the global context (for example, of an entire production node) and applies a coordination policy to redirect tasks among machines. This corresponds to μ -arch (**A**), where a single agent consumes multiple DTs as data sources and delegates tasks to each.

Production Optimization Optimization at the plant level will be guided by a decision-making functionality based on two sources of data: external data about the market price of electricity for a specific time and date and data collected locally from the different production nodes. Depending on the cost of electricity, for instance, the policy may decide to shut down the nodes that are not actively involved in the ongoing process, delegating tasks to different nodes.

Accordingly, μ -arch (**A**) is the best suited in this scenario, involving a DT that mirrors the production process as the main data source, and an AA enacting the policy and acting on the production nodes. It is worth noting that, having already introduced the DTs of the different production nodes, it is possible to consider linking the DT of the production process to the DTs of the nodes, in a

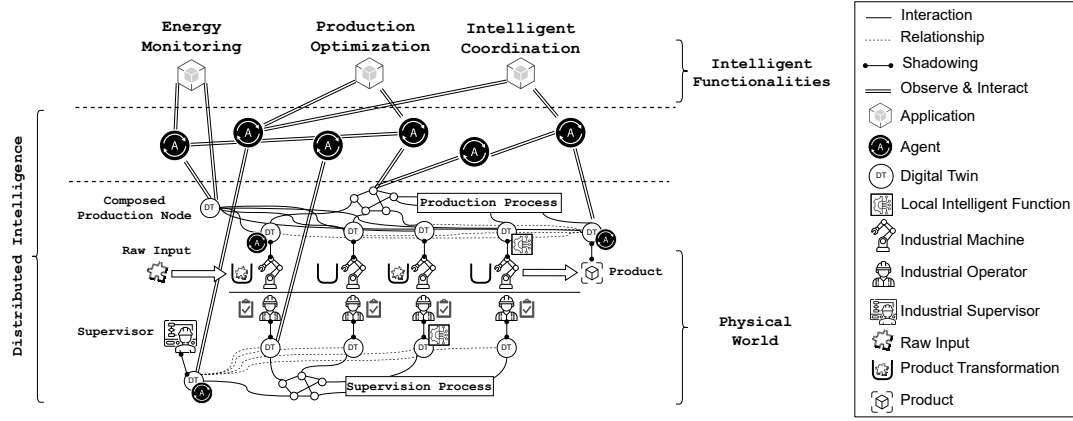


Figure 9.9: A layered architecture with the combination of DTs and Agents for a Smart Manufacturing reference scenario.

composition pattern that would facilitate the modeling of the production process DT as represented in Figure 9.8.

9.5.3 Discussion of Resulting Architecture

Figure 9.9 schematically illustrates the integrated architecture, using both AAs and DTs, resulting from the application of the proposed design principles and the consequential composition of the identified μ -archs.

The physical world, composed of multiple heterogeneous entities such as machines, operators, raw inputs, and products, can be effectively digitalized through the use of DTs. These DTs are responsible for representing the associated PAs in cyberspace through interoperable and homogeneous digital replicas, operating locally to the devices and associated just to their context without the need for a global scope. This approach can also be applied to industrial processes, such as product transformation from raw inputs to final products, the interaction and collaboration between machines and operators, etc.

DTs excel in abstracting and decoupling PAs and processes into modular digital representations, simplifying interaction and integration. They support standardized communication protocols, enhancing interoperability across various platforms and systems, that is critical for integrated manufacturing operations. DTs can be composed of higher-level aggregates, providing a holistic view of the system,

which is crucial for understanding interdependencies and optimizing system-wide processes. Moreover, DTs can embed *localized intelligence*, enabling real-time monitoring, anomaly detection, and predictive maintenance, while reducing latency and improving responsiveness.

On the other hand, AAs are designed to operate autonomously, making decisions based on predefined rules or learned behaviors, which allows them to adapt to changing conditions and respond effectively to real-time events. They excel in tasks requiring dynamic coordination and optimization, such as reallocating resources and adjusting schedules based on real-time data and system conditions. Additionally, AAs can be easily scaled and deployed across different parts of the manufacturing system, providing a flexible solution that can grow with the system's needs. In the envisioned and reference smart manufacturing example, AAs can be distributed across the entire deployment spectrum. They support both local inference and decision-making within a DT, allowing it to achieve its internal target goals, and broader spectrum operations enabling dynamic coordination throughout the production line. This includes adapting planning and task allocation in response to variations in the production line context or external market demand.

AAs and DTs also benefit significantly from each other. AAs benefit from DTs decoupling of complexity when interacting with the physical world and the exploitation of a uniform and interoperable digital representation of it. Conversely, AAs encapsulate decision-making in the system, operating either internally as components of the DT's model (both for single and composed twins), or on a broader scope utilizing the different DTs to collect information and act to improve overall performance and behavior.

An additional benefit of such an integrated approach that combines both DTs and AAs is that the components can evolve separately. Policies can change at the AA level (even online, via learning) without having to modify the DTs. Similarly, new and improved DT models could be produced as data is collected about the system, improving support given to the AA decision-making, without changing the AA at all.

As a final remark, emphasizing modularity, it is worth noting that in this example the decision-making is all performed by AAs. This means that in case of

need (e.g. some critical failure or an expected contingency), it would be possible to temporarily disable the AAs and let the system fall back to a *manual control*. Human operators could amend the system as needed, while still benefiting from the real-time data collected by DTs that can continue to operate.

All of these desirable non-functional properties do not come without costs challenges include system integration, resource scaling, and initial deployment complexity, requiring careful planning and optimization strategies to overcome.

9.6 Final Remarks

This chapter proposes a practical, software-engineering-oriented view on how to combine DTs and MAS to engineer intelligent IoT applications. It introduces three design principles (specificity, scoping, timing) to reason about which intelligent functionalities are better encapsulated by DTs or by AAs, and describes a small catalog of reusable micro-architectures that realize their synergies in practice.

The smart manufacturing example illustrates how the principles drive concrete design choices (e.g., DTs as time-aware, asset-centric providers of data and local services; AAs as goal-driven coordinators for planning and adaptation), and how composed DTs and distributed agents can be combined to achieve modularity, evolvability, and clear separation of concerns. Trade-offs and antipatterns are also identified (e.g., avoiding agent-mediated shadowing of PAs), together with opportunities for runtime evolution (independent improvement of DT models or AA policies) and safe fallbacks to manual control.

RQ4 What are the synergies between DTEs and MAS for the development of intelligent applications? DTs and MAS are complementary: DTs supply a time-aware, modelled, and interoperable digital substrate that decouples applications from physical heterogeneity and provides reusable, asset-scoped services (monitoring, prediction, simulation). AAs (and MAS) supply goal-oriented, autonomous decision-making, coordination, and adaptation that operate over collections of DTs or on system-level objectives. Combining them (using the presented principles and micro-architectures) enables clear responsibility separation. DTs handle data fidelity and temporal modelling, AAs handle planning and policy

enactment while allowing reuse, scalability, and independent evolution of models and policies. This synergy supports robust intelligent behaviors (local autonomy, global optimization, online learning) and practical engineering benefits (modularity, interoperability, graceful degradation).

Chapter 10

Engineering BDI Multi-Agent Systems

This chapter contains contributions from the following works I authored: [Bur23, BBP24, BBC⁺24b, BBP24d, BBC⁺25, CBC⁺25, CAB⁺25, HBRM25b, YBHR23, BYB25]

This chapter presents an overview of contributions in the area of engineering BDI MAS targeting the goal outlined in **RQ5** of making BDI MAS engineering more accessible for developers wishing to integrate BDI agents into larger software systems, there including DTEs and more broadly Web-based systems. Additionally, the chapter explores the role of *explainability* in BDI agents, to make agent behavior more transparent to users, as this is a key requirement for intelligent applications applied to critical domains such as healthcare (Section 1.4).

10.1 BDI Agents in Digital Twin Ecosystems

The previous chapter presented a general alignment between the abstractions of autonomous agents and DTs, highlighting the complementary nature of the two paradigms, and the potential benefits of their integration. This section explores how BDI agents, as a specific subset of autonomous agents, can be relevant in the context of DTs in general, and specifically in relationship with the proposal of this thesis of DTEs and their implementation as hypermedia systems.

The specific application of BDI agents is considered in this thesis alongside three main aspects:

- **Controllability of Autonomous Behavior:** BDI agents belong to the class of *programmed* agents and have been historically tied to the design of programming languages and frameworks, that allow developers to specify and implement agent behavior which could exhibit a degree of autonomy, interpreted as ability to adapt such behavior depending on contextual information. This makes BDI agents particularly suitable for scenarios where developers need to encode business processes and domain logic in a clear and controllable manner.
- **Expressiveness of Cognitive Abstractions:** BDI agents provide high-level cognitive abstractions to encode autonomous behavior. The declarative nature of BDI specifications allows developers to focus on defining system-level goals and strategies to achieve them explicitly, rather than detailing low-level control flows. BDI agents further reason on an explicit semantic model in form of beliefs, that can encode knowledge about a specific domain and encapsulate contextual information relevant to the agent's operation.
- **Explainability of Agent Behavior:** BDI agents offer inherent explainability features, as their internal state and decision-making processes are explicitly represented through beliefs, desires, and intentions which can be inspected and queried. This transparency is crucial in applications where understanding the rationale behind agent actions is important for user trust and system validation.

These features make BDI agents a compelling choice for implementing autonomous components within the context of DTEs. Namely, this thesis studies and contributes to the engineering of BDI MAS envisioning their application in two main scenarios, outlined in the following.

BDI Agents for *Cognitive* Digital Twins: Section 2.4 introduced the concept of *cognitive* DTs as a specific class of DTs that leverage AI techniques to enhance their ability to autonomously and proactively adapt to changes in the

physical counterpart in a goal-driven fashion. This thesis envisions BDI agents as a suitable technology to implement such autonomous components within cognitive DTs, through the mechanism of *augmentation* outlined in Section 5.4. This would effectively implement the micro-architecture **(E)** presented in Section 9.4. In this context, BDI agents can encapsulate the cognitive capabilities of the DT, and be encapsulated as an asynchronous augmentation function within a DT, that can access the DT state, observe its evolution over time, and suggest actions to adapt the DT behavior. To this end, the features of BDI agents outlined above, are particularly relevant: *(i) controllability* is necessary to ensure that the autonomous behavior is aligned with the domain requirements and constraints; *(ii) cognitive abstractions* can be an effective way to interact with the structured and semantic-rich model of the DT digital representation; *(iii) explainability* is crucial to audit and understand the decisions made by the autonomous components within the DT.

BDI Agents as consumers of the Hypermedia Web of Digital Twins: In the original vision of the WoDT, software agents were considered as primary consumers of the WoDT digital representation, leveraging its structured and semantic-rich model to reason about the physical counterpart and make decisions accordingly [RCM⁺22a]. This aligns with the HWoDT proposed in this thesis, that can serve as a shared hypermedia environment for MAS that follow the vision of hMAS introduced in Section 4.4. DTs can serve the purpose of simplifying the interaction of agents with the physical world, and enhance the reasoning capabilities of agents by providing rich models of the PA that can be used also to simulate possible future states hence allowing an agent to perform *what-if* analysis of its actions [Bur23]. Micro-architecture **(A)** presented in Section 9.4 envisions agents interacting with DTs following this pattern. When scaling to ecosystems, the HWoDT provides a set of high-level interaction mechanisms that can be exploited by agents to interact with the whole set of DTs in a uniform manner. BDI agents can be particularly suitable for this scenario to encode business logic that spans multiple DTs, such as coordinating the behavior of different DTs, or implementing complex workflows that involve interactions among multiple DTs. In this context, the features of BDI agents outlined above are again particularly relevant: *(i) controllability* is

necessary as the interactions of multiple DTs must be predictable; *(ii) cognitive abstractions* can help in reasoning about complex interactions among multiple DTs and express clear system-level goals; *(iii) explainability* is crucial to audit and understand the decisions made by agents, especially when they interact with multiple DTs and potentially affect critical operations.

The integration of BDI agents within DTEs with the goals and strategy outlined above is a long-term research direction of this thesis, especially relevant for the domain of healthcare, in which the automation of complex processes involving autonomous agents is seen as a major opportunity to enhance the quality of care and reduce costs [CGMR20]. Accordingly, the rest of this chapter focuses on short-term actions aimed at enhancing the engineering of BDI MAS in general, with actions tailored to achieve a more robust engineering process, as well as facilitate their integration with external systems such as DTEs in the future. The potential integration of BDI agents with DTEs further opens up interesting perspectives on how to better align the knowledge representation of BDI agents with the semantic models used in DTs, how to enhance adaptability in dynamic environments such as the HWoDT, including enhancing BDI agents with future-oriented reasoning capabilities that may leverage the simulation capabilities of DTs, and, finally, how to enhance the explainability of BDI agents so that the autonomous behavior is transparent to both system developers and end-users.

10.2 Tooling for BDI Agent Programming

BDI [RG91] agents are considered an excellent tool for modelling autonomous or intelligent entities via high-level *cognitive* abstractions. Therefore, in the early 2000s, the expectation of the community was for AOP [Sho93] to gain its spot among other prominent paradigms, such as object-oriented (OOP), functional (FP), imperative (IP), and logic (LP) programming.

However, a few decades later, it can be observed that, although BDI has deeply impacted the research field of MAS and AI, it still fails at reaching mainstream programming, even in contexts where the application scenario would make it a good choice to design autonomous behavior.

Limited adoption of BDI in mainstream programming has been widely discussed [Lin00, MWR19, DD10, MF14, Win12], with no consensus on the root causes. Some argue that improved tooling is needed [Hin14], while others suggest that the main barrier is the cost of learning a new paradigm [Log18] without the gain of significant benefits in terms of engineering complex behavior, claiming that new features are needed to make BDI more appealing.

These perspectives are not necessarily conflicting, but rather arguably complementary: paradigm adoption depends on both effective abstractions and supportive tools. In [Log18] C++ is mentioned as an example of language that despite little tooling support at its inception, gained popularity thanks to the introduction of new abstractions (e.g., classes) that made it more appealing to mainstream developers. It could be argued that it was the seamless, gradual integration of such new abstractions into existing Imperative Programming (IP) practices that made C++ successful. As seen in the evolution of C++ and Java, such gradual integration of new abstractions can lower adoption barriers and foster community-driven innovation. Thus, paradigm and tooling evolution should proceed in parallel to facilitate broader adoption, possibly by integrating BDI concepts into existing mainstream programming practices, paradigms, software ecosystems, and tools.

10.2.1 State of the Art

Several open-source, publicly available, and actively maintained¹ frameworks exist for BDI AOP programming. This section highlights representative frameworks, based on recent surveys of logic-based agent-oriented technologies [CCMO21] and agent-based programming for MAS [CF21].

Table 10.1 summarizes key characteristic of selected frameworks. Other notable actively developed frameworks that did not meet the selection criteria but still deserve mention are JACK [Win05] (not open-source), the aforementioned JaCaMo [BBH⁺13] frameworks which extends Jason (Section 4.2), SARL [RGG14] (not BDI-specific), and GOAL [Hin09] (not BDI-adhering).

This short overview highlights how existing frameworks differ in terms of intended target applications, execution platforms and agent programming syntax. A

¹Following the definition in [CF21].

Framework	Platform	Syntax	Intended Target
Astra [CRL15]	JVM	custom DSL	distributed systems
Gwendolen [DF08]	MCAPL [Den18]	AgentSpeak(L)-based	BDI verification
Jadex [PBL05]	JVM	Java + annotations	distributed systems
Jason [BHW07]	JVM	AgentSpeak(L)	BDI research
JS-son [KN19]	JavaScript	JavaScript	Web applications
LightJason [AKM16]	JVM	AgentSpeak(L)-based	scalable applications
Phidias [DLS19]	(Micro)Python	Python DSL	IoT/robotic applications
SPADE-BDI [PRC ⁺ 22]	Python	AgentSpeak(L)	distributed systems
JaKtA [BBCP24]	JVM*	Kotlin DSL	general-purpose

Table 10.1: A selection of active BDI programming frameworks and their respective execution platforms, syntax and intended target. JaKtA, the framework proposed in this section, is included to compare with existing frameworks. *JaKtA is currently implemented on the JVM but could be ported to other platforms thanks to Kotlin’s multiplatform capabilities.

more in-depth analysis of the state-of-the-art frameworks is presented in [BBCP24], focusing on their features and limitations. From this analysis, it is possible to observe that most frameworks are build with custom Domain-Specific Language (DSL) syntaxes that require learning new languages, as well as dedicated supporting tools (e.g., IDE plugins) that require additional effort to set up and maintain. This also hinders integration with existing software ecosystems and tools, as well as reuse of existing libraries and components, as the mechanisms to interface with external code are often limited and not always straightforward to use, requiring developers to shift their mindset away from familiar programming practices.

10.2.2 Desiderata for Mainstream BDI Tooling

As emerging from the previous section, the BDI AOP community has primarily developed toolkits as libraries, extensions, or entirely custom languages supported by existing execution platforms. This approach provides a practical environment for prototyping and deployment, allowing developers to leverage the features of the *host* platform. Hence, a BDI framework typically expects users to work in a top-down manner, approaching system design as a MAS and using BDI abstractions to model agents, while leveraging the host language primarily for implementing vertical, low-level, practical details.

Although this may be effective for BDI experts it may pose a barrier for newcomers. An effective way to learn a new paradigm is to start with a simple yet functional fragment, then incrementally expand its functionalities, gradually introducing new concepts and abstractions (example-based learning, see [vGR10]).

Based on these considerations and the analysis of existing frameworks, the following desiderata for mainstream BDI tooling are identified:

Interoperability with Mainstream Paradigms: to have BDI work as a *practical* paradigm it is necessary to integrate with external systems that typically are not built using BDI concepts, which results in a developer to likely incorporate other paradigms in parts of the developed system. Many modern programming languages (including Kotlin, Python, Scala, etc.) are designed to be *multi-paradigm*, allowing developers to switch among paradigms with relative ease. This approach, known as *paradigm blending*, prioritizes integration of multiple paradigms in one code fragment over composition of multiple fragments using different paradigms. When paradigms are blended, the same code fragment may contain *syntactical* constructs capturing abstractions of diverse paradigms. In this context, BDI AOP could be envisioned as yet another paradigm to support, letting developers free to use a mix of different abstractions in the same codebase.

Code Reuse and Sharing Mechanisms: Code reuse is fundamental in software engineering for reducing redundancy and improving maintainability. General-purpose languages support reuse through constructs such as packages, modules, classes, and functions, complemented by import and extension mechanisms. This is important for both *in-project* reuse, but especially for *cross-project* reuse, enabling the creation and sharing of libraries and frameworks. These are usually supported by build automation and dependency management tools that facilitate integration and versioning. In AOP, similar mechanisms would enable modularization and reuse of agent specifications. For BDI MAS, this could involve sharing plans, beliefs, or rules across agents. However, there is no standardized approach for modularity and reuse at the paradigm level. When a framework provides a Foreign Function Interface (FFI), reuse mechanisms from the host language can be applied to relevant code sections. For example, Jason and SPADE-BDI allow

internal actions to be defined and reused as Java or Python code, respectively. For code written in the BDI language, reuse is often limited to file inclusion (e.g., like in Jason). Some like Astra, support specification extension, while others, such as JACK and Jadex, use capabilities [BPL05] to encapsulate reusable components, though interpretations vary.

Development Tools: Development tools are critical for effective BDI programming. Modern development environments offer features such as syntax highlighting, code completion, static analysis, and refactoring, which are now expected for a swift development experience. Testing libraries and debugging support are also essential for building reliable systems. However, most BDI frameworks lag behind mainstream languages in tool maturity, often due to the high maintenance cost for small communities and limited incentives in tool development in research-driven projects. Developing and maintaining compatibility of plugins for target development environments is challenging. This can be alleviated by developing *internal* DSLs which can directly leverage existing tools, though sometimes at the expense of language expressiveness. Testing and debugging MAS remain complex. BDI frameworks offer limited support, and recent efforts [RTW23, AHK23] are still maturing. Additionally, debugging through host language tools is possible but often exposes low-level details irrelevant to BDI abstractions. Simulation is another important tool, especially for distributed systems. Existing Multi-Agent Based Simulation (MABS) frameworks are not tailored for BDI agents, leading to an abstraction gap and duplicated effort when implementing both production and simulation environments. Ideally, frameworks should facilitate seamless switching between real and simulated deployments as discussed in Section 10.3

Configurability: BDI frameworks also manage the runtime environment for agents, including input/output, communication, and concurrency models (Section 10.3.1). Configurability in these aspects is essential for adapting frameworks to diverse application requirements. The concurrency model, mapping the agent control flow to execution primitives (e.g., threads, processes, event loops), directly impacts efficiency, determinism, and reproducibility and affects system performance. Most frameworks support one or more concurrency models, but few of-

fer user-oriented APIs for easily customizing or selecting the concurrency model. Agent communication is another key aspect, with frameworks differing in their support for local and distributed messaging. Communication typically relies on agent communication languages (ACLs) [KSN00] and underlying network protocols. Pluggable communication mechanisms, using standard protocols, are considered good practice, allowing users to select or extend protocols as needed. However, multi-protocol support and well-documented APIs for communication configuration remain uncommon. Overall, high configurability in concurrency and communication is a desirable property for mainstream BDI tooling, enabling broader applicability and easier integration with existing systems.

10.2.3 The JaKtA Framework

JaKtA is a new BDI programming framework, aimed at (gradually) addressing the issues discussed in the sections above. JaKtA is available on GitHub² under a free, open-source, and permissive licence. The main design principles of the tool are *modularity*, *pluggability* and *paradigm interoperability*.

JaKtA clearly separates: (i) definition of concepts (e.g., agents, environment, plans, actions), (ii) the logic that rules concepts' behavior (e.g., execution loops, selection functions), and (iii) how these are presented to the final user (e.g., syntax, API). Multi-paradigm interoperability is obtained by designing JaKtA as an internal DSL in Kotlin, thus leveraging Kotlin's multi-paradigm features that already support OOP, FP, and IP styles.

The JaKtA framework is composed of four main modules (Figure 10.1), namely:

1. the **BDI interpreter**, which governs the execution of agents in environments, regardless of the particular syntax used to define them;
2. the **DSL**, which provides a Kotlin syntax to idiomatically define BDI MAS;
3. the **concurrency manager**, which regulates runtime, concurrency, and scheduling aspects for any system run by the BDI interpreter; and

²<https://github.com/jakta-bdi/jakta>

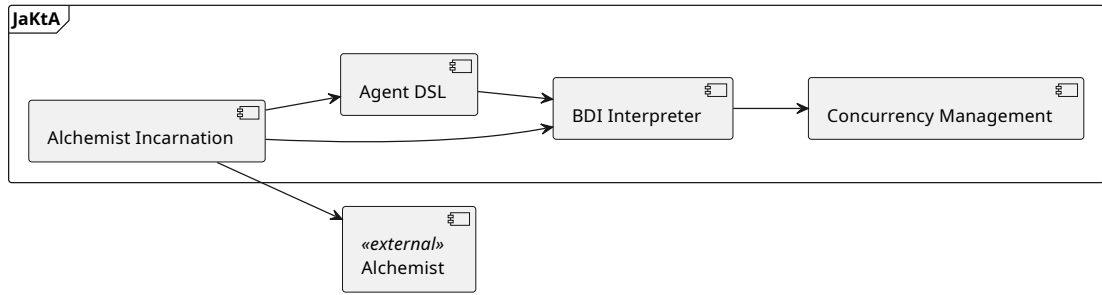


Figure 10.1: The main modules composing the JaKtA framework and their dependencies.

4. the **ALCHEMIST incarnation**, which bridges the BDI interpreter with the **ALCHEMIST** [PMV13] discrete-event simulator.

A detailed description of the JaKtA syntax is available in [BBCP24], for the interested reader. Here some examples of JaKtA code are reported to showcase the main features of the framework.

Listing 10.1: Overall structure of a MAS specification in JaKtA.

```

1 mas {
2   environment { /* ... */ }
3   agent("id1") { /* ... */ }
4   agent("id2") {
5     beliefs { /* ... */ }
6     goals { /* ... */ }
7     plans { /* ... */ }
8     actions { /* ... */ }
9   }
10 }.start()

```

Users can define and launch a JaKtA MAS specification using a `mas` block (Listing 10.1). All elements composing the MAS are defined within it, including the (initial) set of agents, the environment model and other MAS configuration elements. The `agent` block in Listing 10.1 defines an individual BDI agent, specifying its initial beliefs and goals, as well as the plan library it will use at runtime. The syntax is valid Kotlin code, which implements at the language level features to support the construction of DSLs (e.g., lambda with receiver, operator overload-

ing, block-like lambdas, etc.) that can be implemented as *type-safe builders*³. This makes the MAS specification fully supported by existing Kotlin development tools.

Listing 10.2: Example of paradigm-blending in JaKtA. The code snippet uses together OOP, FP, and IP constructs to create a MAS.

```
1  val athletesWebPage = URL("https://olympics.com/en/athletes/")
2  val athleteRegex = Regex("<span class='athlete--name no-trunc'>(.*?)</span>"
3
4  mas {
5      athletesWebPage.readText() // reads the Web page as HTML text
6      .let { athleteNamePattern.findAll(it) } // selects athletes' names spans
7      .take(10) // takes the first 10 athletes
8      .map { it.groupValues[1] } // selects athletes' names
9      .map { it.replace(" ", "-") } // replaces spaces with dashes in names
10     .forEach { agent("athlete-$it") { /* ... */ } } // one agent for each
11     athlete
12 }
```

The adoption of a multi-paradigm language such as Kotlin natively exposes OOP, FP, and IP constructs within the BDI DSL. This supports a first level of paradigm blending, which allows composing and parametrizing the MAS specification with ease. The blending is exemplified in Listing 10.2, it describes, using JaKtA syntax, a toy example defining a MAS with ten agents named as the first ten athletes scraped from a web page. The example showcase the combination of the OOP paradigm employed for dealing with regular expressions matching and data extraction, and the FP paradigm used to map URLs to athletes names, and finally to JaKtA agents.

Further paradigm integration can be leveraged when defining actions. In JaKtA, a valid action is any instance of the `Action` interface (more precisely, of one of its specializations `InternalAction` or `ExternalAction`), and can be written in-place directly inside a `mas` definition, possibly using OOP or FP constructs directly in Kotlin.

JaKtA has an explicit notion of *environment*. The JaKtA environment is responsible for: (i) tracking the agents in the system; (ii) governing each agent's perception by determining which percepts to deliver; (iii) governing each agent's action by making external actions available; (iv) governing communication among

³<https://kotlinlang.org/docs/type-safe-builders.html>

agents by deciding how messages are delivered; (*v*) enabling stigmergic interaction by making the environment's data accessible to agents. The basic implementation of the environment acting as a shared data container can be extended to implement more complex functionalities, integrate external systems within the MAS, and support distributed communication.

10.3 Testing MAS with Simulation

Aside from being useful for in-silico studies, *simulation* may also aid the *development* and *validation* [Uhr02] of MAS intended for real-world deployment. In fact, simulation enables developers to test the behavior of agents and their dynamics in complex environments ahead of deployment, while postponing costs, risks, and efforts associated with real-world execution.

The price for such flexibility, however, is paid in additional development effort: first and foremost, to model the deployment context in a simulation environment, and, additionally, to maintain alignment between the two versions of the MAS codebase, unless the same MAS specification can execute *with no changes* on both real hardware and a simulator of choice.

This is particularly challenging when dealing with BDI MAS. There, the abstraction gap between the high-level cognitive model of BDI agents and most simulators [SPL16], along with the minimal support of BDI technologies for producing *reproducible* simulations of articulated scenarios [Keh16] lead developers to resort to one of the following approaches [SPL16]:

1. extension of the BDI platform with a dedicated simulation engine, which requires additional development effort (e.g., [HB09, RCB⁺20]);
2. construction and maintenance of two parallel codebases one for a simulator extended with BDI modeling tools (e.g., [SKS08, TBC⁺16, US98]) and one for the actual system, leading to consistency issues;
3. integration of the BDI platform with a general-purpose simulation engine, typically by synchronizing their execution through some form of middleware (e.g., [SPL16, DGRM⁺20]).

Despite the idea of using simulation to support MAS development being not new there are not many tools that effectively allow one “*to execute agents as they are and to switch arbitrarily between execution in the real environment and the virtual test environment*” [Uhr02].

This section discusses the challenges of decoupling BDI agent concurrency when designing MAS frameworks, as a necessary step towards enabling the same specification to run both in real concurrent deployments and in discrete-event simulation environments. The proposed integration of JaKtA with the ALCHEMIST simulator explores the mapping of the BDI control loop, showing that different granularities are possible, to address the challenge of providing a seamless transition between real and simulated environments for BDI MAS.

10.3.1 Decoupling Concurrency in BDI MAS

The analysis of a selection of technologies from table Table 10.1 presented in [BBC⁺24b] suggests that most BDI frameworks tightly couple the BDI control loop with the underlying *concurrency model*. This means that the way in which agents are scheduled and executed is often hardcoded into the framework, limiting flexibility and adaptability to different application requirements.

Several concurrency models are possible for BDI MAS:

- *One-Agent-One-Thread (1A1T)*: is the most common where each agent runs in a dedicated thread. This model provides true parallelism but can lead to high resource consumption.
- *All-Agents-One-Thread (AA1T)*: the degenerate case where all agents run in a single thread, leading to sequential execution (e.g. round-robin scheduling), agents do not run in parallel, but the system is lightweight and easier to debug.
- *All-Agents-One-Event-Loop (AA1EL)*: at the conceptual level is similar to AA1T, but instead of using threads, it employs an event loop to manage agent execution. This model is efficient and suitable for handling blocking operations in the agent logic.

- *All-Agents-One-Executor (AA1E)*: similarly to AA1EL but there is a shared executor allowing for concurrent execution of agents while still benefiting from the efficiency of event-driven programming. Scaling the executor allows for tuning the level of resource usage and parallelism.

Decoupling the specification of the adopted concurrency model from the BDI control loop allows to select the most appropriate model for different applications. This requires defining clear interfaces between the BDI interpreter and the concurrency manager when designing BDI frameworks. Each agent control-loop phase (e.g., perception, deliberation, action) should be treated as a task that can be scheduled on a concurrency abstraction. This decoupling can also serve the purpose of switching between real and simulated execution, as the simulator would act as yet another *executor* of BDI tasks.

10.3.2 Discrete-Event Simulation for BDI

Discrete Event Simulation (DES) is a powerful technique for modeling complex systems, that can support the scheduling of events over simulated time, and advance the simulation clock to when the next event is due. Given the fact that BDI agents can be seen as driven by events (e.g., percepts, messages, internal triggers), DES appears as a natural fit for simulating BDI MAS. DES allows for a fine-grained control over the timing and ordering of events, making it possible to accurately model the asynchronous and concurrent nature of agent interactions.

The integration of a BDI MAS over a DES simulator require mapping the BDI control loop iterations onto the simulator's event scheduling. Different granularities of mapping are possible, each with its own trade-offs:

- *Atomic MAS Advancements (AMA)*: the entire MAS advances in lockstep, with all agents performing one iteration of the control loop in a sequential order at every step. This is the coarsest granularity, usually adopted when simulating BDI agents for debugging purposes (e.g., [HB09]).
- *Atomic Control-Loop Iterations (ACLI)*: each full iteration of the agent's control loop is treated as a single event. Interleave of agent actions is possible,

but the agent execution is treated as atomic steps and cannot model the duration of individual phases.

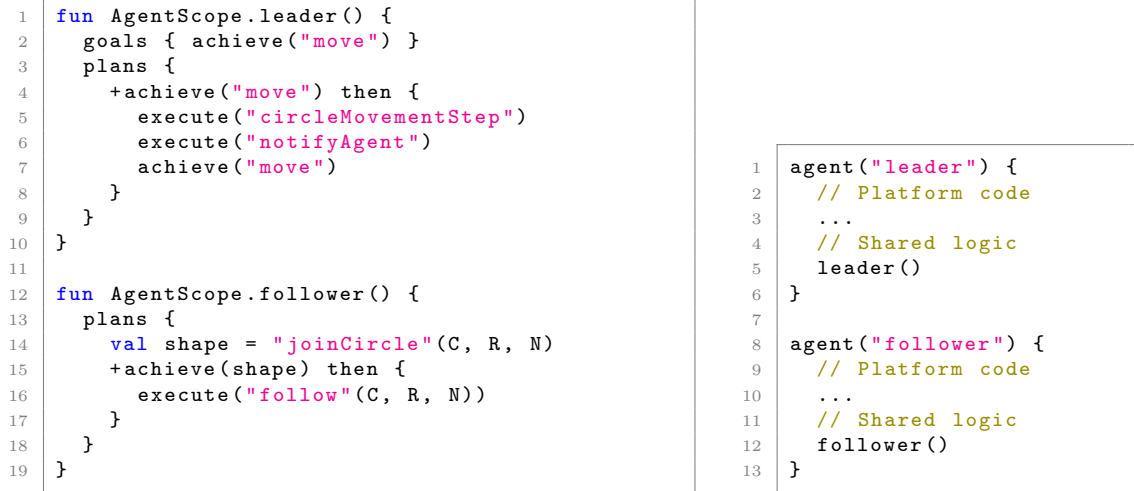
- *Atomic Control-Loop Phase (ACLP)*: each phase of the control loop (perception, deliberation, action) is mapped to separate events. This allows for more detailed modeling of agent behavior and interactions. All possible interleaving are possible, like in a true concurrent system and hence providing the most accurate simulation of real-world execution.
- *Atomic BDI Event (ABE)*: each event in the BDI agent (e.g., perception, message receipt, internal trigger) is mapped to a discrete event in the simulator. This allows for fine-grained simulation but may increase the number of scheduled events significantly, potentially impacting performance of the simulation, while not providing significant benefits over ACLP in most scenarios as the behavior of agents encapsulate its internal events.

Other than choosing the appropriate granularity for the mapping, switching between real and simulated execution requires careful handling of time management and *external actions* that are invoked on the environment.

In real execution, time can be perceived from the system clock, while in simulation, time is controlled by the simulator. Therefore, agents must be designed to use a time abstraction that can be adapted to both real and simulated contexts. The same goes for randomness, which should be abstracted to allow for reproducible simulations controlled by the same *seed*.

For what concerns environment actions, modeling the environment is crucial for accurate simulation. The agent should be decoupled by the specific implementation of the environment, relying on an interface that can be implemented differently for real and simulated contexts. The simulated environment should mimic the behavior of the real environment as closely as possible, including the generation of perceptions, the effects of actions that modify the environment state and the communication mechanisms among agents which may need to be tested for reliability under different conditions.

The design of JaKtA has been influenced by these considerations. This made it possible to integrate JaKtA with the ALCHEMIST DES simulator [PMV13]. The



```

1 fun AgentScope.leader() {
2   goals { achieve("move") }
3   plans {
4     +achieve("move") then {
5       execute("circleMovementStep")
6       execute("notifyAgent")
7       achieve("move")
8     }
9   }
10 }
11
12 fun AgentScope.follower() {
13   plans {
14     val shape = "joinCircle"(C, R, N)
15     +achieve(shape) then {
16       execute("follow"(C, R, N))
17     }
18   }
19 }

```

```

1 agent("leader") {
2   // Platform code
3   ...
4   // Shared logic
5   leader()
6 }
7
8 agent("follower") {
9   // Platform code
10  ...
11  // Shared logic
12  follower()
13 }

```

Figure 10.2: The agent specifications (left) are completely platform-agnostic and reusable, some glue code (right) wires the logics with the underlying execution platform.

integration is achieved by letting ALCHEMIST schedule the execution of JaKtA agents by scheduling their control loop iterations and phases with a time distribution.

10.3.3 Experimental Demonstrator

The prototype implementation demonstrates the feasibility of the approach, and showcases how different granularities of mapping can lead to different simulation behaviors with the same MAS specification.

To exemplify this, a simple MAS is defined to simulate a leader Unmanned Aerial Vehicle (UAV) moving in a circular path, while other UAVs must follow its movement retaining their formation around the leader.

Figure 10.2 shows code extracts from the scenario implementation, demonstrating that agent logic is reusable across different execution platforms.

To investigate the effect of the mapping granularity on the system's behavior, experiments are conducted with the AMA, ACLI, and ACLP granularities. To do so, AMA is used as baseline, letting the entire MAS run a full cycle every simulated second. Then the baseline is compared with ACLI and ACLP, for which each agent

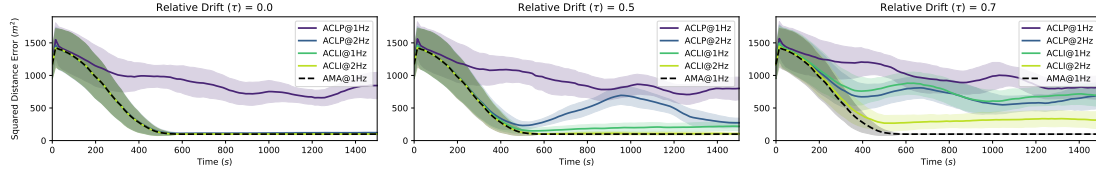


Figure 10.3: Average error with time for different granularities, with ACLI and ACLP running at $f = 1Hz$ and $f = 2Hz$. Different charts show different values of relative drift τ . Coloured shadows represent $\pm 1\sigma$ over multiple runs.

is modeled with an execution frequency f following a Weibull distribution with mean f and deviation $f \cdot \tau$ modeling the *relative drift* in the internal clock of different devices. Additionally, for the ACLP granularity deliberation and action delays are modeled, associating each phase with an exponential distribution with rate $\lambda = f$: faster agents (larger f values) have less delay. Every experiment is repeated 100 times with a different random seed, changing the initial positions of the followers and the distribution in time of the events for the ACLI and ACLP.

Figure 10.3 shows the error on the followers position with respect to the ideal formation over time as the mean square distance of all followers UAVs. As expected given the simplistic implementation of the agent's logic, the AMA granularity appears to be the most stable, converging quickly to a low error. Differently from AMA, both ACLI and ACLP instead, show a degradation in performance at low frequencies and higher relative drifts. This is due to the fact that the implementation assumed implicit synchronization among agents and virtually no-delay in the perception-action loop (which is the case in AMA), but fail to adapt fast enough when the execution is asynchronous and delayed, similarly to real-world deployments.

This showcase demonstrates that selecting fine-grained granularities for mapping the BDI control loop into DES events is necessary to accurately simulate real-world deployments, thus supporting the argument that the BDI framework should expose the ability to finely decouple the execution of individual phases of the control loop from the concurrency model.

10.4 BDI Agents in Hypermedia Environments

As discussed in Section 4.4, the Web is increasingly being seen as a *hypermedia environment* where autonomous agents can operate and interact with resources through REST interactions, and by consuming semantic descriptions of resources and services.

Even more recently, with the advent of LLM-based *Agentic AI* [AKD25] the creation of LLM-driven agents that interact with Web APIs [DWZ24] (or directly with Websites [ZGK⁺24]) has gained significant attention. However, despite being promising, these approaches do not make structured knowledge representation obsolete [PLW⁺24], and often trade the controllability offered by traditional agent-programming paradigms for more guarantees in terms of correctness, and completeness.

Arguably, Web-integrated agents developed with cognitive architectures, such as BDI agents, would retain controllability and predictability: they could directly exploit hypermedia to discover new resources and actions, and use Semantic Web technologies like RDF and OWL to act intelligently without compromising on quality. However, it can be argued that the integration of BDI agents with the Web is hindered by a conceptual and technological gap, which limits developers in creating agents that need to interact with (Semantic) hypermedia. More precisely:

- BDI agents are historically tied to Logic Programming (LP) whereas the Semantic Web is grounded on description logic, and although the two paradigms are theoretically compatible, their practical implementations interoperate poorly, increasing integration cost significantly;
- BDI agents are typically based on the PRS architecture [GL87] which relies on pre-defined plans and does not natively support discovering new actions at runtime, a necessary feature to interact proficiently with hypermedia environments.

In [BBC⁺25], a detailed analysis of these gap is presented, reflecting on the nuances of different models and concrete syntaxes used for logic representation, query semantics and inference mechanisms. The analysis results in the definition of a set of requirements that can guide future research on bridging the gap between

BDI agents and hypermedia environments. For a deeper integration of BDI agents with semantic hypermedia environments, the following requirements should be addressed:

- (R1) direct manipulation of RDF and OWL triples in the agent's belief base;
- (R2) ontological inference for deliberation (e.g., plan selection and execution);
- (R3) support for querying the belief base via SPARQL;
- (R4) ability to exploit affordances discovered in the environment to dynamically adapt the agents' plans.

10.4.1 A Generalized BDI Engine

Towards addressing (R1), (R2), and (R3), [BBC⁺25] proposed the development of a generalized BDI engine that can be specialized to work with different belief, goal, plan representations and reasoning mechanisms. A similar approach has been proposed in [ND06], in which authors discuss the idea of a modular BDI architecture, to make it independent of the logic representation and reasoning mechanisms, although not in the context of hypermedia. Sharing the core modularity idea, the proposal of a generalized BDI interpreter encapsulating agents reasoning cycle, while decoupling the specific technology used for knowledge representation and manipulation would allow for tailoring the reasoning mechanisms to application-specific needs. In this way, beliefs can flexibly support different formats depending on the domain at hand and be adapted to work directly with e.g., RDF triples, to JSON, YAML, and other Web standards.

Similarly, the fundamental operations performed by a BDI agent (e.g., plan selection, belief update, goal adoption), could be implemented using customized mechanisms that leverage the specific semantics of the adopted representation.

Figure 10.4 schematically illustrates the architecture of the proposed BDI engine showing three layers:

- **generalized BDI reasoning layer**, where the agent's deliberation process is implemented as a reasoning cycle, yet agnostic to how beliefs are represented and manipulated. This is where agents' intentions are maintained

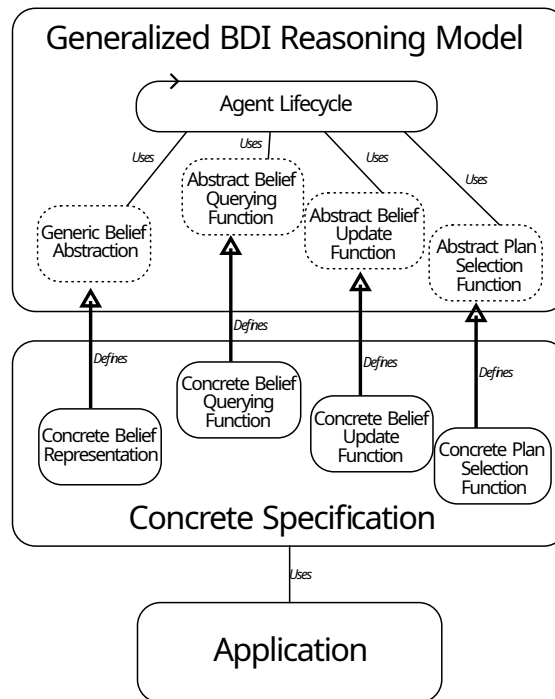


Figure 10.4: The generalized BDI engine architecture. The core BDI reasoning cycle is decoupled from the specific implementations of beliefs, goals, plans, and reasoning mechanisms. This allows for specializing the engine to work with different representations and logics.

and updated as agents perceive, decide what to do, and finally act. Any operation involving beliefs, or plan selection, should rely on some *abstract* API, to be implemented by the next layer.

- **concrete specification layer**, where the aforementioned API is implemented to provide concrete operations on the belief base and plan library. This layer acts as an adapter between the generalized BDI reasoner and the specific knowledge representation technologies of choice. Lastly,
- **application layer**, where actual goals, plans, and interactions of a MAS are defined to tackle some specific use case.

This architectural blueprint is a starting point, to which the JaKtA framework (Section 10.2) is intended to evolve, to better support the integration of BDI agents with other kinds of systems, including hypermedia environments.

10.4.2 Embodiment for Agents on the Web

When considering open, dynamic, and large-scale systems where different agents may join and leave a shared environment at any time, the ability to *discover information about other agents* becomes a crucial factor for enabling interaction. One way to enable the discovery of agents in such open systems is to equip them with *bodies* that are reified in the shared environment. Such bodies may also permit agents to communicate indirectly through reciprocal observation of their actions, facilitate contextual interaction, and support accountability mechanisms through the observability of visible agent behavior—all of which is enabled through embodiment. The body itself becomes an indicator of an agent’s presence, offering means to perceive its (observable) state and actions—and possibly even to ascribe goals to the agent [Cas12].

Software agents are typically defined as *situated* (Chapter 4), but not necessarily *embodied* as the two properties are not dependent on each other when considering virtual environments. On the Web, being *situated* implies agents can browse the Web by discovering and following links from page to page. However, the agents are not *embodied*: Multiple agents browsing the same website are typically unaware of one other, as there is no representation of the other agents currently *on* the same page. This does not preclude *designing* websites that support embodiment—and this can be witnessed regularly, for instance, in collaborative document editors. However, such support must be designed and implemented explicitly into the website.

Following the proposal of hMAS to make all significant entities in a MAS represented within the hypermedia environment [CBR19], it is possible to design an agent body as a Web resource that other agents can discover and interact with. A step in this direction is taken in [ZCF⁺23], which motivates the need for agents’ situatedness and embodiment in open hypermedia environments and illustrates a potential solution based on the Solid specifications [CBLVK21]. The authors outline several requirements for agent embodiment in decentralized hypermedia systems, but they do not provide a conceptual model for agent bodies as first-class abstractions which can guide the design of digital embodiment in hMAS and other kinds of virtual environments.

CARtAgO (Section 4.2) which is an inspiration for hMAS provides an implementation of agent bodies as *artifacts* that are automatically assigned to agents when they join a *workspace* [RPVO09]. The artifact serves as a proxy to the agent in the shared environment, enabling CARtAgO to support heterogeneous agents in the same environment.

Following these ideas, [CBC⁺25] proposes a conceptual notion of *agent body* for hMAS based on the following set of features:

- **Discoverability:** bodies enable agents to discover one another in an environment and optionally identify the corresponding agent associated to it.
- **Communication Through Behavior:** bodies enable implicit communication between agents through their observable behavior, by showing the actions being performed by the agent.
- **Accountability:** bodies support accountability mechanisms by making the agent's actions and state observable.
- **Situation-Dependent Interaction:** bodies facilitate interaction by exposing contextually relevant information about the agent as well as affordances to interact with it.

To support such desired features, the agent body is hence defined as:

A body is an artifact reifying and identifying an agent in an environment. The body situates the agent, transparently mediating its actions and allowing the agent to perceive in a timely fashion. The body allows others to observe the owner agent and its behavior, and to perceive agent-specific affordances to interact with it.

This definition captures the fundamental properties of an agent body in virtual environments, emphasizing its role in reifying the agent (**concreteness**), identifying it (**identifiability**), implicitly mediate actions (**transparency**), routing perceptions from the environment (**timely perception**) and allowing other agents to observe the agent's behavior through it (**focusability**) [CBC⁺25].

Listing 10.1: A description of agent `alice` and its body artifact.

```
1 @base <http://localhost:8080/> .
2 @prefix hmas: <https://purl.org/hmas/> .
3 @prefix td: <https://www.w3.org/2019/wot/td#> .
4 @prefix jacamo: <https://purl.org/hmas/jacamo/> .
5
6 <workspaces/production/artifacts/alice/#artifact> a td:Thing, hmas:Artifact, jacamo:Body;
7   hmas:isContainedIn <workspaces/production/#workspace>;
8   jacamo:isBodyOf <workspaces/production/agents/alice/#agent>;
9   td:hasActionAffordance [ a td:ActionAffordance, jacamo:Focus;
10     td:name "focus";
11     td:hasForm [ htv:methodName "POST";
12       hctl:hasTarget <workspaces/production/artifacts/alice/focus>;
13       hctl:forContentType "application/json";
14       hctl:hasOperationType td:invokeAction ];
15     td:hasInputSchema [ a js:ObjectSchema;
16       js:properties [ a js:StringSchema;
17         js:propertyName "callbackURI"
18       ]
19     ].
20
21 <workspaces/production/artifacts/alice/#agent> a hmas:Agent .
22 <workspaces/production/#workspace> a hmas:Workspace .
```

Listing 10.1 shows an example of implementation of an agent body as an artifact within an hypermedia environment exposed by the Yggdrasil platform [CBR19]. Through RDF triples, the agent `alice` is described as an instance of `hmas:Agent` (with the hMAS ontology [CGL⁺24]), and its body is shown as currently contained in a workspace, situating the agent in the hypermedia environment. The body artifact is also identified as an instance of WoT *Thing* [MKK23], and exposes an action affordance `focus` that other agents can invoke to observe `alice`'s body (e.g., to monitor its behavior).

The definition of agent bodies for hMAS opens up interesting research directions towards better supporting societies of agents operating in hypermedia environments. (R4) assumes particular importance in this context, as agents would not only discover actions exposed by passive *artifacts* in the environment, but possibly also from other agents through their bodies.

10.4.3 Enhancing BDI Agent Adaptability

This section briefly reports contributions in the area of enhancing the adaptability of BDI agents towards addressing (R4). Although not directly focused on hyper-

media environments, the proposed approaches can be extended to such contexts to improve agent adaptability in open and dynamic environments. This follows the ideas presented in Section 5.4 of [BCH⁺23], which discusses the enhancement of agents through a set of reusable strategies that can support their exploration of open environments such as the Web.

Along this line of research, [CAB⁺25] proposes an approach to leverage the generative capabilities of LLMs to generate plans on-the-fly, based on the current context and goals of the agent. Although still in preliminary stages, this approach can significantly enhance the adaptability of BDI agents in dynamic environments, as they can generate new plans when existing ones are not suitable for the current situation. In hypermedia environments, where agents might be programmed to expect specific structures or resources, the ability to generate plans dynamically can help them adapt to unexpected changes. LLM can also be used to interpret the semantics of Web resources and map relevant information to the agent's beliefs and goals, possibly guiding the adaptation process. The exploitation of GenAI techniques for the design of agents is an open research direction, to which BDI agents can contribute by providing a structured framework of cognitive abstractions that support the design and interpretation of autonomous behavior [RMZ⁺24].

A different approach is taken in [HBRM25b], where a proposal for enabling BDI agents to forecast their behavior in a simulated environment before acting in the real deployment context is presented as a way to detect future issues and adapt accordingly by refining ordering of options. [HBRM25a] further explores this idea when considering multiple future-oriented BDI agents, noticing how, without the exchange of information, agents may still end up in conflicting situations as they adapt their behavior based on their individual forecasts which may miss the effects of other agents' actions.

This approach can be particularly useful in hypermedia environments, where the dynamics of the environment and the presence of other agents can lead to unexpected situations. In this context, DTs can actually help agents to simulate their actions in a controlled environment, letting agents play out their plans on a simulation of the PA first to detect potential issues and react accordingly. This opens up interesting future research directions on the interplay of adaptable BDI agents and DTEs [Bur23].

10.5 Explainability in BDI Agents

Explainability emerged in recent years as an important desired property for artificial intelligence (AI) based systems [XUD⁺19]. While the term is more prominently employed to describe the process of extracting human-understandable explanations from Machine Learning models to improve the users' level of trust or allow developers to gain insights into the learned parameters [BADD⁺20], the field of AI is much broader and includes many other kinds of systems that may benefit from the ability to generate such explanations.

Among these AI-based systems, MAS play a prominent role. In this context, explainability mainly concerns the ability to motivate the agents' decision-making process and enable users to validate whether the choices conform to expected behavior.

As discussed in Section 1.4, this is an essential requirement for the development of intelligent applications in safety-critical domains such as healthcare. Accordingly, this section explores how explainability mechanisms can be integrated into BDI agents, to enhance their transparency.

10.5.1 Multi-Level Explainability

High-level cognitive models and architectures such as BDI support explainability *by design* [HvdBM10, BHH⁺10], in principle. This has been clearly recognized in literature by different approaches in the last two decades, including works about exploiting explainability for debugging BDI agents [Hin12, Win17], and validating BDI agent's behavior [WSDD21].

The approach proposed in [YBHR25] follows the intuition of these related works, recognizing that explainability in BDI agents can be addressed at different levels of abstraction, each targeting different stakeholders and use cases. This range from developers working at the implementation level—i.e., using a specific agent programming language—to designers and engineers who may want to abstract from details of the specific technology and focus more on the architectural level, to finally the domain experts and end-users, who focus on the functional and non-functional requirements of the system as a whole, viewing it as a “black-box”.

The proposed *multi-level explainability framework* identifies three levels of explainability for BDI agents:

- **Implementation Level:** explanations at this level target **software engineers and developers**, supporting them in the phases of debugging and testing agent behavior. It focuses on the specific implementation details of the agent and is specific to the agent programming language used (in the prototype, Jason [BHW07])
- **Design Level:** explanations at this level target **designers and architects** as well as developers of the system that want to abstract from the low-level implementation details – concerning specific e.g. agent programming languages or technologies – and focus more on the behavior of the agents at the design level. This level is a good abstraction for designers and architects who want to comprehend the current system’s behavior, interact with other stakeholders and developers or formulate new requirements for improving the system.
- **Domain Level:** the top level, abstracting from how the system has been designed and focusing more on what functionalities the system is meant to provide to stakeholders – i.e. functional and non-functional requirements – dealing with domain-specific knowledge and insights. This level is meant to be useful both for **end-users**, to understand and explain the system’s behavior as a whole while using it, and for **software engineers**, to validate the system given the specification and requirements defined during the analysis stage of the software engineering process.

The proposed framework relies on a process that converts system logs into natural language *narratives* that can be used to derive explanations for BDI agents at the different levels of abstraction, as illustrated in Figure 10.5.

[YBHR25] details how the process can be implemented in practice, identifying set of events for each level, and a mapping function from one layer to the next one. This allows to generate explanations at one level, depending on the events at the lower level, thus enabling a hierarchical approach to explainability.

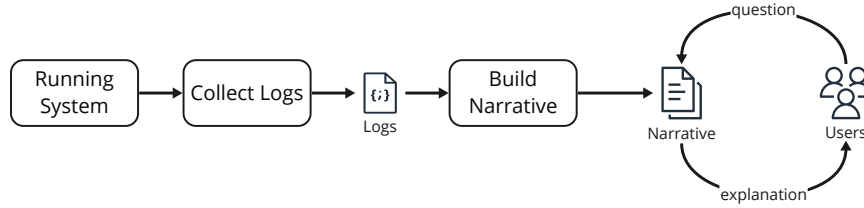


Figure 10.5: The explainability process converting system logs into narratives that can be used to derive explanations for BDI agents.

The resulting process is illustrated in Figure 10.6, showing how explanations at each level are derived from the narratives generated at the lower level, essentially reducing the complexity of the explanation as the level of abstraction increases. This is coherent with the idea that users at different levels of abstraction require different kinds of explanations, that progressively abstract from low-level implementation details. The middle level is implemented using high-level cognitive constructs such as goals, plans and intentions. This ideally would make it possible to use it as a common ground for different BDI implementations, as these constructs are shared across different BDI agent programming languages and frameworks.

The framework has been prototypically implemented and evaluated in the context of a BDI agent developed with Jason [BHW07]. Logs of the agent’s execution are collected by extending the Jason interpreter, and then they can be processed by an external system to generate explanations at the implementation and design level. The domain level remains an open research direction, as it requires the injection of additional knowledge such as functional and non-functional requirements of the system, to be able to generate explanations that are coherent with the domain at hand.

10.5.2 Inter-Agent Explainability

In addition to generating explanations for human stakeholders, BDI agents may also benefit from the ability to explain their behavior to other agents in the system. This is particularly relevant in scenarios where agents need to collaborate or coordinate their actions to achieve common goals, or possibly learn from each other.

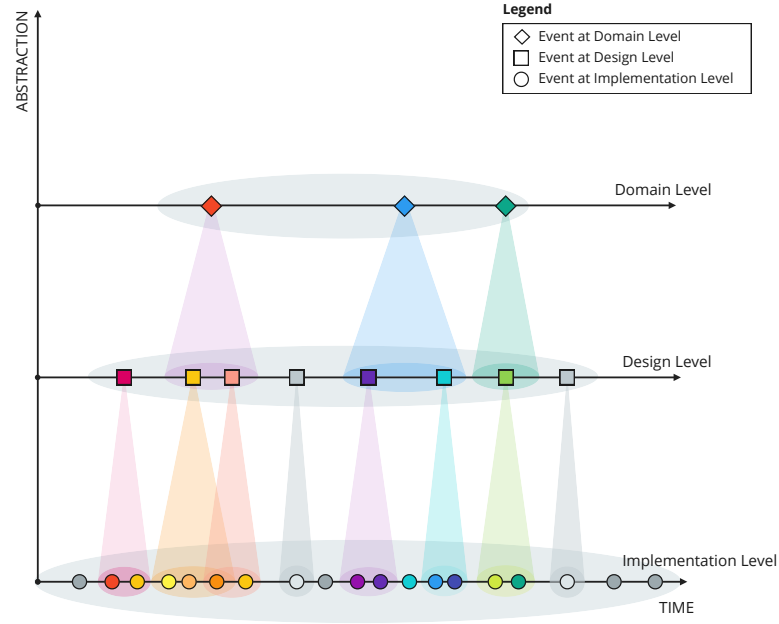


Figure 10.6: Graphical representation of the multi-level explainability framework for BDI agents.

[BYB25] explores this idea by proposing implementation strategies for building blocks that enable the support of inter-agent explainability in BDI agents. The base building block is to extend the agent architecture with a dedicated *explanation store*: a special purpose memory in which agents can store and retrieve *explanatory content* to generate explanations. This is different from the belief base, which is typically used to store *present* knowledge, extending the agent’s memory with a dedicated component for storing *past* knowledge and action traces.

The proposed strategies are the following:

- **Inter-agent explanation protocol:** Define interaction protocols (following the proposal by Ciatto et al. [CMB⁺23]) to enable agents to request and exchange explanations via messages, supporting interoperability and structured dialogue.
- **Implicit and explicit addition mechanisms:** Support both automatic (implicit) and developer-driven (explicit) ways to add *explanatory content* to the explanation store, allowing greater control and flexibility.

- **Runtime state identifiers and inspection:** Provide language-level identifiers and inspection mechanisms for agent runtime state (e.g., plans, intentions), enabling agents to extract and relate relevant information for explanations.
- **Configurable retrieval and generation:** Allow developers to configure how explanatory content is retrieved and explanations are generated (e.g., via rules, ML, or logic), adapting to different application needs.
- **Uniform machine-understandable representations:** Use structured, ontology-based formats for explanations to ensure machine interpretability and interoperability among heterogeneous agents.

Exploring inter-agent explainability opens up interesting research directions for BDI agents, especially in the context of open and dynamic environments such as hypermedia systems, where agents may need to interact with unknown agents and justify their actions to others.

10.6 Final Remarks

This chapter presents different contributions in the area of engineering BDI agents and MAS, focusing on addressing challenges that would make agents easier to develop and integrate with external systems, with a specific focus on hypermedia-based systems, linking to the research activities on hMAS [CBR19] and the HWoDT proposed in Chapter 7. As a complementary perspective to the development of agents, the chapter also explores the role of testing through simulation and explainability mechanisms to enhance the reliability and transparency of BDI agents, towards their adoption in real-world applications such as the automation of healthcare processes. These contributions go towards the long-term vision of combining BDI agents with DTEs to create intelligent, adaptive, and explainable systems that can operate in complex environments, leveraging the rich semantic models and the separation of concerns in the interaction with the physical environment provided by DTs to support the agents' reasoning and decision-making processes.

RQ5 How to make the development of BDI agents more accessible and easier to integrate with external systems such as DT? BDI agents are arguably not as accessible as other programming paradigms. This is not necessarily due to the complexity of the BDI model itself—which can be rather considered intuitive due to the cognitive abstractions it provides—but rather to the gap of current BDI frameworks that originated in the context of AI research rather than mainstream software engineering. BDI agent development can be made more accessible through the conception of new frameworks that better align with modern software engineering practices, *blending* abstractions with other mainstream paradigms and decouple the BDI reasoning cycle both from concurrency models and knowledge representation technologies, letting developers choose the most suitable ones for their application. Improved reliability of BDI agents can be achieved through simulation-based testing. Explainability features can further support the debugging process, and enhance user trust in autonomous systems. Fundamental research on extending the capabilities of BDI agents to adapt to dynamic environments is also necessary to make them more resilient and autonomous in real-world deployment scenarios.

Part IV

Applications Scenarios in Healthcare

Chapter 11

Pharmaceutical Supply Chain

This chapter contains contributions from the following works I authored: [BMG⁺24]

Pharmaceutical supply chain (PSC) concerns the process of sourcing raw materials, manufacturing, distributing, and delivering medications to patients [Sha04]. It is a complex process that includes various stakeholders and requires careful coordination and adherence to regulatory guidelines at every stage to ensure patients receive safe and effective medications. PSC is especially critical in the case of oncological treatments due to the consequences of shortages on treatment quality [NL18, MHDP⁺22], and challenging due to the high degree of therapy personalization [ZGP⁺16] that calls for the on-demand preparation of drugs and accurate timing in the distribution and delivery process. Moreover, patients can receive the treatment only after blood assessments. This can delay the administration and require to dynamically adapt the whole process. PSC management is then very difficult to optimize since it needs to be responsive to sudden change while at the same time efficient in terms of costs and usage of resources.

Generally speaking, the PSC process involves the capability of tracking in real-time different kinds of physical assets (patients, therapy, vehicles, etc.), in different locations and with their own dynamics, enacting a process according to a plan that may need to be promptly revised and adapted depending on events and situations that concern those assets.

In this context, DTEs can be envisioned as a uniform approach to model and manage the ecosystem of heterogeneous physical assets involved in the oncologic

PSC, enabling the development of smart digital systems to help achieve the proper level of responsiveness and adaptability.

11.1 Case Study Analysis

The diffusion of cancers and the consequent expenditure burden of cancer care, requires an effective management of the whole PSC [CCP⁺23]. This work focuses on the downstream of the process, namely, it is not interested in the enterprise production pipeline, but mainly in the in-loco drug preparation and timely administration [PBB20].

For this work, the case study of the Istituto Romagnolo per lo Studio dei Tumori “Dino Amadori”(IRST) has been analyzed. IRST is a research centre located in Emilia-Romagna, Italy involved in the assistance and care for oncologic and onco-haematologic patients. IRST manage the whole PSC for more than 5,000 patients each year and registers more than 49,000 accesses for treatments. Oncologic pharmaceutical laboratories prepare more than 61,700 therapies by using more than 430 active substances. In this context, the process of preparation for cancer treatment begins after the medical prescription and patient’s condition assessment, it involves multiple sub-tasks related to purchasing, stocking, picking, manipulating, scheduling and delivering treatment to the patient (in Hospital or other administration settings). It is a very complex process that should be optimized to reduce waste and guarantee a secure, right, and timeliness drug administration. The main challenges are the following:

- *Optimized warehouse management:* Maintaining stock levels that respect the physical constraints of available spaces and provide precise information on the value (including forecasts) of materials in the warehouse is essential for efficient resource management.
- *Right-time logistics:* Organizing, adapting, and monitoring transport to both input and administration sites must ensure punctuality and precision, guaranteeing safety and quality of care access for patients.
- *Traceability and accountability:* The entire process must ensure traceability

of materials and procedures, as well as accountability for the professionals involved, to enable continuous process review and rapid, accurate reconstruction of any incidents.

- *Anticipation of information:* Digitizing the process ensures information is available at all times, and connecting different phases enables the anticipation of accurate data for planning subsequent steps.
- *Level of automation:* The use of robots for efficient and safe production requires seamless interfacing between automation tools and information systems to support the process.
- *Scenario simulation:* Oncology treatments are frequently updated with new guidelines and administration configurations, and varying costs of active ingredients impact organizational sustainability. The ability to simulate the effects of new therapeutic opportunities is therefore crucial for professionals involved in planning.
- *Clinical trials:* Prescribing, confirming, preparing, administering, and monitoring therapies in clinical trials requires special attention and high levels of safety and traceability to ensure successful research outcomes.

A typical care center is organized as follows:

- *Production Units* are responsible for preparing treatments, generally assembling personalized packages by mixing active ingredients to produce individualized dosages, as well as including ancillary drugs and other medical equipment necessary for the treatment. The process is only partially automatable (using fully automated production lines), since manual operations and checks are required, especially for experimental treatments.
- *Warehouses* store active ingredients, drugs, and other relevant medical equipment to be used in the production units. They may be physically separated from the production units and hierarchically organized to move required stocks from larger centralized warehouses to smaller local ones.

- *Blood test facilities* collect samples from patients and analyze them to verify their suitability for the next iteration of therapy. Sampling and analysis may be carried out in different locations, requiring the transport of samples to the appropriate laboratory.

Patients undergoing therapy typically follow monthly treatment plans, which are prepared and administered after a blood test to help medical staff assess the effectiveness of the therapy and determine suitability for a new round of (either the same or a different) treatment. In the reference context, patients are called into therapy delivery facilities, where blood samples are collected. Samples are then analyzed in a separate laboratory. Production of the treatment begins as soon as results are positive, aiming to deliver the therapy as quickly as possible, so patients can wait at the care center and do not need to be given another appointment.

Given the strict time requirements for the overall process, it is clear that information should be tracked to ensure consistency and availability at all times, enabling an understanding of whether primary resources are available and whether the production process can handle upcoming requests. Moreover, effective tracking of resources, ongoing therapies, and patient conditions could support better planning of appointments, selecting the ideal time for patients to come and be checked to both maximize the probability they can receive treatment and minimize trips to care centers, as well as optimize the overall utilization of center resources such as hospital beds. Not only patients but also stockpile management in warehouses can be improved. This is critical because medical equipment and drugs often have expiration dates and special conservation requirements, such as temperature thresholds that must be maintained even during transport between facilities.

Finally, the scenario described is, of course, an ideal setting. Medical staff may need to adjust or entirely re-evaluate therapy based on patient reactions, impacting the overall process. Additionally, advancements in oncological research can introduce new experimental treatments that must be considered. The system needs not only to track the current situation but also to react to unexpected variables and allow simulation of new scenarios.

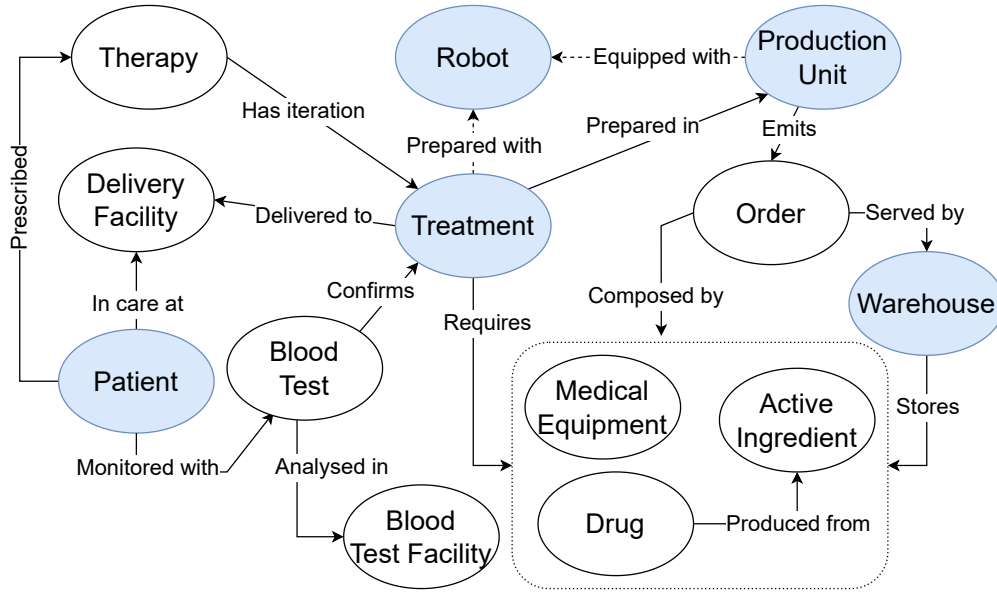


Figure 11.1: Entities in the system and their relationships, blue ellipsis represent the ones selected to be reified as Digital Twins.

11.2 Digital Twin Ecosystem Proposal

Having identified the complex nature of supply chain management for oncologic treatments, this work proposes a model for a DTE to improve the tracking of resources in production and distribution processes.

Figure 11.1 illustrates the Digital Twins that compose the system and the main data each of them tracks to map the supply-chain process. A minimum set of entities is selected to be reified as Digital Twins for system implementation. Patients, treatments, production units, and warehouses are modeled as Digital Twins. Their roles and responsibilities are described in the following paragraphs. Blood test and therapy delivery facilities are considered outside the scope of live tracking and are indirectly monitored through their interaction with the patient. Future extensions of the system could include these facilities as well as the medical staff involved in treatment delivery for a more comprehensive overview.

Patient Digital Twin tracks patient personal data, prescribed therapies, and periodic blood tests to monitor suitability for new treatments. The Patient Digital Twin also records the delivery facility where the patient is receiving care, support-

ing resource utilization monitoring. It can provide prediction services to forecast patient conditions, typically assessed with blood tests, and schedule subsequent treatment iterations accordingly. This service is detailed in the next section.

Treatment Digital Twin exhibits the most dynamic behavior and is used to track the entire process, following its evolution and interactions with other system components. Each Treatment Digital Twin begins in a **scheduled** state for a given date based on the therapy, then transitions to a **confirmed** state if the corresponding blood test is positive. This initiates the production process, with the treatment **assigned** to a production unit depending on local resource availability and current production capacity. Each Treatment Digital Twin stores information about required materials and is linked to the actual resources used in production to ensure batch tracking. The Digital Twin then moves to a **production** state during preparation and, once ready, to a **distribution** state, tracking its transport to the therapy delivery facility where the patient is in care. Finally, it transitions to a **delivered** state once the treatment is administered.

Production Unit Digital Twin monitors and estimates the production capability of a unit, allowing planning algorithms to distribute treatments and optimize selected performance indicators. When treatments are assigned to a production unit, necessary resources must be retrieved from warehouses, prompting the production unit to issue shipment orders. This Digital Twin should provide simulation services to evaluate the impact of new requests or production failures on overall unit efficiency.

Warehouse Digital Twin monitors stockpiles of medical equipment, active ingredients, and ready-to-deliver drugs, tracking stored quantities, expiration dates, and environmental conditions required for special resources. This Digital Twin can issue warnings if stocks fall below set thresholds, expiration dates approach, or required environmental conditions are not met, enabling new orders or resource redistribution to minimize waste.

11.3 Implemented Solution

For this case study, the proposed DTE model has not been fully implemented; however, parts of the system have been prototyped to demonstrate the feasibility of the approach and explore its potential.

To represent the data collected by the system, a KG has been developed to model the various entities and their relationships. Listing 11.1 presents the representation in Turtle syntax for RDF of the data collected by the system. The HL7 FHIR¹ standard is adopted as it provides a comprehensive vocabulary for electronic health records and facilitates interoperability of collected data. Using FHIR resources, therapies for each patient are modeled as **CarePlan** resources and treatments as **MedicationRequest** resources, which can be fulfilled by a **SupplyRequest** handled during production and distribution, and a **MedicationAdministration** to represent where and how the produced treatment is administered during an **Encounter**.

Listing 11.1: Knowledge Graph representing a model of the data collected by the DT system in FHIR RDF.

```

1  @prefix fhir: <http://hl7.org/fhir/> .
2  @base <http://example.com/test/> .
3
4  # a Patient resource with it's ID
5  <patient/1234> a fhir:Patient .
6
7  # the Therapy for a Patient
8  <therapy/1234> a fhir:CarePlan;
9    fhir:status [ fhir:v "active" ] ;
10   fhir:intent [ fhir:v "order" ] ;
11   fhir:subject <patient/1234> ;
12   fhir:addresses ( [ fhir:reference <condition/cd1234> ] ) ;
13   fhir:activity ([
14     # the planned Treatment
15     fhir:plannedActivityReference <request/md1234>;
16     fhir:performedActivity (
17       # and links to record administrations
18       [ fhir:reference <encounter/ec1> ]
19       [ fhir:reference <administration/adm1> ]
20     );
21   ]).
22
```

¹<http://hl7.org/fhir>

```
23  # the Treatment Resource
24  <request/md1234> a fhir:MedicationRequest;
25    fhir:status [ fhir:v "active" ] ;
26    fhir:intent [ fhir:v "order" ] ;
27    fhir:subject <patient/1234> ;
28    # the drug to be administered
29    fhir:medication [ fhir:reference <medication/med345> ];
30    fhir:dosageInstruction (
31      fhir:timing [
32        #repeated once every three months
33        fhir:repeat [
34          fhir:frequency [ fhir:v 1 ] ;
35          fhir:period [ fhir:v 3 ] ;
36          fhir:periodUnit [ fhir:v "mo" ];
37        ];
38        # until the end of the year
39        fhir:bounds [
40          fhir:start [ fhir:v "2023-01-15"];
41          fhir:end [ fhir:v "2023-12-31" ];
42        ];
43      ]).
44
45  # the request for production and distribution
46  # of the treatment
47  <request/sp1234> a fhir:SupplyRequest;
48    fhir:status [fhir:v "active"];
49    fhir:basedOn <request/md1234>.
```

The resulting KG enables accurate recording of all events tracked during patient therapy. By using a standard vocabulary, the data is guaranteed to be interoperable and enriched with useful semantics. This data can then be used to measure and monitor process KPIs, as well as to train models and make predictions.

Each DT can host knowledge-based and data-driven models of the PA. These can be used to simulate scenarios, predict future states, and optimize processes. To experimentally develop a Patient DT, a data-driven model that includes anthropometric, cancer characteristics, blood-test results, treatment history, and recent lab values was trained to predict the patient suitability for the next therapy cycle.

A dataset containing 12,045 patients was used to train the model using a 70/30 split. SVM, Random Forest, and AdaBoost models were trained for a binary task: predict whether the next blood test will be positive. Despite model performance not being sufficient for conclusive results (probably due to dataset imbalance) this prototype served the purpose of demonstrating the feasibility of embedding

predictive models in DTs to enhance the overall system. Refined models could be embedded in the Patient DT to predict treatment readiness in advance, reducing wasted tests and staff time, minimizing patient wait and transport, improving warehouse and drug-preparation efficiency, and enabling effective PSC planning.

11.4 Final Remarks

The analysis and prototype design presented in this chapter demonstrate the potential of DTEs to address scenarios that involve complex processes that span across multiple heterogeneous physical assets. The scenario includes DTs that track people (patients), processes (treatments), and infrastructures (production units and warehouses), each with its own dynamics and interactions. By modeling these entities as DTs, it is possible to create a comprehensive digital representation of the entire pharmaceutical supply chain for oncologic treatments.

The case study is meant to showcase how the healthcare domain may require tackling complex business processes that go beyond the clinical aspects of patient care, involving logistics, resource management and process optimization. In this context, DTEs can provide a uniform approach to model and manage the ecosystem of heterogeneous assets involved. The KG-based model allows to represent the various entities and their relationships with clear, standardized semantics, enabling interoperability and data integration.

Although the full implementation of the proposed DTE model has not been realized, the design has proven feasible and useful for the administration to conceptually understand and analyze the role that digital technologies can play in optimizing the pharmaceutical supply chain for oncologic treatments.

Chapter 12

Trauma Management

This chapter contains contributions from the following works I authored: [GBCR26]

Major Trauma Management is among the most challenging scenarios in the health-care context due to its time-dependent nature. This use case has been used in several works [RCM⁺22a, CGMR20, MCR⁺19] to demonstrate how DTs can support the complex processes involved in trauma care. Timely diagnosis, identification, and response are paramount in ensuring patient health. The entire process has been divided into three main stages:

1. *Emergency Call Management*, where the Centralized Emergency Unit (CEU) receives, plans, and initiates a first-aid emergency mission;
2. *Pre-Hospital Management*, which encompasses the time from the rescuer's initial assessment and treatment to the patient's transfer to the trauma center; and
3. *Trauma Management*, which includes all the processes undertaken by the trauma team in the trauma center to preserve the patient's life.

A recent solution presented in [MCR⁺19] already deeply investigates the latter stage. Hence, this chapter is dedicated to the design and implementation of a DTEs for the *Emergency Call* and the *Pre-Hospital Management* stages of trauma management. This case study is used to demonstrate how the HWoDT framework (Chapter 7) can be employed to integrate heterogeneous DTs from different systems

into a coherent ecosystem, enabling interoperability and advanced functionalities across the entire trauma management process.

12.1 Case Study Analysis

A detailed analysis of the case study is provided in [RCM⁺22a]. Here main aspects are summarized to provide context for the DTEs proposal and introduce the terminology.

The typical scenario begins when a CEU operator receives an emergency call. The operator gathers the *event* first-contact information and initiates a new *mission* for each victim involved. An *ambulance* and a designated *rescuer* are then assigned to each mission to reach the patient and provide first aid. Upon arrival, the emergency crew establishes contact with the *patient*, who may be identified as an authorized *healthcare user* by the health insurance card. According to their conditions, a destination *trauma center* is selected. The patient is continuously monitored during the whole process, especially while establishing a *diagnosis*.

A variety of departments must interact within the context of a complex operational environment, as is the case in the healthcare domain. In order to accomplish their responsibilities, each of them exploits a range of systems that may be developed with different technological stacks due to the fragmented evolution of electronic health systems (e.g., legacy, subcontractors). In the context of this scenario, two principal responsibilities can be identified. The first is the management of emergency calls, which includes the resources planning for missions. The second is the pre-hospital management of the ongoing trauma.

For this hypothetical case study, both systems are assumed to be implemented with a DT-based rationale, using different technologies. Namely, the prototype DTEs is constructed assuming that White Label Digital Twins is used for emergency call management and planning, while Azure Digital Twins is used for pre-hospital management. Finally, Eclipse Ditto has been employed to represent the Healthcare User DT, to simulate an external data source, typically the National Health Service.

12.2 Digital Twin Ecosystem Proposal

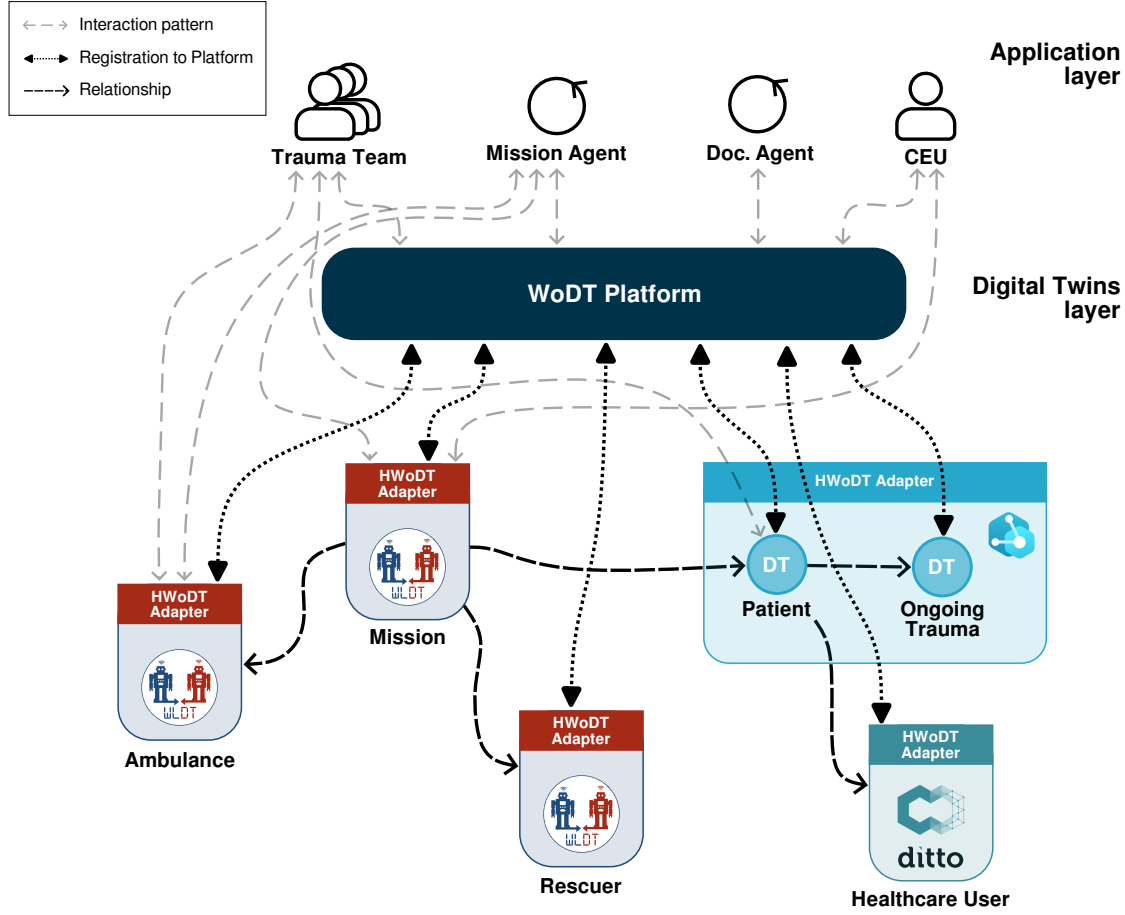


Figure 12.1: The HWoDT-based DTE for the trauma management case study. DTs implemented with different technologies (WLDT, Ditto, Azure) become interoperable through the HWoDT Adapters and are integrated through the WoDT Platform. Black dotted arrows represent the interaction with the platform, black dashed arrows represent relationships between DTs while grey arrows represent the interaction between consumers, the platform and DTs.

Figure Figure 12.1 illustrates the envisioned HWoDT-based DTE for the trauma management use case. DTs of different entities are modeled:

- *Mission* DT, representing the emergency mission, to track the state of the mission as the process goes on, implemented with WLDT;

- *Ambulance* DT, representing the ambulance vehicle, allowing to monitor its location and availability, implemented with WLDT;
- *Rescuer* DT, representing the medical staff member that can be assigned to a mission, again allowing to monitor availability and qualification, implemented with WLDT;
- *Patient* DT, representing the patient being rescued. This DT represents the patient in the hospital system and can be used to collect medical documentation about the patient from as soon as the medical team arrives at site and starts monitoring vital signs, implemented with Azure Digital Twins;
- *Ongoing Trauma* DT, representing the trauma being managed, used to track the state of the process from a medical perspective, and continuing even after the mission is closed at the hospital, implemented with Azure Digital Twins;
- *Healthcare User* DT, representing the patient's healthcare profile in a national external record to access its clinical history, implemented with Eclipse Ditto.

12.3 Implemented Solution

Each DT is mapped to the HWoDT uniform interface through the respective adapter, enabling interoperability within the ecosystem.

The WoDT platform is deployed to provide ecosystem services, including the DTE KG, which integrates data from all DTs using the HL7 FHIR standard to represent healthcare resources.

The trauma management process is supported by the DTE as follows:

1. When the *CEU operator* receives an emergency call, it collects the first-contact information and, according to the rescue process, a *Mission DT* is instantiated for each mission.
2. Upon receiving the URI of a new Mission DT, the *Mission Agent* – a software for supporting the automatic allocation of resources – is responsible for identifying free resources, namely a *rescuer* and an *ambulance*, to assign to the

mission. The *Mission Agent* requires a comprehensive and updated view of the current state of the DTE. In order to identify an inactive ambulance and an available rescuer with the right qualification (Paramedic), the SPARQL query described in Listing 12.1 is executed on the WoDT Platform.

Listing 12.1: SPARQL Query performed by the Mission Agent to obtain the available ambulances and rescuers.

```

1  PREFIX fhir: <http://www.hl7.org/fhir/>
2
3  SELECT ?ambulance ?rescuer
4  WHERE {
5      ?ambulance a fhir:Location ;
6                  fhir:status "inactive" .
7
8      ?rescuer a fhir:Practitioner ;
9               fhir:qualification ?qualification .
10     ?qualification fhir:code ?qualificationObj .
11     ?qualificationObj fhir:coding ?qualificationCoding .
12     ?qualificationCoding fhir:code ?qualificationCode .
13     ?qualificationCode fhir:v "397897005" .
14
15     FILTER NOT EXISTS { ?mission fhir:participant ?rescuer }
16     FILTER NOT EXISTS { ?mission fhir:location ?ambulance }
17 }

```

3. The resources are allocated to the mission. The relationships with the identified *Ambulance* and *Rescuer DTs* are updated by the shadowing process of the involved DT.
4. The *Mission Agent* controls the ambulance onboard system setting the destination to reach. It obtains the Ambulance DTD (Listing 12.1), fetches the interaction affordance of the *SetDestinationCommand* action, and performs a request to it.

Listing 12.1: Portion of the Ambulance DTD that shows the relevant data for the *SetDestinationCommand* action

```

1  {
2      ...
3      "id": <#ambulance-uri>,
4      "@type": [ "fhir:Location" ],
5      ...
6      "wodt:physicalAssetId": "AMBULANCE010203",

```

```

7   ...
8   "properties": {
9     "availableActions": {
10      "@type": "wodt:AvailableActions",
11      "readOnly": true
12    },
13    "position": {
14      "wodt:domainTag": "fhir:position",
15      "wodt:augmentedInteraction": false,
16      "readOnly": true
17    },
18    ...
19  },
20  "actions": {
21    "setDestination": {
22      "wodt:domainTag": "onto:SetDestinationCommand",
23      "wodt:augmentedInteraction": false,
24      "forms": [
25        {
26          "href": "<#ambulance-uri>/action/setDestination"
27        }
28      ]
29    }
30  },
31  ...
32 }

```

5. When the ambulance reaches the patient, a new *Patient DT*, is created to track their health status. A relationship is established between the Mission DT and the Patient DT. The resulting Mission DTKG is reported in Listing 12.2.

Listing 12.2: The Mission DTKG when the ambulance reaches the patient, showing the relationship with the patient.

```

1  PREFIX fhir: <http://www.hl7.org/fhir/>
2  PREFIX onto: <https://patient.ontology#>
3
4  <#mission>
5    a fhir:Encounter;
6    fhir:location <#ambulance>;
7    fhir:participant <#rescuer>;
8    fhir:status "in-progress";
9    onto:patient <#patient> .

```

Meanwhile, the patient's tax code, if available, is collected and the associated

Healthcare User DT is discovered via the Platform DT Discovery Service. This service, starting from a PA identifier, returns the corresponding DTs within the ecosystem. Subsequently, a relationship is established with the Patient DT.

6. The trauma team is informed about the imminent arrival of the patient by receiving the Mission DT URI. In order to prepare to handle the incoming patient, they can observe the incoming patient and the ambulance DTs. The aggregation of DTKGs and DTDs on the WoDT Platform is exploited to obtain both interaction affordances by executing the SPARQL query described in Listing 12.3. The returned affordances are then used to observe the respective DTs.

Listing 12.3: SPARQL Query performed by the trauma team support tools to obtain the ambulance and patient observation endpoints.

```
1 PREFIX fhir: <http://www.hl7.org/fhir/>
2 PREFIX onto: <https://patient.ontology#>
3 PREFIX wot: <https://www.w3.org/2019/wot/td#>
4 PREFIX hypermedia: <https://www.w3.org/2019/wot/hypermedia#>
5
6 SELECT ?ambulanceEndpoint ?patientEndpoint
7 WHERE {
8     <#mission>
9         fhir:location ?ambulance ;
10        onto:patient ?patient .
11
12    ?ambulance wot:hasForm ?ambulanceForm .
13    ?ambulanceForm
14        hypermedia:hasOperationType wot:observeAllProperties ;
15        hypermedia:forSubProtocol "websocket" ;
16        hypermedia:hasTarget ?ambulanceEndpoint .
17
18    ?patient wot:hasForm ?patientForm .
19    ?patientForm
20        hypermedia:hasOperationType wot:observeAllProperties ;
21        hypermedia:forSubProtocol "websocket" ;
22        hypermedia:hasTarget ?patientEndpoint .
23 }
```

7. Finally, the *OnGoingTraumaDT* is created to collect information on the trauma process supporting the trauma management stage.

On top of DTs and WoDT Platform, the business logic and automated control of the scenario are implemented through different applications:

- The aforementioned *Mission Agent*, which observes the DTE KG and manages new missions by assigning resources.
- The *Documentation Agent*, which observes the DTE KG for the automatic generation of the trauma documentation (cfr. TraumaTracker [MCR⁺19]).
- The *CEU Dashboard* periodically monitors the currently active missions by executing the SPARQL query in Listing 12.4.

Listing 12.4: SPARQL Query performed by the CEU Dashboard to obtain the currently active missions.

```
1 PREFIX fhir: <http://www.hl7.org/fhir/>
2
3 SELECT ?mission ?status
4 WHERE {
5     ?mission fhir:status ?status .
6     FILTER (?status="on-hold" || ?status="in-progress")
7 }
```

12.4 Final Remarks

The demonstration of the implementation and exploitation of a HWoDT ecosystem through the use case of trauma management, allows showcasing the potential benefits of the proposed approach from a practical perspective.

As already discussed in Section 7.3, this approach allows to integrate heterogeneous DTs implemented with different technologies under a uniform interface. This further enhances the decoupling of applications from the underlying technologies, further supported by the explicit semantic layer that hides the concrete implementation of the individual DTs. Applications exploiting the DTE can be programmed using Web technologies. This eases the creation of supporting dashboards and automation tools (i.e. the *Mission Agent*) since the DTs are exposed as Web resources. Without the HWoDT this would much harder, as each DT may expose different API and maintaining the functionalities of the application would require dealing with the fragmented evolution of the underlying systems.

As a result, application developers can focus solely on the application logic, abstracting away the complexity of diverse DT technology APIs and SDKs. This approach yields two key advantages:

- a more stable **application layer**, which remains unchanged when new DT technologies are introduced, thereby simplifying the integration of additional organizations and stakeholders; and
- a more stable **DT layer**, which can evolve or replace underlying technologies without impacting the upper layers, thanks to effective information encapsulation and technology hiding.

Additionally, the scenario demonstrates the advantages of tracking relationships between DTs which in this application are crucial to understand the context of each entity involved in the trauma management process and monitor the overall state of ongoing processes effectively. In this scenario, new DTs tracking process state are created dynamically when needed (i.e., Mission DT, Patient DT, OngoingTrauma DT), and relationships are established among them to represent their interactions and dependencies. This would be very hard to achieve without an explicit notion of DTE, and the functionalities the abstraction can serve (e.g., discovery, navigation, querying, observation).

Being based on the notion of a KG, the resulting DTE can benefit from the explicit semantic representation of entities and concepts, allowing powerful and expressive queries to find information on the state of the ecosystem. Of course, this requires an additional effort when developing (or mapping) DTs, as the appropriate domain ontologies need to be used to represent the knowledge provided by DTs. In this example, the use of the HL7 FHIR standard to unify healthcare data coming from different sources greatly improves the quality of the information generated by the DT ecosystem. Hence, this effort is considered a valuable trade-off for the benefits it can bring to the overall ecosystem.

Chapter 13

Operating Room Management

This chapter contains contributions from the following works I authored: [BMC⁺ 23]

Effective management of facilities is a key challenge in the healthcare domain to reduce costs and procedure times and improve the overall quality of services for both the patients and medical staff. In this context, operating rooms are considered one of the most critical and expensive resources in a hospital environment, due to their high operational costs, the complexity and relevance of the surgeries that take place within them.

Surgery schedules and allocation of operating rooms and medical teams are now usually managed through manual processes that rely on the accumulated expertise of the medical staff. Given the ever-increasing digitalization trend in healthcare, the next generation of hospital information systems should allow for better overall management and monitoring of such facilities, to support the medical staff in the planning and allocation of resources.

The typical complex task in the context of Operating Room Management (ORM) is strictly related to the identification of optimized planning of surgery schedule that prioritizes surgeries and maximizes the utilization of resources while minimizing waste of time and poor exploitation of resources. Generally speaking, the system running around the (peri-)operative context can be seen as an ecosystem of surgeons, nurses and anesthesia health care providers, who must smoothly coordinate their work to deliver high-quality surgical care. Moreover, limited physical resources that are exploited in the process—the operating room

itself and its supporting systems, but also specific biomedical equipment—are part of the ecosystem and must be considered when planning the surgery schedule.

The timely execution of the surgical workflow might be influenced by:

- the composition of the team and the interactions between different professional figures;
- the availability of the required equipment and facilities;
- the specific characteristics of the patient and the surgery to be performed;
- the potential for unexpected events or complications during surgery;
- the risk of surgery cancellation.

Planning surgeries means considering all these (and more) factors to provide the best possible service. In the current practice, ORM often relies on the experience and intuition of staff, which can lead to suboptimal performance in operating theaters, as it is challenging to account for all requirements from the various stakeholders involved.

The literature offers a range of methods primarily grounded in mathematical and computational models to support and enhance ORM. These approaches enable more accurate perioperative risk assessments and optimized resource allocation [ZFY⁺19]. A prominent example is the use of Artificial Intelligence, particularly Machine Learning, which leverages large datasets from operating suites to develop predictive models. These models can forecast factors such as surgical case duration, risk of cancellation, and post-anesthesia care unit occupancy, thereby facilitating the creation of safer and more efficient surgery schedules [FSB19, luo20]. Other AI-based strategies include the integration of multi-objective optimization models with metaheuristic techniques, as demonstrated in [ADLM22] for surgery scheduling, and the use of hybrid genetic optimization models, as proposed in [GC17]. Simulation-based approaches are also explored in the literature. For instance, [SKG⁺21] presents a flow-chart-based model of the operating room workflow, which is used to simulate scenarios and evaluate management policies according to indicators such as patient waiting time, idle time, and the number of postponed surgeries.

All these factors make adopting a DT-based approach particularly suiting for ORM. DTs can support the automatic collection of data concerning operating room planning and utilization and construct models that can be used to evaluate new policies, hence supporting the future allocation of resources.

13.1 Case Study Analysis

Managing ORM effectively is a crucial goal for AUSL Romagna. This is a complex task due to the large number of operating rooms (more than 90) distributed across 10 different hospital facilities, each with its own peculiarities in the management of the surgery process. Currently, planning is mainly performed manually with a meeting of the surgical board of each hospital on a weekly basis. Operating rooms slots (typically half-day periods) are assigned to different surgical specialties, which may fit how many surgeries per-slot depending on their expected duration.

Tracking of the actual execution of surgeries, is partially automated and relies on a QR code scanning system, a printed schedule of the planned surgeries for the day and a paper form for each surgery that is filled by hand by the medical staff.

The QR scanning system is designed to allow keeping track of the most relevant phases of a surgical procedure. Additional steps might be possible, depending on the facilities available (e.g. dedicated induction rooms for anesthesia, recovery rooms, etc.), but the main phases that are always present in every surgery are the following:

1. Patient exits the hospital ward
2. Patient enters the operating suite
3. Patient enters the operating room¹
4. Anesthesia is performed²
5. Anesthesia is effective

¹From now on if, for any unexpected reason, the surgical procedure is interrupted step 8 is the next one

²In some cases this step can happen before step 3 if there are dedicated facilities to support it

6. Surgery is started
7. Surgery is completed
8. Patient is moved outside the operating room
9. Patient leaves the operating suite
10. Patient enters a hospital ward (which can be different from the one which he came from)

After a surgical procedure, the operating room is cleaned and prepared for the next one. This process involved the sterilization of the room and reusable equipment, as well as the disposal of any medical waste produced during the surgery.

Timing of each step execution is also annotated on the paper form, one of the main current issues is the discrepancy between the time the QR codes are scanned and the time steps are actually performed, as there might be a delay in reporting or scanning resulting in inaccurate data. This is due to the fact that the medical staff is usually very busy and focused on the patient and can forget to scan at the appropriate time. The paper form allows them to correct the timings annotating afterwards making an estimate of what they remember doing as well as other relevant information about the surgery, which they may find useful to keep track of. This process is also supported by a Surgical-Safety CheckList (SSCL) that ensures that protocols are followed to enhance patient safety and reduce risks of errors during surgery. The checklist covers patient admission, pre-surgery, and post-surgery phases, forcing the medical staff to verify critical information at each step.

Another critical issue with the current state of the system is that data registered by the QR scanning system is stored in a database that is local to each hospital. This, of course, implies that the coordinators that need to manage the network of operating rooms are not able to automatically retrieve information about the current state of the ongoing surgical procedures, leading to less efficient management.

The main pain points that emerged in the conversation with the medical staff are here summarized:

- Low readability of manually annotated data
- Limited space on the paper form to annotate surgery's steps
- Discrepancy between the scanned times and the real times
- Deletion or delay of planned surgery is currently not tracked since the schedule is printed
- Accountability for issues regarding data collection is impossible to define
- Sub-optimal management of operating rooms due to the fact that the availability of a facility is not known in real-time

From the analysis of the current situation and the staff needs the following functional requirements for a new management system are defined:

- Real-time monitoring of the availability of operating rooms: whether they are busy at the moment and what the planned surgeries for each one are
- Time monitoring of surgical procedures and individual procedures' steps to understand the overall efficiency
- Automatic generation of warnings when there is inconsistency in the transmitted data (e.g. two patients in the same room at the same time, a step that is taking an unexpected amount of time)
- Storage of detected warnings for historical data analysis
- Easy data access through a dashboard to visualize the current state, review warnings, and annotate steps. The dashboard should also be accessed through an account to solve the problem of accountability.

From a more technical point of view, the new system is also required to be integrated with the existing solution, and to support interoperability with other future possible systems that may require access to data.

13.2 Digital Twin Ecosystem Proposal

This section describes a DTE designed to collect data and monitor ongoing surgeries in support of ORM. Assuming the adoption of DTs as a pervasive approach, an ecosystem of connected DTs can be envisioned to represent real-world facilities. Each hospital can be represented by a DT, which is connected to the DTs of each operating room. These, in turn, can be linked to the DTs of the medical equipment available in each room. Similarly, individuals, such as patients and medical staff, can be mirrored to track their interactions with facilities and with each other.

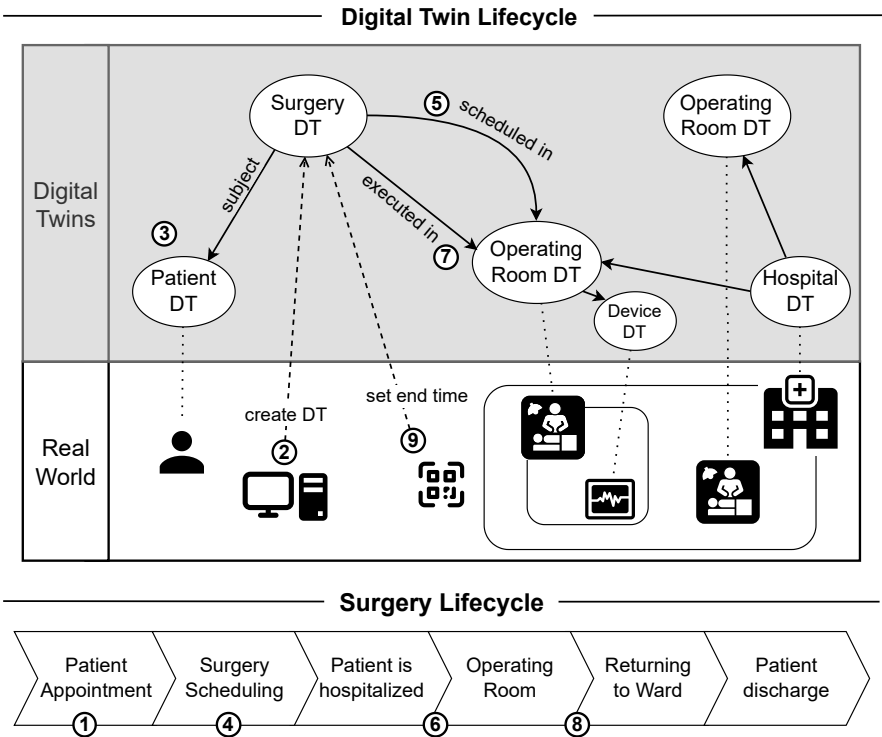


Figure 13.1: An example of how a DT ecosystem could support the management of ORs as well as the real-time tracking of elective surgery processes.

The resulting DTE can represent the state of the operating rooms at any given time, Figure 13.1 shows an overview of the planned DTE architecture, and its relationship with the surgical process. When a doctor and a patient meet for an appointment and the need for a surgical procedure is identified ① a new *Surgery DT* is created ② and linked to the *Patient DT* ③. Then, whenever the surgery

is scheduled in a specific room ④ a relationship with the *Operating Room DT* is added to keep track of the planned surgeries per each room ⑤. The patient is then usually hospitalized and transferred to the operating room ⑥. When the patient actually enters the room the *Surgery DT* creates a new relationship with the *Operating Room DT*³ ⑦ which will report that the room is busy. Finally, when the patient leaves the room ⑧, the end time is registered—e.g. by scanning a barcode like in our case study—⑨ and the operating room can now be set back to an available state after the necessary cleaning operations.

The main responsibilities of the identified DTs in the context of ORM are summarized in Table 13.1.

13.3 Implemented Solution

A prototype DTE has been implemented to demonstrate potential benefits introduced by the proposed approach in the context of ORM. The prototype focuses on the real-time monitoring of ongoing surgeries, and the ability to verify the consistency of data reported by the automatic scanning system, allowing the medical staff to be notified of anomalies and supporting the amendment of incorrect data.

Entities in the domain have been modeled in FHIR for standard healthcare data representation as shown in Figure 13.2. The DT layer is used to translate the existing data sources into FHIR resources, promoting interoperability and standardization. The main resources that the system needs to manage are the various *locations*—namely the hospital structure, the operating suite, the wards and the operating rooms—the *patient* and the *surgery* which has been mapped in three different connected FHIR resources which are

- the *appointment* defining the nature of the planned surgery, its expected duration and its priority
- the *encounter* which is a generic resource for whenever patients and medical staff interact

³For any reason it could also be a different room from the planned one.

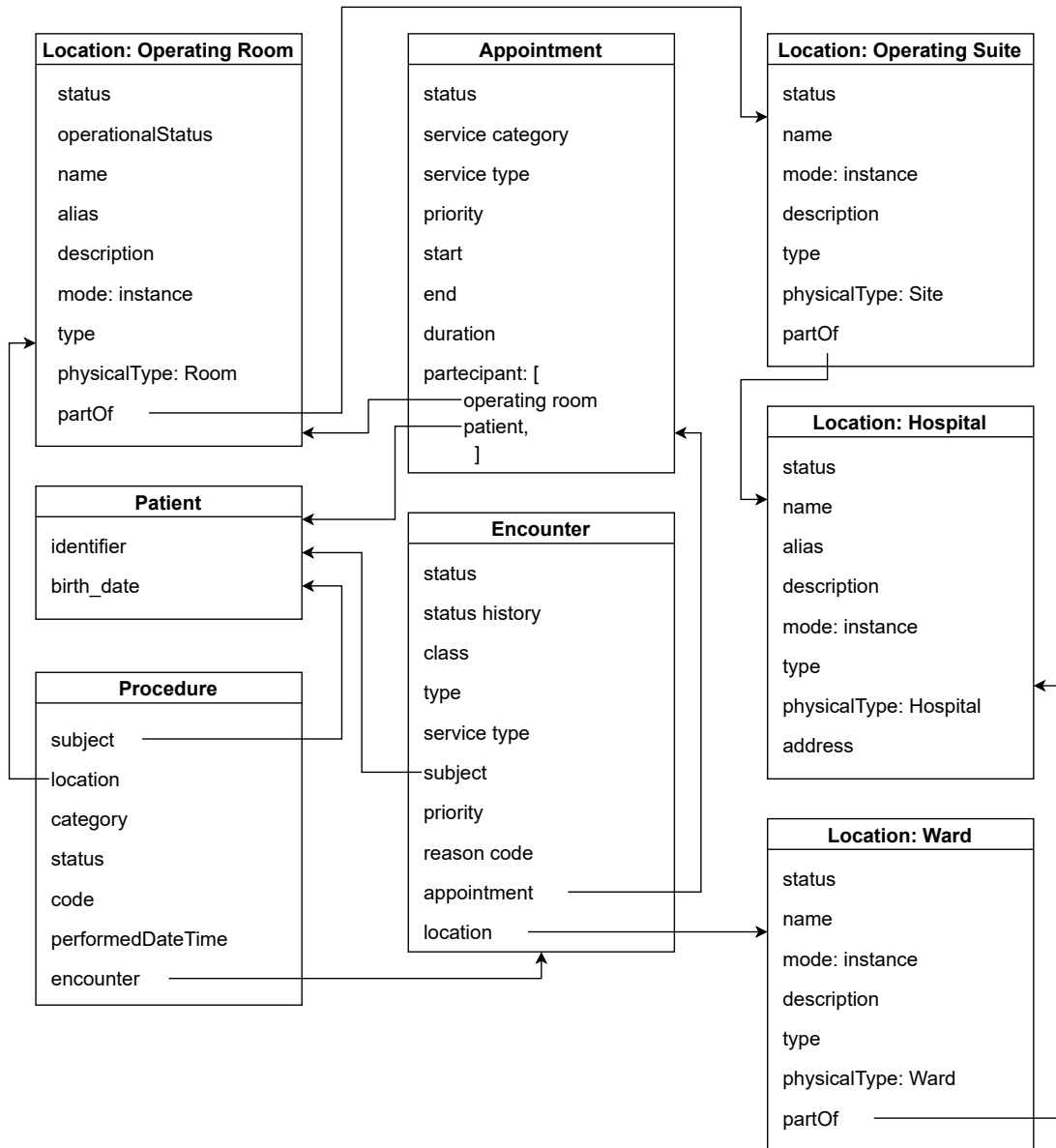


Figure 13.2: The final FHIR resources mapping data related to operating rooms and procedures. Each box represent a FHIR resource, while arrows represent fields holding references between resources.

DT Type	Responsibilities
Patient DT	Stores all patient-related data, including personal and historical data. Mostly static, but forms relationships with other DTs during the surgical process. If indoor localization is possible, tracks the patient's real-time position.
Surgery DT	Tracks different stages of the surgical process and associated measured times. Relationships allow understanding of the surgery in the context of team, patient, and equipment. Before surgery, estimates time and expresses requirements; during surgery, reflects real-time state and emits warnings on unexpected durations; after surgery, serves as a source for planning and patient history.
Operating Room DT	Tracks the state of the room and its equipment, showing real-time and planned availability. Emits warnings if equipment is damaged or cleaning is necessary.
Vital Monitor DT	Medical IoT device recording patient data during (and potentially before/after) surgery. Shares data with the Patient DT for historical records, possibly via dynamic DT composition.
Hospital DT	Aggregates information about operating rooms and provides a general summary of availability. May encapsulate scheduling and planning for associating operating rooms to surgeries.
Medical Staff DT	Represents a staff member, tracking job scheduling, shifts, vacations, roles, and qualifications. Serves as a data source for scheduling.
Stocks DT	Monitors medical devices and consumable resources stocks and can signal or automatically generate supply orders when thresholds are reached. Warns if stocks are expiring to enable relocation.

Table 13.1: Main responsibilities of the identified DTs in the context of ORM.

- and multiple individual *procedures* which represent each action performed on the patient during the overall surgery process and are linked to the same encounter.

The Surgery DT supports a basic anomaly detection algorithm that verifies the consistency of the data being reported by the scanning system and can produce warnings in case data is missing or is not coherent with the expected behavior (e.g., surgery is excessively long compared to the expected duration, surgeries overlap in the same room). The Operating Room DT maintains the state of the room (available or busy) and the list of planned surgeries for that room.

Finally, a prototype management dashboard is implemented as a web appli-

cation using the DTs as the source of data. The dashboard displays the list of all operating rooms, showing their current state and planned surgeries, as shown in Figure 13.3. Additionally, for each surgery it is possible to expand the view and resolve any warning generated by the anomaly detection algorithm, as well as to visualize and annotate each step of the surgical procedure. Annotations and edits are stored alongside which user made them, solving the accountability issue present in the current system.

Cesena	Blocco Operatorio	SALA1	7			5	4	2	2
Presidio	Blocco Operatorio	Sala	Intervento in Nota	U.O. di Riferimento	Codice Intervento	Pianificato	Uscita da Reparto	Dentro al Blocco	Induzione in Corso
Cesena	Blocco Operatorio	Sala1		Oculistica	1832c6b3841-90c253e9-b59b-48e1-8f2c-3f8c8f687df9		11:00 	11:25	11:35
Cesena	Blocco Operatorio	Sala1		Cardiologia	1832c6b3841-90c253e9-b59b-48e1-8f2c-3f8c8f687df0		12:03 	12:14 	
Cesena	Blocco Operatorio	SALA2	8			8	7	6	6
Cesena	Totale Blocco NCH		8			6	6	6	5
Cesena	Blocco Operatorio	SALA3	6			4	4	3	2
Cesena	Blocco Operatorio	SALA2	4			15	2	9	78

Figure 13.3: The expanded view of an operating room, showing the surgical procedures of that room and their steps.

13.4 Final Remarks

This preliminary implementation demonstrates how a DTE can be used to support the design of quality-of-life tools and improve the management of complex healthcare processes such as ORM. The current implementation is simplistic and limited in scope, but it provides a demonstrator of applicability of the DT paradigm to the domain of ORM, showcasing the potential benefits of the proposed approach including data standardization, legacy system integration, real-time monitoring, anomaly detection, and accountability.

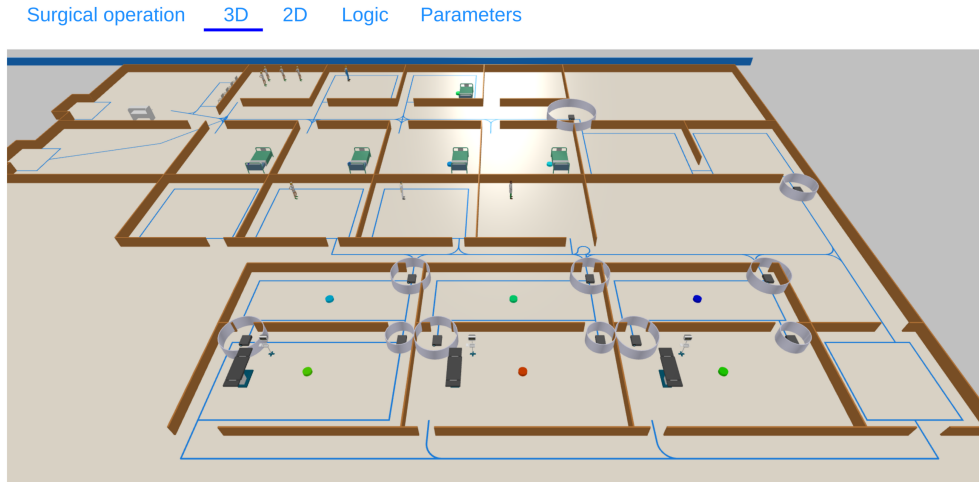


Figure 13.4: Snapshot of the 3D view of the AnyLogic simulation of surgical processes used to generate synthetic data for the DTE prototype.

Simulation of the surgical process has been explored to test the proposed DTE in a controlled environment. Specifically, as shown in Figure 13.4, an agent-based simulation developed in AnyLogic⁴ has been used to model the patient flow through the operating rooms and generate synthetic data that can feed a prototype implementation of the DTE.

The simulation models the process assuming a hospital with 3 operating rooms, and a variable number of medical professionals currently on shift. Patients are scheduled for surgery according to a predefined plan, but random events can occur that may lead to delays in the process. The simulation allows for key event in the patient flow to be tracked, generating traces of events that can be used to communicate with the relevant DTs in the DTE prototype.

Although in this preliminary implementation the focus has been on using the simulation as a tool to support development of the DTE prototype, in future works the simulation model could be integrated into the DTE itself to enhance the capabilities of the system in terms of predictive analytics and scenario evaluation.

⁴<https://www.anylogic.com/>

Conclusion

Motivated by the digital transformation needs of healthcare, this thesis presents a collection of contributions following the hypothesis that

A Digital Twin Ecosystem, integrating heterogeneous Digital Twins (DTs) through semantic and interoperable representations, can provide a unified, always up-to-date view of Physical Assets (PAs), thereby enabling the development of intelligent, adaptive applications capable of supporting and partially automating processes in complex domains, such as healthcare.

The following sections summarize the main contributions of the thesis, discuss the findings in light of the central hypothesis and outline potential directions for future works.

Summary of Contributions

After an analysis of the state of the art (Part I) in research areas connected to the topics of healthcare digitalization (Chapter 1), Digital Twins (Chapter 2), Web and Semantic Web technologies (Chapter 3) and Multi-Agent Systems (MAS) (Chapter 4), the thesis provides answers to the research questions presented in the introduction through the following contributions, each supporting different aspects of the central hypothesis:

Chapter 5 A reference architecture, architectural patterns, and a supporting implementation framework for the engineering of DTs with a focus on

Internet of Things (IoT) integration, heterogeneity, lifecycle management and design of augmented behavior (**RQ1**). These contributions support the hypothesis by demonstrating that individual DTs can be engineered to integrate heterogeneous data and expose enhanced capabilities, forming the building blocks of a unified ecosystem.

Chapter 6 A conceptualization of Digital Twin Ecosystems (DTEs) as an abstraction to engineer systems based on DTs capable of offering a unified view of several physical entities, along with an analysis of functionalities and possible architectural patterns to support their design and their trade-offs (**RQ2**). This conceptualization directly addresses the hypothesis by showing how multiple DTs can be integrated into a coherent ecosystem that provides a unified, up-to-date view of PAs.

Chapter 7 A proposal, and a supporting prototype implementation for the concrete realization of DTEs based on existing DTs applying principles from the Web architecture and re-using technologies and Semantic Web standards to address the challenges of interoperability, data integration and knowledge representation (**RQ2**). By implementing a prototype framework based on semantic and interoperable representations, this contribution validates a key aspect of the hypothesis: enabling multiple DTs to provide a unified, semantic, and machine-understandable ecosystem-level view.

Chapter 8 A proposal and implementation strategies for the Digital Twin Continuum (DTC): a middleware to support the deployment of DTEs in distributed computing environments, while supporting the interaction of different stakeholders with the DTEs through different phases of the development-to-deployment lifecycle of DTs (**RQ3**). This contribution supports the hypothesis by providing practical mechanisms to deploy and manage DTEs in real-world distributed infrastructures, a necessary step to bridge the operational gap for managing DTEs.

Chapter 9 An analysis of the complementary nature of DTs and MAS as abstractions to encapsulate intelligent applications, with a proposal of design

principles and micro-architectural patterns that can guide the implementation of intelligent IoT systems through combinations of DTs and autonomous agents (**RQ4**). This contribution supports the hypothesis by demonstrating how MAS can leverage DTs to implement intelligent, adaptive behaviors over complex cyber-physical domains, identifying the benefits of combining these two abstractions.

Chapter 10 A collection of contributions in the area of MAS targeting the re-design of supporting tools to lower the entry barrier, favor the use of simulation for stronger engineering guarantees of the developed MAS behavior, support a more direct integration with open environments such as the Web, and support for explainability features that are essential for real-world deployments and user trust (**RQ5**). These contributions provide foundational elements to support the integration of Belief-Desire-Intention (BDI) agents within DTEs, facilitating the development of intelligent, adaptive, and explainable applications that can support the hypothesis of automating complex processes through the exploitation of DTEs.

Finally, Part IV presents the application of the proposed contributions in three healthcare-related scenarios developed thanks to the partnership with AUSL Romagna: the management of the oncologic pharmaceutical supply chain (Chapter 11), the management of rescue operations for trauma patients (Chapter 12), and the management of operating rooms (Chapter 13). These applications demonstrate the feasibility and potential value of the proposed contributions for addressing real-world problems, offering initial conceptual and prototype-level validation for the hypothesis in the healthcare domain.

Discussion

The work presented in this thesis validates the hypothesis that DTEs offer a coherent framework to integrate heterogeneous DTs and support intelligent applications. Beyond validating the conceptual feasibility through prototypes and case-study

analyses, the results highlight several critical insights about the practical realization of this vision.

First, while individual DTs can effectively expose data and augmented functionalities of PAs, the thesis also reveals the complexity of orchestrating multiple DTs into a unified ecosystem. To this end, the proposed modular architecture supports interoperability across DTs, but the exposure of DT-specific functionalities such as predictions or simulations is currently under-explored, especially in terms of how to coordinate and compose such functionalities across multiple DTs to support ecosystem-level analysis and decision-making. The thesis explores this aspect through a preliminary analysis of interaction patterns to access the (current) DT state and invoke actions, while other patterns may be needed to support a broader range of DT functionalities.

Second, semantic interoperability provides a powerful mechanism for unifying heterogeneous DTs, yet it introduces its own complexities. Ensuring meaningful alignment between different vocabularies and evolving ontologies, while avoiding inconsistencies, demands both technical solutions and careful governance practices. The work focuses on a practical demonstration of these principles, but does not resolve these challenges fully as they are inherently complex in real-world settings due to factors that go beyond the technical domain, such as organizational and social factors that influence the adoption of shared semantics.

Third, despite the thesis advocating for DTEs as the foundation for intelligent applications, it is currently limited in its exploration of different kinds of artificial intelligence techniques that could be integrated within DTEs. Namely, this work focuses on the integration of MAS in general, and specifically in the connotation of BDI agents to automate decision-making processes within the DTEs, contributing to the alignment of DTs and MAS as complementary abstractions, and proposing design principles and architectural patterns. The integration of MAS with DTs lacks validation in the form of concrete implementations in the application scenarios due to the narrow scope of the case studies which focused on enabling a digitalization layer based on DTEs without fully exploiting it to automate processes through the integration of autonomous agents. Nevertheless, the proposed approach conceptually demonstrates that, through the interaction with DTEs, agents can exploit unified views to implement adaptive behaviors. Fully leverag-

ing predictions, what-if analyses, when implementing cognitive decision-making requires richer twin models and deeper exploration of the integration between agents and DTs. Towards this end, the thesis focuses on supporting the engineering of BDI agents with new tools, support for simulation and explainability features, as well as increased flexibility to integrate with external systems and open environments. These contributions act as a foundation to engineer real implementations of intelligent applications based on DTEs. Human factors, usability, and trust also emerge as important considerations when intelligent actions affect real-world processes.

Finally, real-world application exposes additional constraints around performance, scalability, and security. Network delays, heterogeneous hardware, and privacy requirements introduce variability that must be addressed for operational deployment. The prototype implementations confirm feasibility in controlled settings but highlight the need for systematic evaluation under realistic conditions.

Taken together, these observations suggest that the core idea of DTEs is supported at a conceptual and prototype level, but its full potential depends on addressing orchestration, semantic governance, and operational scalability. These critical aspects provide the roadmap for future works to move from proof-of-concept to practical, real-world deployments of DTEs that can truly support intelligent applications in complex domains such as healthcare.

Future Works

While this thesis establishes the conceptual foundations, architectural principles, and initial prototype implementations of DTEs, it also exposes a set of open challenges that must be addressed to move from proof-of-concept systems to mature, large-scale deployments. This section outlines concrete directions for future research and development that emerge from the trade-offs observed in the proposed architectures, implementations, and application scenarios. These directions identify both technical and socio-technical challenges that represent necessary next steps for realizing the full potential of DTEs as enablers of intelligent, adaptive applications in complex domains such as healthcare.

From a maturity perspective, the contributions of this thesis span conceptual,

architectural, and prototype levels. Advancing towards fully operational and autonomous DTEs requires addressing the challenges outlined in this section, particularly with respect to orchestration, governance, scalability, and trust.

Advanced Functionalities in Digital Twins

The contributions in Chapter 5 support the engineering of DTs that can integrate heterogeneous data sources, manage their lifecycle and expose digital services that augment the capabilities of the physical entities they represent.

The lifecycle management features provide a structured approach to handle the synchronization between DTs and their corresponding PAs. Measuring and assessing the quality of this synchronization remains an open challenge, as it depends on several factors such as the nature of the PA and the specific application scenario for which the DT is used. These requirements could also vary over time, depending on the lifecycle of the DT.

Future works could continue exploring strategies to monitor and adapt the DT lifecycle to detect and react to potential issues in the synchronization with the PA, possibly leveraging prediction capabilities to compensate for temporary disconnections or data unavailability. These features should be designed in order to possibly guide practitioners in selecting the appropriate synchronization strategies based on the specific requirements of their DTs, but also DT consumers in understanding the guarantees and *fidelity* of the DT representation.

The work on augmented behavior provides initial patterns to design DTs that can encapsulate intelligent behavior. This has currently been tested in controlled lab environments, with simple DT behavior. Embedding more advanced forms of augmentation in DTs, possibly integrating prediction and simulation capabilities remains unexplored, and although the model proposed should be general enough to support these features, further validation is needed to assess its effectiveness in supporting more complex DT behaviors.

Finally, the exploitation of such advanced DT functionalities at an ecosystem scale through DTEs remains an open challenge. Especially interesting in this setting is the idea of composing simulations or chaining predictions from multiple DTs to support a forward-looking analysis of the behavior of the ecosystem, based

on its current state and graph of relationships among DTs. This would require further investigation on how to orchestrate and coordinate the execution of such advanced functionalities across multiple DTs in a coherent manner.

Semantic Interoperability in Digital Twin Ecosystems

The findings highlight the feasibility of re-using Web and Semantic Web technologies to address interoperability challenges in DTEs, following a similar rationale as the ones adopted by the Web of Things (WoT) and Hypermedia Multi-Agent System (hMAS) initiatives. In this sense, the proposal in Chapter 7 of the *Hypermedia Web of Digital Twins (HWoDT)* aims to provide a practical mapping of the Web of Digital Twins (WoDT) vision onto a concrete architecture and implementation based on existing standards from the Web and Semantic Web domains.

In that context, the explicit semantic representation in the HWoDT is both a tool to support easy integration of DTs and a deliberate choice to push shared semantics so that users can get more information on DT functionalities. The HWoDT does not enforce the adoption of a single ontology: while DTs expose a uniform Digital Twin Description (DTD), their Digital Twin Knowledge Graph (DTKG) content may rely on different vocabularies from diverse application domains.

Semantic consistency is thus delegated either to DT designers – who may agree on shared representations – or to different implementations of the WoDT platform, which may translate and align models to a common vocabulary. If semantic consistency were a requirement of a specific DT ecosystem, the platform could implement validation of DT representation through existing Semantic Web mechanisms such as SHACL shapes [PK21]. In general, guaranteeing consistency is an open problem for the Semantic Web community, which introducing semantic representations does not solve on its own [GBF⁺22].

Of course, it must be acknowledged that this requires an additional effort spent in identifying and using the appropriate domain ontologies when developing (or mapping) DTs. Nevertheless, this represents a trade-off aligned with the efforts towards DT interoperability [KAA24] and the trend to include some form of semantic representation in many DT tools (including, for instance, the aforementioned and Ditto).

The preliminary descriptions proposed in Chapter 7 can be further extended and refined to cover additional aspects of DTs, such as the representation of DT behaviors, the modeling of DT lifecycle and versioning, or the inclusion of security and privacy-related metadata. Additionally, current descriptions are tailored to the representation of the *present* state of a DT, while, given the nature of DTs, it would be interesting to explore how to represent historical data, but also (possible) future DT states, using semantic models. Finally, the proposed metamodel allows DTs to expose their functionalities as *actions*. Future works could be dedicated to explore how to extend this representation to include more complex DT *services*, such as requesting predictions or running simulations. Preliminary work in this direction has been proposed in [BZPR24], but further investigation is needed to define appropriate models and mechanisms to represent temporal aspects in DTs.

Integration of Intelligent Applications and Digital Twins

The contribution towards the integration of intelligent applications and DTs in DTE has been focused on the conceptual alignment and technological enablers.

Hence, despite the contributions in Part III providing a solid basis to support the integration of DTs and MAS, further work is needed to validate the proposed design principles and micro-architectural patterns with concrete implementations of intelligent applications based on DTEs.

This requires the definition and practical realization of more complex scenarios, where autonomous agents can leverage the unified view of multiple PAs offered by DTEs to implement more advanced behaviors. Moving on to full-scale implementations would allow to evaluate the effectiveness of the currently proposed patterns and to identify potential area of improvements.

An interesting perspective for future works in this direction is the integration of autonomous agents towards the implementation of *cognitive DTs* [MCFA23]. The patterns proposed for augmented behavior in Chapter 5 could be adapted to support the integration of autonomous agents as components of DTs, enabling them to model intelligent behaviors directly at the DT level.

A particularly interesting area in this direction is the integration of prediction and what-if analysis capabilities in DTs with the decision-making capabilities of

autonomous agents. Ideally, agents would be able to request predictions from DTs and use them to inform their actions and decisions, thus creating a feedback loop where DTs provide insights that guide agent behavior, and agents adapt their strategies based on the evolving state of the physical entities represented by the DTs, trying to anticipate future states and optimize outcomes.

Finally, the thesis focuses on MAS and specifically on BDI agents, but other forms of intelligent applications could be explored in relationship to DTEs to evaluate whether the proposed conceptualization supports other forms of intelligence.

The integration of intelligent applications and DTEs also raises important questions around human factors, usability and trust, which are critical for real-world deployments. Future works could explore how to design human-in-the-loop interactions with DTEs that allow users to understand and control the behavior of intelligent applications, as well as to build trust in their decisions and actions.

Addressing Physical Distribution and Operational Management

The DTC middleware, proposed in Chapter 8 provides a first step towards the support of DTEs in distributed computing environments, addressing challenges related to the deployment, discovery and interaction with DTs across different networked nodes. The DTC is designed to be extensible and adaptable to different deployment scenarios, abstracting the complexity of managing DTEs.

However, further work is needed to evaluate the practical implications of deploying DTEs in real-world distributed infrastructures, assessing performance, guaranteeing reliability of the DT-PA connection and synchronization and dynamically adapting to changing functional or non-functional requirements.

The proposal of the DTC opens up interesting perspectives on dynamically migrating DTs across different nodes to optimize resource usage, latency and reliability, which remain unexplored in this thesis.

Likewise, the current design principles to distribute intelligence across DTEs and autonomous agents presented in Chapter 9 could be further refined to address challenges related to the physical distribution of intelligence in the edge-cloud continuum.

A further interesting perspective to explore is the management of DTEs as long-

lived infrastructures, which may require to address challenges related to versioning, deprecation and backward compatibility of DTs and their representations in the DTE.

Evaluation in Real-World Scenarios

Despite the practical applications presented in Part IV have been analyzed thanks to the interaction with stakeholders and domain experts from AUSL Romagna, they currently remain at a prototype level that serves only to demonstrate the feasibility and potential value of the proposed contributions. Real-world settings would provide a more robust validation of the proposed concepts, architectures and tools, allowing to assess their effectiveness in addressing real challenges, their usability by end-users and their performance under realistic conditions.

The preliminary performance evaluation of the HWoDT in Chapter 7 provides encouraging results considering the lack of optimization, but further evaluations are needed to assess scalability and robustness in larger deployments with more complex DTs and higher data volumes. For future works, distributed approaches to the management of the DTE Knowledge Graph (KG), caching strategies and optimization of data retrieval and update mechanisms could be explored to enhance performance, exploiting existing techniques that are being explored in the context of Web-scale knowledge graphs.

Similarly, the proposed DTC in Chapter 8 attempts to facilitate the roles of different stakeholders in the lifecycle of DTs, but further validation is needed to assess its effectiveness in real-world development and deployment processes, considering usability aspects and integration with existing tools and workflows.

Finally, the design principles and micro-architectural patterns proposed in Chapter 9 for the integration of DTs and MAS need to be validated through more extensive implementations of intelligent applications based on DTEs in real-world scenarios, assessing their effectiveness in supporting complex behaviors.

Security and Governance in Digital Twin Ecosystems

The openness and Web-based nature of the HWoDT approach requires to consider security features such as authentication, authorization, and access control which

are critical in DT settings. Although this is not implemented in the HWoDT prototype yet, standard Web security mechanisms (e.g., OAuth 2.0, HTTPS) can be employed to secure interactions.

Fine-grained access control policies must be enforced both at the data layer (e.g., per-property visibility) and at the hypermedia interface level to possibly restrict access to specific resources. For instance, WoT Thing Description (TD)-based DTD already supports the description of security policies [MKK23] to define access control of properties and actions.

In federated DTEs spanning multiple organizations, security mechanisms must be complemented by governance models that define responsibilities, accountability, and trust boundaries across organizational domains.

Furthermore, provenance tracking and data integrity verification (e.g., using digital signatures) can enhance trust. Guarantees of authenticity of the data needs to be established. Namely, *(i)* a DT cannot update the representation of another DT, *(ii)* DT updates need validation so that they conform to the expected structure (e.g. via SHACL).

Finally, the mapping of a DT in its DTKG can purposefully avoid exposing sensitive information, or a DT could support mechanisms to expose different DTKGs depending on authorization level of the client. Existing efforts in the Semantic Web community such as the Solid project⁵ or using access control metadata, may offer interesting solutions to integrate additional security features on the management of the DTE KG.

Summary of Open Challenges and Research Directions

The future work discussed in this section can be summarized along five core research directions.

- **Ecosystem-level orchestration of digital twin functionalities**, including the coordination and composition of simulations, predictions, and actions across multiple DTs to support forward-looking and adaptive decision-making.

⁵<https://solidproject.org/>

- **Governance and evolution of semantic interoperability**, addressing long-term alignment of heterogeneous vocabularies, ontology evolution, validation mechanisms, and organizational aspects influencing shared semantic adoption.
- **Deeper integration of intelligence within DTEs**, through the validation of design principles for autonomous agents, exploration of alternative AI techniques, and tighter coupling between predictive DTs models and cognitive decision-making mechanisms.
- **Scalability, distribution, and deployment strategies**, focusing on performance evaluation, adaptive distribution of DTs, dynamic migration, and optimization across the edge–cloud continuum.
- **Security, trust, and governance mechanisms**, including access control, provenance, validation, and privacy-preserving representations suitable for open and federated DTEs.

Together, these challenges define a research roadmap that extends the contributions of this thesis and provides concrete entry points for future work aimed at advancing DTEs from experimental systems to robust infrastructures supporting real-world intelligent applications. Overall, this thesis should be seen not as a closed solution, but as the definition of an architectural and conceptual baseline upon which future research can systematically build to realize fully operational Digital Twin Ecosystems.

Funding

This work has been funded by the European Union - NextGenerationEU and Azienda Unità Sanitaria Locale (AUSL) della Romagna through the Italian "National Recovery and Resilience Plan" (PNRR) Mission 4, Component 2, Investment 3.3 (DM 352/2022) - CUP J33C22001400009



Bibliography

- [AAOL25] Marcos Alonso, Aníbal M. Astobiza, and Ramón Ortega Lozano. AI-mediated healthcare and trust. a trust-construct and trust-factor framework for empirical research. *Artificial Intelligence Review*, 58(11):337, August 2025. doi:10.1007/s10462-025-11306-7.
- [ACS24] Stefano V. Albrecht, Filippas Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, December 2024.
- [ADLM22] Roberto Aringhieri, Davide Duma, Paolo Landa, and Simona Mancini. Combining workload balance and patient priority maximisation in operating room planning through hierarchical multi-objective optimisation. *Eur. J. Oper. Res.*, 298(2):627–643, 2022. doi:10.1016/J.EJOR.2021.07.033.
- [AE17] Kazi Masudul Alam and Abdulmotaleb El-Saddik. C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. *IEEE Access*, 5:2050–2062, 2017. doi:10.1109/ACCESS.2017.2657006.
- [AGD21] Ahmad Alelaimat, Aditya Ghose, and Hoa Khanh Dam. Abductive design of BDI agent-based digital twins of organizations. In Takahiro Uchiya, Quan Bai, and Iván Marsá Maestre, editors, *PRIMA 2020: Principles and Practice of Multi-Agent Systems*, page 377–385, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-69322-0_27.

- [Age25] Agenzia per l'Italia Digitale. Piano triennale 2024–2026: Aggiornamento 2025. Technical report, Agenzia per l'Italia Digitale, 2025. Accessed: 2025-10-16. URL: https://www.agid.gov.it/sites/agid/files/2025-02/Piano_Triennale_2024-2026_Aggiornamento2025acc_0.pdf.
- [Agh12] Sareh Aghaei. Evolution of the world wide web: From web 1.0 to web 4.0. *International journal of Web & Semantic Technology*, 3(1):1–10, January 2012. doi:10.5121/ijwest.2012.3101.
- [AHK23] Cleber Jorge Amaral, Jomi Fred Hübner, and Timotheus Kampik. TDD for AOP: test-driven development for agent-oriented programming. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 3038–3040. ACM, 2023. URL: <https://dl.acm.org/doi/10.5555/3545946.3599165>.
- [AJMA20] Jameela Al-Jaroodi, Nader Mohamed, and Eman Abukhousa. Health 4.0: On the way to realizing the healthcare of the future. *Ieee Access*, 8:211189–211210, November 2020. doi:10.1109/ACCESS.2020.3038858.
- [AKD25] Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B. Divya. Agentic AI: Autonomous intelligence for complex goals—a comprehensive survey. *IEEE Access*, 13:18912–18936, 2025. doi:10.1109/ACCESS.2025.3532853.
- [AKK⁺23] Mamoun Alazab, Latif U. Khan, Srinivas Koppu, Swarna Priya Ramu, Iyapparaja M, Parimala Boobalan, Thar Baker, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Ahamed Aljuhani. Digital twins for healthcare 4.0—recent advances, architecture, and open challenges. *IEEE Consumer Elec-*

- tronics Magazine*, 12(6):29–37, November 2023. doi:10.1109/MCE.2022.3208986.
- [AKL⁺23] Mansoor Ali, Georges Kaddoum, Wen-Tai Li, Chau Yuen, Muhammad Tariq, and H. Vincent Poor. A smart digital twin enabled security framework for vehicle-to-grid cyber-physical systems. *IEEE Trans. Inf. Forensics Secur.*, 18:5258–5271, 2023. doi:10.1109/TIFS.2023.3305916.
- [AKM16] Malte Aschermann, Philipp Kraus, and Jörg P. Müller. Light-Jason - A BDI framework inspired by Jason. In Natalia Criado Pacheco, Carlos Carrascosa, Nardine Osman, and Vicente Julián Inglada, editors, *Multi-Agent Systems and Agreement Technologies - 14th European Conference, EUMAS 2016, and 4th International Conference, AT 2016, Valencia, Spain, December 15-16, 2016, Revised Selected Papers*, volume 10207 of *Lecture Notes in Computer Science*, pages 58–66. Springer, 2016. doi:10.1007/978-3-319-59294-7_6.
- [AKN20] Dmytro Adamenko, Steffen Kunnen, and Arun Nagarajah. Comparative analysis of platforms for designing a digital twin. In Vitalii Ivanov, Justyna Trojanowska, Ivan Pavlenko, Jozef Zajac, and Dragan Peraković, editors, *Advances in Design, Simulation and Manufacturing III*, page 3–12, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-50794-7_1.
- [AKP24] Sarthak Acharya, Arif Ali Khan, and Tero Päiväranta. Interoperability levels and challenges of digital twins in cyber-physical systems. *Journal of Industrial Information Integration*, 42:100714, November 2024. doi:10.1016/j.jii.2024.100714.
- [Ans25] Ansys. Ansys twin builder. Product documentation, 2025. Accessed: 2025-09-23. URL: <https://www.ansys.com/products/digital-twin/ansys-twin-builder>.

- [Any18] AnyLogic. An introduction to digital twin development. White paper, 2018. URL: <https://www.anylogic.com/resources/white-papers/an-introduction-to-digital-twin-development/>.
- [APA⁺21] Muhammad Ayaz, Muhammad F Pasha, Mohammed Y Alzahrani, Rahmat Budiarto, and Deris Stiawan. The fast health interoperability resources (FHIR) standard: Systematic literature review of implementations, applications, challenges and opportunities. *JMIR Med Inform*, 9(7):e21929, 2021. doi:10.2196/21929.
- [AQHAA21] Tariq Alsboui, Yongrui Qin, Richard Hill, and Hussain Al-Aqrabi. Distributed intelligence in the internet of things: Challenges and opportunities. *SN Computer Science*, 2(4):277, May 2021. doi:10.1007/s42979-021-00677-7.
- [AST21] Juuso Autiosalo, Joshua Siegel, and Kari Tammi. Twin-base: Open-source server software for the digital twin web. *IEEE Access*, 9:140779–140798, 2021. URL: <https://ieeexplore.ieee.org/abstract/document/9568895>, doi:10.1109/ACCESS.2021.3119487.
- [Aus75] J. L. Austin. *How to Do Things with Words: Second Edition*. Harvard University Press, April 1975.
- [AXZL21] Shohin Ahelerooff, Xun Xu, Ray Y. Zhong, and Yuqian Lu. Digital twin as a service (dtaas) in industry 4.0: An architecture reference model. *Adv. Eng. Informatics*, 47:101225, 2021. doi:10.1016/J.AEI.2020.101225.
- [BADD⁺20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020. doi:10.1016/j.inffus.2019.12.012.

- [Bar24] Alfred Barnard. Digital twin frameworks: A comparative study of AAS, OPC UA, and AML. In *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*, page 1–6, July 2024. doi:10.1109/ICECET61485.2024.10698583.
- [BBC⁺09] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for continuous querying. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 1061–1062. ACM, 2009. doi:10.1145/1526709.1526856.
- [BBC⁺24a] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, Danilo Pinini, Andrea Omicini, and Alessandro Ricci. Concurrency model of BDI programming frameworks: Why should we control it? In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, pages 2147–2149. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024. doi:10.5555/3635637.3663089.
- [BBC⁺24b] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, Danilo Pinini, Alessandro Ricci, and Andrea Omicini. On the external concurrency of current BDI frameworks for MAS. In Daniela Briola, Rafael C. Cardoso, and Brian Logan, editors, *Engineering Multi-Agent Systems - 12th International Workshop, EMAS 2024, Auckland, New Zealand, May 6-7, 2024, Revised Selected Papers*, volume 15152 of *Lecture Notes in Computer Science*, pages 42–63. Springer, 2024. doi:10.1007/978-3-031-71152-7_3.
- [BBC⁺25] Samuele Burattini, Martina Baiardi, Giovanni Ciatto, Danilo Pinini, et al. The gap between BDI agents and semantic hypermedia and what we can do about it. In *CProceedings of the Sec-*

- ond International Workshop on Hypermedia Multi-Agent Systems (HyperAgents 2025) co-located with 28th European Conference on Artificial Intelligence (ECAI 2025)*, volume 4084, pages 18–27. CEUR-WS, 2025. URL: <https://ceur-ws.org/Vol-4084/short3.pdf>.
- [BBCP23] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, and Danilo Pianini. Jakta: BDI agent-oriented programming in pure kotlin. In Vadim Malvone and Aniello Murano, editors, *Multi-Agent Systems - 20th European Conference, EUMAS 2023, Naples, Italy, September 14-15, 2023, Proceedings*, volume 14282 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 2023. doi: 10.1007/978-3-031-43264-4_4.
- [BBCP24] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, and Danilo Pianini. Blending BDI agents with object-oriented and functional programming with jakta. *SN Comput. Sci.*, 5(8):1003, 2024. doi: 10.1007/S42979-024-03244-Y.
- [BBCPed] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, and Danilo Pianini. Using discrete event simulation to test bdi systems before deployment. *In Revision, Submitted*.
- [BBD⁺18] Alexandra Bolton, Lorraine Butler, Ian Dabson, Mark Enzer, Matthew Evans, Tim Fenemore, Fergus Harradence, Emily Keaney, Anne Kemp, Alexandra Luck, Neill Pawsey, Steve Saville, Jennifer Schooling, Miranda Sharp, Tom Smith, Jeni Tennis, Jennifer Whyte, Alan Wilson, and Chara Makri. *Gemini Principles*. Centre for Digital Built Britain, 2018. doi: 10.17863/CAM.32260.
- [BBE⁺19] Bergthor Björnsson, Carl Borrebaeck, Nils Elander, Thomas Gasslander, Danuta R. Gawel, Mika Gustafsson, Rebecka Jörnsten, Eun Jung Lee, Xinxu Li, Sandra Lilja, David Martínez-Enguita, Andreas Matussek, Per Sandström, Samuel Schäfer,

- Margaretha Stenmarker, X. F. Sun, Oleg Sysoev, Huan Zhang, Mikael Benson, and on behalf of the Swedish Digital Twin Consortium. Digital twins to personalize medicine. *Genome Medicine*, 12(1):4, December 2019. doi:10.1186/s13073-019-0701-3.
- [BBF⁺23] Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories. *Computers in Industry*, 149:103918, August 2023. doi:10.1016/j.compind.2023.103918.
- [BBF⁺24a] Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. Exploiting microservices and serverless for digital twins in the cloud-to-edge continuum. *Future Generation Computer Systems*, 157:275–287, August 2024. doi:10.1016/j.future.2024.03.052.
- [BBF⁺24b] Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. ODTE: a metric for digital twin entanglement. *IEEE Open J. Commun. Soc.*, 5:2377–2390, 2024. doi:10.1109/OJCOMS.2024.3385659.
- [BBH⁺13] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761, 2013. doi:10.1016/j.scico.2011.10.004.
- [BBHR19] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, and Alessandro Ricci. Dimensions in programming multi-agent systems. *The Knowledge Engineering Review*, 34:e2, January 2019. doi:10.1017/S026988891800005X.
- [BBHR20] Olivier Boissier, Rafael H. Bordini, Jomi Hubner, and Alessandro Ricci. *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. MIT Press, 2020.

- [BBKY20] Sergey Yevgenievich Barykin, Andrey Aleksandrovich Bochkarev, Olga Vladimirovna Kalinina, and Vladimir Konstantinovich Yadykin. Concept for a supply chain digital twin. *International Journal of Mathematical, Engineering and Management Sciences*, 5(6):1498–1515, December 2020. doi:10.33889/IJMEMS.2020.5.6.111.
- [BBM⁺24] Antonello Barbone, Samuele Burattini, Matteo Martinelli, Marco Picone, Alessandro Ricci, and Antonio Viridis. Digital twin continuum: a key enabler for pervasive cyber-physical environments. In *33rd International Conference on Computer Communications and Networks, ICCCN 2024, Kailua-Kona, HI, USA, July 29-31, 2024*, pages 1–9. IEEE, 2024. doi:10.1109/ICCCN61486.2024.10637565.
- [BCB21] Sadeer Beden, Qiushi Cao, and Arnold Beckmann. Semantic asset administration shells in industry 4.0: A survey. In *4th IEEE International Conference on Industrial Cyber-Physical Systems, ICPS 2021, Victoria, BC, Canada, May 10-12, 2021*, pages 31–38. IEEE, 2021. doi:10.1109/ICPS49255.2021.9468266.
- [BCGR23] Samuele Burattini, Andrei Ciortea, Meshua Galassi, and Alessandro Ricci. Towards framing the agents & artifacts conceptual model at the knowledge level: First ideas and experiments. In Andrei Ciortea, Mehdi Dastani, and Jieting Luo, editors, *Engineering Multi-Agent Systems - 11th International Workshop, EMAS 2023, London, UK, May 29-30, 2023, Revised Selected Papers*, volume 14378 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2023. doi:10.1007/978-3-031-48539-8_14.
- [BCH⁺23] Olivier Boissier, Andrei Ciortea, Andreas Harth, Alessandro Ricci, and Danai Vachtsevanou. Agents on the web (dagstuhl seminar 23081). *Dagstuhl Reports*, 13(2):71–162, 2023. URL: <https://drops.dagstuhl.de/entities/document/10.4230/DagRep.13.2.71>, doi:10.4230/DagRep.13.2.71.

- [BCHR21] Olivier Boissier, Andrei Ciortea, Andreas Harth, and Alessandro Ricci. Autonomous agents on the web (dagstuhl seminar 21072). *Dagstuhl Reports*, 11(1):24–100, 2021. URL: <https://drops.dagstuhl.de/entities/document/10.4230/DagRep.11.1.24>, doi:10.4230/DagRep.11.1.24.
- [BDD⁺20] M. Majid Butt, Indrakshi Dey, Merim Dzaferagic, Maria Murphy, Nicholas Kaminski, and Nicola Marchetti. Agent-based modeling for distributed decision support in an IoT network. *IEEE Internet of Things Journal*, 7(8):6919–6931, 2020. doi:10.1109/JIOT.2020.2976802.
- [BEFSH⁺20] Rafael H. Bordini, Amal El Fallah Seghrouchni, Koen Hindriks, Brian Logan, and Alessandro Ricci. Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems*, 34(2):1–31, October 2020. Company: Springer Distributor: Springer Institution: Springer Label: Springer publisher: Springer US. doi:10.1007/s10458-020-09453-y.
- [BFRLG21] Arnaud Braud, Gaël Fromentoux, Benoit Radier, and Olivier Le Grand. The road to european digital sovereignty with Gaia-X and IDSA. *IEEE Network*, 35(2):4–5, March 2021. doi:10.1109/MNET.2021.9387709.
- [BG16] Tim Benson and Grahame Grieve. *Principles of Health Interoperability: SNOMED CT, HL7 and FHIR*. Springer International Publishing, Cham, Switzerland, 3rd edition, 2016. doi:10.1007/978-3-319-30370-3.
- [BGZ06] Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Springer Science & Business Media, April 2006.
- [BHBL23] Christian Bizer, Tom Heath, and Tim Berners-Lee. *Linked Data - The Story So Far*, volume 52, page 115–143. Association for

Computing Machinery, New York, NY, USA, 1 edition, 2023. doi: 10.1145/3591366.3591378.

- [BHH⁺10] Joost Broekens, Maaïke Harbers, Koen Hindriks, Karel Bosch, Catholijn Jonker, and John-jules Meyer. Do you get it? user-evaluated explainable BDI agents. In *Multiagent System Technologies: 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings 8*, volume 6251, pages 28–39. Springer, 09 2010. doi:10.1007/978-3-642-16178-0_5.
- [BHS07] Olivier Boissier, Jomi Fred Hübner, and Jaime Simão Sichman. Organization oriented programming: From closed to open organizations. In Gregory M. P. O’Hare, Alessandro Ricci, Michael J. O’Grady, and Oğuz Dikenelli, editors, *Engineering Societies in the Agents World VII*, page 86–105, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-75524-1_5.
- [BHW07] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, November 2007.
- [BK13] Premkumar Balaraman and Kalpana Kosalram. E-hospital management & hospital information systems – changing trends. *International Journal of Information Engineering and Electronic Business*, 5(1):50–58, May 2013. doi:10.5815/ijieeb.2013.01.06.
- [BKZ⁺21] Bart Bogaerts, Bas Ketsman, Younes Zeboudj, Heba Aamer, Ruben Taelman, and Ruben Verborgh. Link traversal with distributed subweb specifications. In Sotiris Moschogiannis, Rafael Peñaloza, Jan Vanthienen, Ahmet Soylu, and Dumitru Roman, editors, *Rules and Reasoning - 5th International Joint Conference, RuleML+RR 2021, Leuven, Belgium, September 13-15, 2021, Proceedings*, volume 12851 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2021. doi:10.1007/978-3-030-91167-6_5.

BIBLIOGRAPHY

- [BL89] Timothy J. Berners-Lee. Information management: A proposal. Technical report, CERN, Geneva, Switzerland, 1989. URL: <https://cds.cern.ch/record/369245/files/dd-89-001>.
- [BL94a] Tim Berners-Lee. The future of the web. Video, 1994. Recorded May 1994, Geneva, Switzerland. URL: <https://videos.cern.ch/record/2671957>, doi:10.17181/CDS.2671957.
- [BL94b] Tim Berners-Lee. Plenary at WWW Geneva 94. W3C (HTML version), 1994. First International World Wide Web Conference, September 1994. URL: <https://www.w3.org/Talks/WWW94Tim/>.
- [BL96] T. Berners-Lee. WWW: past, present, and future. *Computer*, 29(10):69–77, October 1996. doi:10.1109/2.539724.
- [BL06] Tim Berners-Lee. Linked data: Design issues. Online; “personal view only”, July 2006. Last changed 18 June 2009. URL: <https://www.w3.org/DesignIssues/LinkedData.html>.
- [BLHL23] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*, pages 91–103. Association for Computing Machinery, New York, NY, 2023. Originally in *Scientific American*, May 2001. doi:10.1145/3591366.3591376.
- [BMC⁺23] Samuele Burattini, Sara Montagna, Angelo Croatti, Nicola Gentili, Alessandro Ricci, Laura Leonardi, Serafino Pandolfini, and Sofia Tosi. An ecosystem of digital twins for operating room management. In João Rafael Almeida, Myra Spiliopoulou, José Alberto Benítez-Andrades, Giuseppe Placidi, Alejandro Rodríguez González, Rosa Sicilia, and Bridget Kane, editors, *36th IEEE International Symposium on Computer-Based Medical Systems, CBMS 2023, L’Aquila, Italy, June 22-24, 2023*, pages 770–775. IEEE, 2023. doi:10.1109/CBMS58004.2023.00317.

- [BMG⁺24] Samuele Burattini, Sara Montagna, Nicola Gentili, Francesca Galardi, Roberto Vespignani, Paolo Zanatto, Angelo Croatti, and Alessandro Ricci. A digital twins approach for oncologic pharmaceutical supply chain. In *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, PerCom 2024 - Workshops, Biarritz, France, March 11-15, 2024*, pages 94–99. IEEE, 2024. doi:10.1109/PERCOMWORKSHOPS59983.2024.10502569.
- [BMM⁺25] Samuele Burattini, Stefano Mariani, Sara Montagna, Marco Picone, and Alessandro Ricci. Distributing intelligent functionalities in the internet of things with agents and digital twins. *Internet Things*, 31:101560, 2025. doi:10.1016/J.IOT.2025.101560.
- [BMMR15] Diego Bosca, David Moner, Jose Alberto Maldonado, and Montserrat Robles. Combining archetypes with fast health interoperability resources in future-proof health information systems. *Studies in Health Technology and Informatics*, 210:180–184, 2015.
- [BMP24] Subrato Bharati, M. Rubaiyat Hossain Mondal, and Prajoy Podder. A review on explainable artificial intelligence for healthcare: Why, how, and when? *IEEE Transactions on Artificial Intelligence*, 5(4):1429–1442, April 2024. doi:10.1109/TAI.2023.3266418.
- [BPL05] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Extending the capability concept for flexible BDI agent modularization. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems, Third International Workshop, ProMAS 2005, Utrecht, The Netherlands, July 26, 2005, Revised and Invited Papers*, volume 3862 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2005. doi:10.1007/11678823_9.

- [BR16] Stefan Boschert and Roland Rosen. *Digital Twin—The Simulation Aspect*, page 59–74. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-32156-1_5.
- [BR20] Michael Bosello and Alessandro Ricci. From programming agents to educating agents – a Jason-based framework for integrating learning in the development of cognitive agents. In Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, editors, *Engineering Multi-Agent Systems*, page 175–194, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-51417-4_9.
- [Bra87] Michael Bratman. *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, Cambridge, 1987.
- [Bra07] Steve Bratt. Semantic web, and other technologies to watch. Presentation, 2007. W3C presentation, January 30, 2007. URL: <https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/>.
- [Bra18] Mark L. Braunstein. Healthcare in the age of interoperability: The promise of fast healthcare interoperability resources. *IEEE Pulse*, 9(6):24–27, November 2018. doi:10.1109/MPUL.2018.2869317.
- [BSH] Payam M. Barnaghi, Amit P. Sheth, and Cory A. Henson. From data to actionable knowledge: Big data challenges in the web of things. 28(6):6–11. doi:10.1109/MIS.2013.142.
- [Bur23] Samuele Burattini. Supporting adaptive multi-agent systems with digital twins environments. In Vadim Malvone and Aniello Murano, editors, *Multi-Agent Systems - 20th European Conference, EUMAS 2023, Naples, Italy, September 14-15, 2023, Proceedings*, volume 14282 of *Lecture Notes in Computer Science*, pages 445–451. Springer, 2023. doi:10.1007/978-3-031-43264-4_30.
- [BW01] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001. doi:10.1145/603867.603884.

- [BYB25] Katharine Beaumont, Elena Yan, and Samuele Burattini. Engineering inter-agent explainability in bdi agents. In *Explainable, Trustworthy, and Responsible AI and Multi-Agent Systems: 7th International Workshop, EXTRAAMAS 2025, Detroit, MI, USA, May 19–20, 2025, Revised Selected Papers*, page 147. Springer Nature, 2025.
- [BYBC25] Katharine Beaumont, Elena Yan, Samuele Burattini, and Rem W. Collier. Agent toolkits: Towards explainable agency in ASTRA. *Accepted*, 2025. URL: <https://euramas.github.io/eumas2025/#EMAS%201>.
- [BZPR24] Samuele Burattini, Antoine Zimmermann, Marco Picone, and Alessandro Ricci. Towards linked data for ecosystems of digital twins. In Manuel Wimmer, Alexander Egyed, Benoît Combemale, and Marsha Chechik, editors, *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion 2024, Linz, Austria, September 22–27, 2024*, pages 332–337. ACM, 2024. doi:10.1145/3652620.3688245.
- [CAB⁺25] Giovanni Ciatto, Gianluca Aguzzi, Riccardo Battistini, Martina Baiardi, Samuele Burattini, and Alessandro Ricci. Exploiting GenAI for plan generation in BDI agents. In *ECAI 2025*, page 3495–3502. IOS Press, 2025. doi:10.3233/FAIA251223.
- [CAMM⁺20] Jorge Corral-Acero, Francesca Margara, Maciej Marciniak, Cristobal Roder, Filip Loncaric, Yingjing Feng, Andrew Gilbert, Joao F Fernandes, Hassaan A Bukhari, Ali Wajdan, Manuel Villegas Martinez, Mariana Sousa Santos, Mehrdad Shamohammdi, Hongxing Luo, Philip Westphal, Paul Leeson, Paolo DiAchille, Viatcheslav Gurev, Manuel Mayr, Liesbet Geris, Pras Pathmanathan, Tina Morrison, Richard Cornelussen, Frits Prinzen, Tammo Delhaas, Ada Doltra, Marta Sitges, Edward J Vigmond, Ernesto Zacur, Vicente Grau, Blanca Rodriguez, Espen W

- Remme, Steven Niederer, Peter Mortier, Kristin McLeod, Mark Potse, Esther Pueyo, Alfonso Bueno-Orovio, and Pablo Lamata. The ‘digital twin’ to enable the vision of precision cardiology. *European Heart Journal*, 41(48):4556–4564, December 2020. doi:10.1093/eurheartj/ehaa159.
- [Cas12] Cristiano Castelfranchi. Ascribing minds. *Cogn. Process.*, 13(Supplement-2):415–425, 2012. doi:10.1007/S10339-011-0423-Y.
- [CBC⁺25] Matteo Castellucci, Samuele Burattini, Andrei Ciortea, Jérémy Lemée, Danai Vachtsevanou, Alessandro Ricci, and Simon Mayer. Towards agents’ embodiment in hypermedia multi-agent systems. In Rem Collier, Alessandro Ricci, Vivek Nallur, Samuele Burattini, and Andrea Omicini, editors, *Multi-Agent Systems*, pages 361–381, Cham, 2025. Springer Nature Switzerland.
- [CBLVK21] Sarven Capadisli, Tim Berners-Lee, Ruben Verborgh, and Kjetil Kjernsmo. Solid protocol. W3c solid community group working draft, World Wide Web Consortium, 2021. URL: <https://solidproject.org/TR/2021/protocol-20211217>.
- [CBR19] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. Engineering world-wide multi-agent systems with hypermedia. In Danny Weyns, Viviana Mascardi, and Alessandro Ricci, editors, *Engineering Multi-Agent Systems*, page 285–301, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-25693-7_15.
- [CCMO21] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. Logic-based technologies for multi-agent systems: A systematic literature review. *Autonomous Agents and Multi-Agent Systems*, 35(1):1:1–1:67, 2021. doi:10.1007/s10458-020-09478-3.
- [CCP⁺23] Simiao Chen, Zhong Cao, Klaus Prettner, Michael Kuhn, Juntao Yang, Lirui Jiao, Zhuoran Wang, Weimin Li, Pascal Geldsetzer,

- Till Bärnighausen, David E. Bloom, and Chen Wang. Estimates and Projections of the Global Economic Cost of 29 Cancers in 204 Countries and Territories From 2020 to 2050. *JAMA Oncology*, 9(4):465–472, 04 2023.
- [CF21] Rafael C. Cardoso and Angelo Ferrando. A review of agent-based programming for multi-agent systems. *Computers*, 10(2):16, February 2021. doi:10.3390/computers10020016.
- [CFG⁺16] Franco Cicirelli, Giancarlo Fortino, Andrea Giordano, Antonio Guerrieri, Giandomenico Spezzano, and Andrea Vinci. On the design of smart homes: A framework for activity recognition in home environment. *J. Medical Syst.*, 40(9):200:1–200:17, 2016. URL: <https://doi.org/10.1007/s10916-016-0549-7>, doi:10.1007/S10916-016-0549-7.
- [CGL⁺24] Andrei Ciortea, Fabien Gandon, Maxime Lefrançois, Danaï Vachtsevanou, Chistopher Leturc, Olivier Boissier, Luis Gustavo Nardin, Simon Mayer, Victor Charpenay, and Antoine Zimmermann. Hypermedia MAS Core Ontology. Technical report, HyperAgents, 2024. URL: <https://purl.org/hmas/core>.
- [CGMR20] Angelo Croatti, Matteo Gabellini, Sara Montagna, and Alessandro Ricci. On the integration of agents and digital twins in healthcare. *Journal of Medical Systems*, 44(9):161, August 2020. doi:10.1007/s10916-020-01623-5.
- [CKK16] Victor Charpenay, Sebastian Käbisich, and Harald Kosch. Introducing thing descriptions and interactions: An ontology for the web of things. In *Joint Proceedings of SR 2016 and SWIT 2016 (Stream Reasoning + Semantic Web Technologies for the Internet of Things)*, volume Vol. 1783 of *CEUR Workshop Proceedings*, pages 55–66, October 2016. URL: <https://ceur-ws.org/Vol-1783/paper-06.pdf>.

- [CLPVK23] Victor Charpenay, Maxime Lefrançois, María Poveda Villalón, and Sebastian Käbisch. Web of things (WoT) thing description (TD) ontology. W3C Editor's Draft, December 2023. Editor's draft; draft document by the W3C Web of Things Working Group. URL: <https://www.w3.org/2019/wot/td>.
- [CMAA⁺22] Javier Conde, Andrés Muñoz-Arcentales, Álvaro Alonso, Sonsoles López-Pernas, and Joaquín Salvachúa. Modeling digital twin data and architecture: A building guide with fiware as enabling technology. *IEEE Internet Computing*, 26(3):7–14, 2022. doi:10.1109/MIC.2021.3056923.
- [CMB⁺23] Giovanni Ciatto, Matteo Magnini, Berk Buzcu, Reyhan Aydoğan, and Andrea Omicini. A general-purpose protocol for multi-agent based explanations. In *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, pages 38–58. Springer, 2023.
- [CMG⁺19] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. A decade in hindsight: The missing bridge between multi-agent systems and the world wide web. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 1659–1663, Richland, SC, May 2019. International Foundation for Autonomous Agents and Multiagent Systems.
- [CMM18] Andrei Ciortea, Simon Mayer, and Florian Michahelles. Repurposing manufacturing lines on the fly with multi-agent systems for the web of things. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 813–822, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.
- [CMS⁺17] Davide Calvaresi, Mauro Marinoni, Arnon Sturm, Michael Schumacher, and Giorgio Buttazzo. The challenge of real-time multi-

- agent systems for enabling IoT and CPS. In *Proceedings of the International Conference on Web Intelligence*, WI '17, page 356–364, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3106426.3106518.
- [Cos05] Massimo Cossentino. *From Requirements to Code with PASSI Methodology*, page 79–106. IGI Global Scientific Publishing, 2005. doi:10.4018/978-1-59140-581-8.ch004.
- [CP14] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, February 2014. <https://www.w3.org/TR/2014/REC-turtle-20140225/>.
- [CRL15] Rem W. Collier, Sean Edward Russell, and David Lillis. Reflecting on agent programming with AgentSpeak(L). In *PRIMA 2015: Principles and Practice of Multi-Agent Systems*, Lecture Notes in Computer Science, pages 351–366. Springer International Publishing, Bertinoro, Italy, 2015. doi:10.1007/978-3-319-25524-8_22.
- [CRvdWA19] João Vidal Carvalho, Álvaro Rocha, Rogier van de Wetering, and António Abreu. A maturity model for hospital information systems. *Journal of Business Research*, 94:388–399, January 2019. doi:10.1016/j.jbusres.2017.12.012.
- [CSB⁺19] Flavio Cirillo, Gürkan Solmaz, Everton Luís Berz, Martin Bauer, Bin Cheng, and Ernő Kovacs. A standard-based open source iot platform: Fiware. *IEEE Internet of Things Magazine*, 2(3):12–18, 2019. doi:10.1109/IOTM.0001.1800022.
- [CSS19] Salvatore Cavalieri, Marco Giuseppe Salafia, and Marco Stefano Scroppo. Integrating OPC UA with web technologies to enhance interoperability. *Comput. Stand. Interfaces*, 61:45–64, 2019. doi:10.1016/J.CSI.2018.04.004.

- [Dav01] Paul Davidsson. Multi agent based simulation: Beyond social simulation. In Scott Moss and Paul Davidsson, editors, *Multi-Agent-Based Simulation*, page 97–107, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44561-7_7.
- [DBB19] Violeta Damjanovic-Behrendt and Wernher Behrendt. An open source approach to the design and implementation of digital twins for smart manufacturing. *International Journal of Computer Integrated Manufacturing*, 32:366–384, May 2019. doi:10.1080/0951192X.2019.1599436.
- [DD10] Virginia Dignum and Frank Dignum. Designing agent systems: state of the practice. *Int. J. Agent Oriented Softw. Eng.*, 4(3):224–243, 2010. doi:10.1504/IJA0SE.2010.036983.
- [Deg14] P. Degoulet. *Hospital Information Systems*, page 289–313. Health Informatics. Springer Paris, Paris, 2014. doi:10.1007/978-2-8178-0478-1_12.
- [Den18] Louise A. Dennis. The MCAPL framework including the agent infrastructure layer and agent java pathfinder. *Journal of Open Source Software*, 3(24):617, 2018. doi:10.21105/joss.00617.
- [DES⁺15] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The BigDAWG polystore system. *SIGMOD Rec.*, 44(2):11–16, August 2015. doi:10.1145/2814710.2814713.
- [DF08] Louise A Dennis and Berndt Farwer. Gwendolen: A BDI language for verifiable agents. In *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 16–23, 2008.
- [DFM⁺24] Giovanni Delussu, Francesca Frexia, Cecilia Mascia, Alessandro Sulis, Vittorio Meloni, Mauro Del Rio, and Luca Lianas. A survey

- of openEHR clinical data repositories. *International Journal of Medical Informatics*, 191:105591, November 2024. doi:10.1016/j.ijmedinf.2024.105591.
- [DGRM⁺20] Alan Davoust, Patrick Gavigan, Cristina Ruiz-Martin, Guillermo Trabes, Babak Esfandiari, Gabriel Wainer, and Jeremy James. An Architecture for Integrating BDI Agents with a Simulation Environment. In *Engineering Multi-Agent Systems*, pages 67–84, Cham, 2020. doi:10.1007/978-3-030-51417-4_4.
- [dHLH⁺22] Anne A. H. de Hond, Artuur M. Leeuwenberg, Lotty Hooft, Ilse M. J. Kant, Steven W. J. Nijman, Hendrikus J. A. van Os, Jiska J. Aardoom, Thomas P. A. Debray, Ewoud Schuit, Maarten van Smeden, Johannes B. Reitsma, Ewout W. Steyerberg, Niels H. Chavannes, and Karel G. M. Moons. Guidelines and quality criteria for artificial intelligence-based prediction models in healthcare: a scoping review. *npj Digital Medicine*, 5(1):2, January 2022. doi:10.1038/s41746-021-00549-7.
- [DJR⁺22] Manuela Dalibor, Nico Jansen, Bernhard Rumpe, David Schmalzing, Louis Wachtmeister, Manuel Wimmer, and Andreas Wortmann. A cross-domain systematic mapping study on software engineering for digital twins. *J. Syst. Softw.*, 193:111361, 2022. doi:10.1016/J.JSS.2022.111361.
- [DLPH08] Rainer Drath, Arndt Lüder, Jörn Peschke, and Lorenz Hundt. Automationml - the glue for seamless automation engineering. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*, pages 616–623. IEEE, 2008. doi:10.1109/ETFA.2008.4638461.
- [DLS19] Fabio D’Urso, Carmelo Fabio Longo, and Corrado Santoro. Programming intelligent IoT systems with a python-based declarative tool. In Claudio Savaglio, Giancarlo Fortino, Giovanni Ciatto, and

- Andrea Omicini, editors, *Proceedings of the 1st Workshop on Artificial Intelligence and Internet of Things; co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019)*, volume 2502 of *CEUR Workshop Proceedings*, pages 68–81, Rende (CS), Italy, 11 2019. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-2502/paper5.pdf>.
- [dMRdC⁺22] Blanda Helena de Mello, Sandro José Rigo, Cristiano André da Costa, Rodrigo da Rosa Righi, Bruna Donida, Marta Rosecler Bez, and Luana Carina Schunke. Semantic interoperability in health records standards: a systematic literature review. *Health and Technology*, 12(2):255–272, March 2022. doi:10.1007/s12553-022-00639-w.
- [DRH19] Rainer Drath, Markus Rentschler, and Michael Hoffmeister. The AutomationML component description in the context of the asset administration shell. In *24th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2019, Zaragoza, Spain, September 10-13, 2019*, pages 1278–1281. IEEE, 2019. doi:10.1109/ETFA.2019.8869214.
- [DSH19] William Danilczyk, Yan Sun, and Haibo He. ANGEL: An intelligent digital twin framework for microgrid security. In *2019 North American Power Symposium (NAPS)*, page 1–6, October 2019. doi:10.1109/NAPS46351.2019.9000371.
- [DSH21] William Danilczyk, Yan Lindsay Sun, and Haibo He. Smart grid anomaly detection using a deep learning digital twin. In *2020 52nd North American Power Symposium (NAPS)*, pages 1–6, 2021. doi:10.1109/NAPS50074.2021.9449682.
- [dSNS13] Herbert Van de Sompel, Michael L. Nelson, and Robert Sanderson. HTTP framework for time-based access to resource states (Memento). RFC 7089, Internet Engineering Task Force

- (IETF), 2013. URL: <https://datatracker.ietf.org/doc/html/rfc7089>, doi:10.17487/RFC7089.
- [DWZ24] Yu Du, Fangyun Wei, and Hongyang Zhang. AnyTool: Self-reflective, hierarchical agents for large-scale API calls. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org, 2024.
- [DZS21] Tianhu Deng, Keren Zhang, and Zuo-Jun (Max) Shen. A systematic review of a digital twin city: A new pattern of urban governance toward smart cities. *Journal of Management Science and Engineering*, 6(2):125–134, 2021. doi:10.1016/j.jmse.2021.03.003.
- [EAG21] Haya Elayan, Moayad Aloqaily, and Mohsen Guizani. Digital twin for intelligent context-aware iot healthcare systems. *IEEE Internet of Things Journal*, 8(23):16749–16757, December 2021. doi:10.1109/JIOT.2021.3051158.
- [EGF⁺21] Lukas Esterle, Cláudio Gomes, Mirgita Frasheri, Henrik Ejersbo, Sven Tomforde, and Peter G. Larsen. Digital twins for collaboration and self-integration. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, page 172–177, 2021. doi:10.1109/ACSOS-C52956.2021.00040.
- [Eib] A. E. Eiben. Evolutionary computing and autonomic computing: Shared problems, shared solutions? In Özalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad P. A. van Moorsel, and Maarten van Steen, editors, *Self-star Properties in Complex Information Systems, Conceptual and Practical Foundations [the book is a result from a workshop at Bertinoro, Italy, Summer 2004]*, volume 3460 of *Lecture Notes in Computer Science*, pages 36–48. Springer. doi:10.1007/11428589_3.

- [EMD20] Tolga Erol, Arif Furkan Mendi, and Dilara Doğan. The digital twin revolution in healthcare. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, page 1–7, October 2020. doi:10.1109/ISMSIT50672.2020.9255249.
- [ET] Behin Elahi and Sadegh Amiri Tokaldany. Chapter 15 - application of internet of things-aided simulation and digital twin technology in smart manufacturing. In Mangey Ram, editor, *Advances in Mathematics for Industry 4.0*, pages 335–359. Academic Press. doi:10.1016/B978-0-12-818906-1.00015-2.
- [ETS23] ETSI Specialist Task Forces (STF) 628. SmartM2M: Digital twins and standardization opportunities in ETSI. TS TR 103 844, V1.1.1, European Telecommunications Standards Institute (ETSI), November 2023. URL: https://www.etsi.org/deliver/etsi_tr/103800_103899/103844/01.01.01_60/tr_103844v010101p.pdf.
- [ETS24] ETSI Specialist Task Forces (STF) 628. SmartM2M; digital twins communication requirements. TS TS-103-845-V1.1.1, European Telecommunications Standards Institute (ETSI), February 2024.
- [FBB22] Enxhi Ferko, Alessio Bucaioni, and Moris Behnam. Architecting digital twins. *IEEE Access*, 10:50335–50350, 2022. doi:10.1109/ACCESS.2022.3172964.
- [FFDB20] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971, 2020. doi:10.1109/ACCESS.2020.2998358.
- [FGG⁺22] Hao Feng, Cláudio Gomes, Santiago Gil, Peter H. Mikkelsen, Daniella Tola, Peter Gorm Larsen, and Michael Sandberg. Integration of the Mape-K loop in digital twins. In *2022 Annual*

- Modeling and Simulation Conference (ANNSIM)*, page 102–113, 2022. doi:10.23919/ANNSIM55834.2022.9859489.
- [FGL24] Nils Freyer, Dominik Groß, and Myriam Lipprandt. The ethical requirement of explainability for AI-DSS in healthcare: a systematic review of reasons. *BMC Medical Ethics*, 25(1):104, October 2024. doi:10.1186/s12910-024-01103-2.
- [FGM04] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV*, page 214–230, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-24620-6_15.
- [Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Phd dissertation, University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.
- [Fie08] Roy T. Fielding. REST APIs must be hypertext-driven. Blog post, 2008. Accessed: 2025-10-07. URL: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.
- [FMB⁺25] Mattia Fogli, Gaia Mazziotta, Samuele Burattini, Marco Picone, Carlo Giannelli, and Cesare Stefanelli. Cyber-physical interaction engineering of industrial digital twin ecosystems. In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 420–427, 2025. doi:10.1109/DCOSS-IoT65416.2025.00073.
- [Fou02] Foundation for Intelligent Physical Agents. FIPA Agent Message Transport Protocol for HTTP Specification. Specification SC00084F, FIPA, 2002. Approved Standard, December 3, 2002. URL: <http://www.fipa.org/specs/fipa00084/SC00084F.html>.

- [FS23] Giancarlo Fortino and Claudio Savaglio. *Integration of Digital Twins & Internet of Things*, page 205–225. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-21343-4_8.
- [FSB19] Michael Fairley, David Scheinker, and Margaret L. Brandeau. Improving the efficiency of the operating room environment with an optimization and machine learning model. *Health Care Management Science*, 22(4):756–767, 2019. doi:10.1007/s10729-018-9457-3.
- [fse] L’emilia-romagna prima regione in italia per utilizzo del fascicolo sanitario elettronico. URL: <https://www.auslromagna.it/ausl-comunica/notizie/12019emilia-romagna-prima-regione-in-italia-per-utilizzo-del-fascicolo-sanitario-elettronico>.
- [GBCR24] Andrea Giulianelli, Samuele Burattini, Andrei Ciortea, and Alessandro Ricci. Engineering interoperable ecosystems of digital twins: A web-based approach. In Manuel Wimmer, Alexander Egyed, Benoît Combemale, and Marsha Chechik, editors, *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion 2024, Linz, Austria, September 22-27, 2024*, pages 476–485. ACM, 2024. doi:10.1145/3652620.3688263.
- [GBCR25] Andrea Giulianelli, Samuele Burattini, Andrei Ciortea, and Alessandro Ricci. HWoDT framework: A toolchain to build interoperable digital twin ecosystems. *SoftwareX*, 31:102275, 2025. doi:10.1016/j.softx.2025.102275.
- [GBCR26] Andrea Giulianelli, Samuele Burattini, Andrei Ciortea, and Alessandro Ricci. Hypermedia-based digital twin ecosystems: A proposal for digital twin interoperability. *Future Gener. Comput. Syst.*, 177:108219, 2026. doi:10.1016/J.FUTURE.2025.108219.

- [GBF⁺22] Giancarlo Guizzardi, Alessander Botti Benevides, Claudenir M. Fonseca, Daniele Porello, João Paulo A. Almeida, and Tiago Prince Sales. UFO: unified foundational ontology. *Appl. Ontology*, 17(1):167–210, 2022. doi:10.3233/AO-210256.
- [GC00] James Gillies and R. Cailliau. *How the Web was Born: The Story of the World Wide Web*. Oxford University Press, 2000.
- [GC17] Rosita Guido and Domenico Conforti. A hybrid genetic approach for solving an integrated multi-objective operating room planning and scheduling problem. *Computers & Operations Research*, 87:270–282, November 2017. doi:10.1016/j.cor.2016.11.009.
- [GCB⁺24] Salvador González-Gerpe, Andrea Cimmino, Socorro Bernardos, María Poveda-Villalón, and Raúl García-Castro. WoTDT: an extension of the WoT thing description ontology for digital twins in the construction domain. In Pieter Pauwels, María Poveda-Villalón, and Walter Terkaj, editors, *Proceedings of the 12th Linked Data in Architecture and Construction Workshop, Bochum, Germany, June 13-14, 2024*, volume 3824 of *CEUR Workshop Proceedings*, pages 147–161. CEUR-WS.org, 2024. URL: <https://ceur-ws.org/Vol-3824/paper11.pdf>.
- [Gel91] David Gelernter. *Mirror Worlds: Or: The Day Software Puts the Universe in a Shoebox... How It Will Happen and What It Will Mean*. Oxford University Press, 11 1991. doi:10.1093/oso/9780195068122.001.0001.
- [GGLR17] Rita Gamberini, Luca Galloni, Francesco Lolli, and Bianca Rimini. On the analysis of effectiveness in a manufacturing cell: A critical implementation of existing approaches. *Procedia Manufacturing*, 11:1882–1891, 1 2017. doi:10.1016/J.PROMFG.2017.07.328.
- [Gib14] James J. Gibson. *The Ecological Approach to Visual Perception: Classic Edition*. Psychology Press, New York, November 2014. doi:10.4324/9781315740218.

- [GL86] M.P. Georgeff and A.L. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74(10):1383–1398, October 1986. URL: <https://ieeexplore.ieee.org/abstract/document/1457907>, doi:10.1109/PROC.1986.13639.
- [GL87] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 01 1987.
- [GMGL24a] Santiago Gil, Peter H. Mikkelsen, Cláudio Gomes, and Peter G. Larsen. Survey on open-source digital twin frameworks—a case study approach. *Software: Practice and Experience*, 54(6):929–960, 2024. doi:10.1002/spe.3305.
- [GMGL24b] Santiago Gil, Peter Høgh Mikkelsen, Cláudio Gomes, and Peter Gorm Larsen. Survey on open-source digital twin frameworks—a case study approach. *Softw. Pract. Exp.*, 54(6):929–960, 2024. URL: <https://doi.org/10.1002/spe.3305>, doi:10.1002/SPE.3305.
- [GMP02] Fausto Giunchiglia, John Mylopoulos, and Anna Perini. The tropos software development methodology: processes, models and diagrams. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, AAMAS '02*, page 35–36, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/544741.544748.
- [GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, May 2004.
- [Gov21] Governo Italiano. Piano nazionale di ripresa e resilienza (PNRR). <https://www.governo.it/sites/governo.it/files/PNRR.pdf>, 2021. Accessed: 2025-10-16. URL: <https://www.governo.it/sites/governo.it/files/PNRR.pdf>.
- [GR18] Jorge Gomes and Mário Romão. Information system maturity models in healthcare. *Journal of Medical Systems*, 42(12):235, October 2018. doi:10.1007/s10916-018-1097-0.

- [Gri05] Michael W. Grieves. Product lifecycle management: the new paradigm for enterprises. *International Journal of Product Development*, 2(1/2):71, 2005. doi:10.1504/IJPD.2005.006669.
- [Gri22] Michael Grieves. Intelligent digital twins and the development and management of complex systems. *Digital Twin*, 2:8, May 2022. URL: <https://digitaltwin1.org/articles/2-8/v1>, doi:10.12688/digitaltwin.17574.1.
- [Gri23] Michael W. Grieves. *Digital Twins: Past, Present, and Future*, page 97–121. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-21343-4_4.
- [GS12] Edward Glaessgen and David Stargel. The digital twin paradigm for future NASA and U.S. air force vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. American Institute of Aeronautics and Astronautics, April 2012. doi:10.2514/6.2012-1818.
- [GSFF15] Jorge J. Gómez-Sanz and Rubén Fuentes-Fernández. Understanding agent-oriented software engineering methodologies. *The Knowledge Engineering Review*, 30(4):375–393, 2015. doi:10.1017/S0269888915000053.
- [GT09] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, April 2009.
- [GT16] Dominique Dom Guinard and Vlad M. Trifa. *Building the Web of Things: With examples in Node.js and Raspberry Pi*. Simon and Schuster, 2016.

- [GTW10] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *2010 Internet of Things (IOT)*, page 1–8, November 2010. doi:10.1109/IOT.2010.5678452.
- [Gui11] Dominique Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. PhD thesis, ETH Zurich, Zurich, Switzerland, August 2011.
- [GV17] Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pages 85–113. Springer International Publishing, Cham, 2017.
- [Hau06] Reinhold Haux. Health information systems – past, present, future. *International Journal of Medical Informatics*, 75(3):268–281, March 2006. doi:10.1016/j.ijmedinf.2005.08.002.
- [HB] Wladimir Hofmann and Fredrik Branding. Implementation of an IoT- and cloud-based digital twin for real-time decision support in port operations. 52(13):2104–2109. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019. doi:10.1016/j.ifacol.2019.11.516.
- [HB09] Jomi Fred Hübner and Rafael H. Bordini. Agent-based simulation using BDI programming in jason. In *Multi-Agent Systems - Simulation and Applications*, pages 451–476. 2009. doi:10.1201/9781420070248.CH15.
- [HBC⁺21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge.

- Springer, 2021. URL: <https://kgbook.org/>, doi:10.2200/S01125ED1V01Y202109DSK022.
- [HBRM25a] Jomi Hübner, Samuele Burattini, Alessandro Ricci, and Simon Mayer. Anticipatory thinking in multi-agent contexts. In *Anais do XIX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 1–12, Porto Alegre, RS, Brasil, 2025. SBC. doi:10.5753/wesaac.2025.37525.
- [HBRM25b] Jomi Fred Hübner, Samuele Burattini, Alessandro Ricci, and Simon Mayer. Reflexive anticipatory reasoning by BDI agents. *Auton. Agents Multi Agent Syst.*, 39(1):7, 2025. doi:10.1007/S10458-025-09687-8.
- [HCKS21] Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. RDF-star and SPARQL-star. Final community group report, W3C Community Group, December 2021. URL: <https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html>.
- [HCMM21] Karl Hribernik, Giacomo Cabri, Federica Mandreoli, and Gregoris Mentzas. Autonomous, context-aware, adaptive digital twins—state of the art and roadmap. *Computers in Industry*, 133:103508, December 2021. doi:10.1016/j.compind.2021.103508.
- [Hen01] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, March 2001. URL: <https://ieeexplore.ieee.org/abstract/document/920597>, doi:10.1109/5254.920597.
- [Hen07] James Hendler. Where are all the intelligent agents? *IEEE Intelligent Systems*, 22(3):2–3, May 2007. doi:10.1109/MIS.2007.62.
- [HFZ⁺] Chunhua Hu, Weicun Fan, Elan Zeng, Zhi Hang, Fan Wang, Lianyong Qi, and Md. Zakirul Alam Bhuiyan. Digital

- twin-assisted real-time traffic data prediction method for 5G-enabled internet of vehicles. 18(4):2811–2819. doi:10.1109/TII.2021.3083596.
- [HIM22] Emram criteria, 2022. URL: <https://www.himss.org/sites/hde/files/2022-01/emram-criteria.pdf>.
- [Hin09] Koen V. Hindriks. Programming rational agents in GOAL. In Amal El Fallah Seghrouchni, Jürgen Dix, Mehdi Dastani, and Rafael H. Bordini, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer, Boston, MA, 05 2009. doi:10.1007/978-0-387-89299-3_4.
- [Hin12] Koen V. Hindriks. Debugging is explaining. In Iyad Rahwan, Wayne Wobcke, Sandip Sen, and Toshiharu Sugawara, editors, *PRIMA 2012: Principles and Practice of Multi-Agent Systems*, pages 31–45, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32729-2_3.
- [Hin14] Koen V. Hindriks. The shaping of the agent-oriented mindset - twenty years of engineering MAS. In Fabiano Dalpiaz, Jürgen Dix, and M. Birna van Riemsdijk, editors, *Engineering Multi-Agent Systems - Second International Workshop, EMAS 2014, Paris, France, May 5-6, 2014, Revised Selected Papers*, volume 8758 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2014. doi:10.1007/978-3-319-14484-9_1.
- [Hog20] Aidan Hogan. The semantic web: Two decades on. *Semantic Web*, 11(1):169–185, January 2020. doi:10.3233/SW-190387.
- [Hor51] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, March 1951. doi:10.2307/2268661.
- [HPSB04] Ian Horrocks, Peter F. Patel-Schneider, and Harold Boley. SWRL: A semantic web rule language. Technical report, World Wide Web

- Consortium (W3C), 2004. W3C Member Submission, 21 May 2004. URL: <https://www.w3.org/Submission/SWRL/>.
- [HRC⁺20] Sebastian Hermes, Tobias Riasanow, Eric K. Clemons, Markus Böhm, and Helmut Krcmar. The digital transformation of the healthcare industry: exploring the rise of emerging platform ecosystems and their influence on the role of patients. *Business Research*, 13(3):1033–1069, November 2020. doi:10.1007/s40685-020-00125-x.
- [HS13] Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- [Huh01] Michael N. Huhns. Interaction-oriented programming. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, page 29–44, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-44564-1_2.
- [HvdBM10] Maaike Harbers, Karel van den Bosch, and John-Jules Meyer. Design and evaluation of explainable BDI agents. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 125–132, 2010. doi:10.1109/WI-IAT.2010.115.
- [IAPGB⁺21] Muhammad Intizar Ali, Pankesh Patel, John G. Breslin, Ramy Harik, and Amit Sheth. Cognitive digital twins for smart manufacturing. *IEEE Intelligent Systems*, 36(2):96–100, March 2021. doi:10.1109/MIS.2021.3062437.
- [IBDdRRT⁺12] Pablo Iñigo-Blasco, Fernando Diaz-del Rio, M Carmen Romero-Ternero, Daniel Cagigas-Muñiz, and Saturnino Vicente-Diaz. Robotics software frameworks for multi-agent robotic systems development. *Robotics and Autonomous Systems*, 60(6):803–821, 2012. doi:10.1016/j.robot.2012.02.004.

- [Int10] International Society of Automation. Enterprise-control system integration (ISA-95), 2010.
- [Jen99] Nicholas R. Jennings. Agent-oriented software engineering. In Francisco J. Garijo and Magnus Boman, editors, *Multi-Agent System Engineering*, page 1–7, Berlin, Heidelberg, 1999. Springer. doi:10.1007/3-540-48437-X_1.
- [JOB⁺14] Antonio J. Jara, Alex C. Olivieri, Yann Bocchi, Markus Jung, Wolfgang Kastner, and Antonio F. Skarmeta. Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence. *International Journal of Web and Grid Services*, 10(2–3):244–272, January 2014. doi:10.1504/IJWGS.2014.060260.
- [JSN⁺20] David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, 29:36–52, 2020. doi:10.1016/j.cirpj.2020.02.002.
- [JU20] Michael Jacoby and Thomas Usländer. Digital twin and internet of things—current standards landscape. *Applied Sciences*, 10(18):6519, January 2020. doi:10.3390/app10186519.
- [JWZ] Wenjie Jia, Wei Wang, and Zhenzu Zhang. From simple digital twin to complex digital twin part I: A novel modeling method for multi-scale and multi-scenario digital twin. 53:101706. URL: <https://doi.org/10.1016/j.aei.2022.101706>, doi:10.1016/J.AEI.2022.101706.
- [JYL⁺21] Yuchen Jiang, Shen Yin, Kuan Li, Hao Luo, and Okyay Kaynak. Industrial applications of digital twins. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2207):20200360, August 2021. doi:10.1098/rsta.2020.0360.

- [KAA24] Robert Klar, Niklas Arvidsson, and Vangelis Angelakis. Digital twins' maturity: The need for interoperability. *IEEE Systems Journal*, 18(1):713–724, March 2024. doi:10.1109/JSYST.2023.3340422.
- [KBA⁺22] Konstantinos Kotzias, Faiza A. Bukhsh, Jeewanie Jayasinghe Arachchige, Maya Daneva, and Abhishta Abhishta. Industry 4.0 and healthcare: Context, applications, benefits and challenges. *IET Software*, 17(3):195–248, November 2022. doi:10.1049/sfw2.12074.
- [KBZ21] Maged N. Kamel Boulos and Peng Zhang. Digital twins: From personalised medicine to precision public health. *Journal of Personalized Medicine*, 11(8):745, August 2021. doi:10.3390/jpm11080745.
- [KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003. doi:10.1109/MC.2003.1160055.
- [KC24] Yogeswaranathan Kalyani and Rem Collier. The role of multi-agents in digital twin implementation: Short survey. *ACM Comput. Surv.*, 57(3):72:1–72:15, November 2024. doi:10.1145/3697350.
- [Keh16] Joseph Kehoe. Robust reproducibility of agent based models. In *The European Simulation and Modelling Conference. Inderscience*, 2016.
- [Ken21] John Kendall. *National Digital Twin: Integration Architecture Pattern and Principles*. Centre for Digital Built Britain, 2021. doi:10.17863/CAM.68207.
- [KFLl] Abdallah Karakra, Franck Fontanili, Elyes Lamine, and Jacques Lamothe. HospiT'Win: A predictive simulation-based digital twin

- for patients pathways in hospital. In *2019 IEEE EMBS International Conference on Biomedical & Health Informatics, BHI 2019, Chicago, IL, USA, May 19-22, 2019*, pages 1–4. IEEE. doi:10.1109/BHI.2019.8834534.
- [KG01] K. A. Kuhn and D. A. Giuse. From hospital information systems to health information systems: Problems, challenges, perspectives. *Methods of Information in Medicine*, 40(04):275–287, 2001. doi:10.1055/s-0038-1634170.
- [KK17] Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (SHACL). W3C recommendation, W3C, July 2017. <https://www.w3.org/TR/2017/REC-shacl-20170720/>.
- [KKT⁺18] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihm. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. doi:10.1016/j.ifacol.2018.08.474.
- [KLC20] Gregg Kellogg, Dave Longley, and Pierre-Antoine Champin. JSON-ld 1.1. W3C recommendation, W3C, July 2020. <https://www.w3.org/TR/2020/REC-json-ld11-20200716/>.
- [KN19] Timotheus Kampik and Juan Carlos Nieves. JS-son - A lean, extensible javascript agent programming library. In Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, editors, *Engineering Multi-Agent Systems - 7th International Workshop, EMAS 2019, Montreal, QC, Canada, May 13-14, 2019, Revised Selected Papers*, volume 12058 of *Lecture Notes in Computer Science*, pages 215–234. Springer, 2019. doi:10.1007/978-3-030-51417-4_11.
- [KPGD24] Erkan Karabulut, Salvatore F. Pileggi, Paul Groth, and Victoria Degeler. Ontologies in digital twins: A systematic literature re-

- view. *Future Generation Computer Systems*, 153:442–456, April 2024. doi:10.1016/j.future.2023.12.013.
- [KPZ24] Tim Kreuzer, Panagiotis Papapetrou, and Jelena Zdravkovic. Artificial intelligence in digital twins—a systematic literature review. *Data & Knowledge Engineering*, 151:102304, May 2024. doi:10.1016/j.datak.2024.102304.
- [KSCH25] Gregg Kellogg, Andy Seaborne, Pierre-Antoine Champin, and Olaf Hartig. RDF 1.2 concepts and abstract syntax. W3C working draft, W3C, February 2025. <https://www.w3.org/TR/2025/WD-rdf12-concepts-20250225/>.
- [KSI23] Krešimir Kušić, René Schumann, and Edouard Ivanjko. A digital twin in transportation: Real-time synergy of traffic data streams and simulation for virtualizing motorway dynamics. *Advanced Engineering Informatics*, 55:101858, 2023. doi:10.1016/j.aei.2022.101858.
- [KSN00] Mamadou Tadiou Kone, Akira Shimazu, and Tatsuo Nakajima. The state of the art in agent communication languages. *Knowledge and Information Systems*, 2(3):259–284, August 2000. doi:10.1007/PL00013712.
- [KSPI21] Sascha Kraus, Francesco Schiavone, Anna Pluzhnikova, and Anna Chiara Invernizzi. Digital transformation in healthcare: Analyzing the current state-of-research. *Journal of Business Research*, 123:557–567, February 2021. doi:10.1016/j.jbusres.2020.10.030.
- [KWH13] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Final report of the industrie 4.0 working group. Technical report, acatech – National Academy of Science and Engineering, Munich, Germany, 2013. URL:

<https://en.acatech.de/publication/recommendations-for-implementing-the-strategic-initiative-industrie-4-0/>.

- [KWW⁺24] Evangelia Katsoulakis, Qi Wang, Huanmei Wu, Leili Shahriyari, Richard Fletcher, Jinwei Liu, Luke Achenie, Hongfang Liu, Pamela Jackson, Ying Xiao, Tanveer Syeda-Mahmood, Richard Tuli, and Jun Deng. Digital twins for health: a scoping review. *npj Digital Medicine*, 7(1):77, March 2024. doi:10.1038/s41746-024-01073-0.
- [LBMC23] Jérémy Lemée, Samuele Burattini, Simon Mayer, and Andrei Ciortea. Domain-expert configuration of hypermedia multi-agent systems in industrial use cases. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 2499–2501. ACM, 2023. doi:10.5555/3545946.3598981.
- [LFK⁺14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, August 2014. doi:10.1007/s12599-014-0334-4.
- [LFW⁺16] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzson, Jörg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, and Andrey Sadovykh. Integrated tool chain for model-based design of cyber-physical systems: The into-cps project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6, 2016. doi:10.1109/CPSData.2016.7496424.
- [Lin00] Jürgen Lind. Issues in agent-oriented software engineering. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering, First International Workshop*,

- AOSE 2000, Limerick, Ireland, June 10, 2000, Revised Papers*, volume 1957 of *Lecture Notes in Computer Science*, pages 45–58. Springer, 2000. doi:10.1007/3-540-44564-1_3.
- [LJJ20] Chao Liu, Pingyu Jiang, and Wenlei Jiang. Web-based digital twin modeling and remote control of cyber-physical production systems. *Robotics and Computer-Integrated Manufacturing*, 64:101956, August 2020. doi:10.1016/j.rcim.2020.101956.
- [LKR⁺16] Paulo Leitão, Stamatis Karnouskos, Luis Ribeiro, Jay Lee, Thomas I. Strasser, and Armando W. Colombo. Smart agents in industrial cyber-physical systems. *Proc. IEEE*, 104(5):1086–1101, 2016. doi:10.1109/JPR0C.2016.2521931.
- [LLC17] Ying Liu, Lei Liu, and Wei-Peng Chen. Intelligent traffic light control using distributed multi-agent Q learning. In *20th IEEE International Conference on Intelligent Transportation Systems, ITSC 2017, Yokohama, Japan, October 16-19, 2017*, pages 1–8. IEEE, 2017. doi:10.1109/ITSC.2017.8317730.
- [LLS21] Carmelo Fabio Longo, Francesco Longo, and Corrado Santoro. Caspar: Towards decision making helpers agents for IoT, based on natural language and first order logic reasoning. *Engineering Applications of Artificial Intelligence*, 104:104269, 2021. doi:10.1016/j.engappai.2021.104269.
- [LMPZ23] Marco Lippi, Matteo Martinelli, Marco Picone, and Franco Zambonelli. Enabling causality learning in smart factories with hierarchical digital twins. *Computers in Industry*, 148:103892, 2023. doi:10.1016/j.compind.2023.103892.
- [LMS⁺21] Ye Liu, Xiaoyuan Ma, Lei Shu, Gerhard Petrus Hancke, and Adnan M. Abu-Mahfouz. From industry 4.0 to agriculture 4.0: Current status, enabling technologies, and research challenges. *IEEE Transactions on Industrial Informatics*, 17(6):4322–4334, 2021. doi:10.1109/TII.2020.3003910.

- [LN16] Samreen Laghari and Muaz A. Niazi. Modeling the internet of things, self-organizing and other complex adaptive communication networks: A cognitive agent-based computing approach. *PLOS ONE*, 11(1):1–26, 01 2016. doi:10.1371/journal.pone.0146760.
- [Log18] Brian Logan. An agent programming manifesto. *Int. J. Agent Oriented Softw. Eng.*, 6(2):187–210, 2018. doi:10.1504/IJA0SE.2018.10015555.
- [LPM⁺24] Gianfranco Lombardo, Marco Picone, Marco Mamei, Monica Mordonini, and Agostino Poggi. Digital twin for continual learning in location based services. *Engineering Applications of Artificial Intelligence*, 127:107203, 2024. doi:10.1016/j.engappai.2023.107203.
- [LPT⁺22] Daniel Lehner, Jérôme Pfeiffer, Erik-Felix Tinsel, Matthias Milan Strljic, Sabine Sint, Michael Vierhauser, Andreas Wortmann, and Manuel Wimmer. Digital twin platforms: Requirements, capabilities, and future prospects. *IEEE Software*, 39(2):53–61, 2022. doi:10.1109/MS.2021.3133795.
- [LSC20] Francesco Lanza, Valeria Seidita, and Antonio Chella. Agents and robots for collaborating and supporting physicians in healthcare scenarios. *Journal of Biomedical Informatics*, 108:103483, 2020. doi:10.1016/j.jbi.2020.103483.
- [LTZ⁺20] Lei Lei, Yue Tan, Kan Zheng, Shiwen Liu, Kuan Zhang, and Xuemin Shen. Deep reinforcement learning for autonomous internet of things: Model, applications and challenges. *IEEE Commun. Surv. Tutorials*, 22(3):1722–1760, 2020. doi:10.1109/COMST.2020.2988367.
- [luo20] Machine learning for identification of surgeries with high risks of cancellation. *Health Informatics J.*, 26(1), 2020. doi:10.1177/1460458218813602.

- [L621] Carlos Eduardo Belman López. Real-time event-based platform for the development of digital twin applications. *The International Journal of Advanced Manufacturing Technology*, 116(3–4):835–845, June 2021. doi:10.1007/s00170-021-07490-9.
- [Mat24] MathWorks. Optimize and automate energy assets with digital twins in MATLAB and Simulink. White paper, 2024. Accessed: 2025-09-23. URL: <https://www.mathworks.com/content/dam/mathworks/white-paper/digital-twins-energy-white-paper.pdf>.
- [MBM⁺25] Matteo Martinelli, Antonello Barbone, Riccardo Morandi, Marco Picone, Samuele Burattini, and Alessandro Ricci. Modular engineering of industrial digital twins: The wldt approach. In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 451–458, 2025. doi:10.1109/DCOSS-IoT65416.2025.00077.
- [MCFA23] Roberto Minerva, Noel Crespi, Reza Farahbakhsh, and Faraz M. Awan. *Artificial Intelligence and the Digital Twin: An Essential Combination*, page 299–336. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-21343-4_12.
- [MCR⁺19] Sara Montagna, Angelo Croatti, Alessandro Ricci, Vanni Agnoletti, Vittorio Albarello, and Emiliano Gamberini. Real-time tracking and documentation in trauma management. *Health Informatics Journal*, 26(1):328–341, February 2019. doi:10.1177/1460458219825507.
- [Mei22] Alexander Meijers. *Hands-On Azure Digital Twins: A practical guide to building distributed IoT solutions*. Packt Publishing Ltd, March 2022. Google-Books-ID: QGpEEAAQBAJ.
- [MET⁺14] Ghezlane Halhoul Merabet, Mohammed Essaaidi, Hanaa Talei, Mohamed Riduan Abid, Nacer Khalil, Mohcine Madkour, and

- Driss Benhaddou. Applications of multi-agent systems in smart grids: A survey. In *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, page 1088–1094, April 2014. doi:10.1109/ICMCS.2014.6911384.
- [MF14] Jörg P. Müller and Klaus Fischer. Application impact of multi-agent systems and technologies: A survey. In Onn Shehory and Arnon Sturm, editors, *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*, pages 27–53. Springer, 2014. doi:10.1007/978-3-642-54432-3_3.
- [MHDP⁺22] Ali McBride, Sarah Hudson-DiSalle, Jeff Pilz, Mark Hamm, Brooke Boring, Larry W. Buie, and David L. DeRemer. National survey on the effect of oncology drug shortages in clinical practice: A hematology oncology pharmacy association survey. *JCO Oncology Practice*, 18(8):e1289–e1296, August 2022. doi:10.1200/op.21.00883.
- [MJW22] Manuel S. Müller, Nasser Jazdi, and Michael Weyrich. Self-improving models for the intelligent digital twin: Towards closing the reality-to-simulation gap. *IFAC-PapersOnLine*, 55(2):126–131, 2022. 14th IFAC Workshop on Intelligent Manufacturing Systems IMS 2022. doi:10.1016/j.ifacol.2022.04.181.
- [MKK23] Michael McCool, Ege Korkan, and Sebastian Käbisch. Web of things (wot) thing description 1.1. W3C recommendation, W3C, December 2023. <https://www.w3.org/TR/2023/REC-wot-thing-description11-20231205/>.
- [ML07] Peter McBurney and Michael Luck. The agents are all busy doing stuff! *IEEE Intelligent Systems*, 22(4):6–7, 2007. doi:10.1109/MIS.2007.77.
- [MLC20] Roberto Minerva, Gyu Myoung Lee, and Noel Crespi. Digital twin in the IoT context: A survey on technical features, scenarios, and

- architectural models. *Proceedings of the IEEE*, 108(10):1785–1824, October 2020. doi:10.1109/jproc.2020.2998530.
- [MLD09] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.
- [MMB⁺20] Jessica Morley, Caio C. V. Machado, Christopher Burr, Josh Cows, Indra Joshi, Mariarosaria Taddeo, and Luciano Floridi. The ethics of AI in health care: A mapping review. *Social Science & Medicine*, 260:113172, 2020. doi:10.1016/j.socscimed.2020.113172.
- [MPR22] Stefano Mariani, Marco Picone, and Alessandro Ricci. About digital twins, agents, and multiagent systems: A cross-fertilisation journey. In Francisco S. Melo and Fei Fang, editors, *Autonomous Agents and Multiagent Systems. Best and Visionary Papers*, page 114–129, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-20179-0_8.
- [MPR23] Stefano Mariani, Marco Picone, and Alessandro Ricci. Towards developing digital twin enabled multi-agent systems. In Andrei Ciortea, Mehdi Dastani, and Jieting Luo, editors, *Engineering Multi-Agent Systems*, page 178–187, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-48539-8_12.
- [MSA15] Ashok Malhotra, Steve Speicher, and John Arwe. Linked data platform 1.0. W3C recommendation, W3C, February 2015. <https://www.w3.org/TR/2015/REC-ldp-20150226/>.
- [MSPM⁺19] Somayeh Malakuti, Johannes Schmitt, Marie Platenius-Mohr, Sten Grüner, Ralf Gitzel, and Prerna Bihani. A four-layer architecture pattern for constructing and managing digital twins. In Tomas Bures, Laurence Duchien, and Paola Inverardi, editors, *Software Architecture*, page 231–246, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-29983-5_16.

- [MSW24] Judith Michael, Maike Schwammberger, and Andreas Wortmann. Explaining cyberphysical system behavior with digital twins. *IEEE Softw.*, 41(1):55–63, 2024. doi:10.1109/MS.2023.3319580.
- [MvdH] John-Jules Charles Meyer and Wiebe van der Hoek. *Epistemic logic for AI and computer science*, volume 41 of *Cambridge tracts in theoretical computer science*. Cambridge University Press.
- [MWD19] Mainak Majumder, Lukasz Wisniewski, and Christian Diedrich. A comparison of OPC UA & semantic web languages for the purpose of industrial automation applications. In *24th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2019, Zaragoza, Spain, September 10-13, 2019*, pages 1297–1300. IEEE, 2019. doi:10.1109/ETFA.2019.8869113.
- [MWR19] Viviana Mascardi, Danny Weyns, and Alessandro Ricci. Engineering multi-agent systems: State of affairs and the road ahead. *ACM SIGSOFT Softw. Eng. Notes*, 44(1):18–28, 2019. doi:10.1145/3310013.3310035.
- [MYH⁺22] Stefan Mihai, Mahnoor Yaqoob, Dang V. Hung, William Davis, Praveer Towakel, Mohsin Raza, Mehmet Karamanoglu, Balbir Barn, Dattaprasad Shetve, Raja V. Prasad, Hrishikesh Venkataraman, Ramona Trestian, and Huan X. Nguyen. Digital twins: A survey on enabling technologies, challenges, trends and future prospects. *IEEE Communications Surveys & Tutorials*, 24(4):2255–2291, 2022. doi:10.1109/COMST.2022.3208773.
- [Nak95] Seiichi Nakajima. *Introduction to TPM: total productive maintenance*. Productivity Press, 1995.
- [ND06] Peter Novák and Jürgen Dix. Modular BDI architecture. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1009–1015. ACM, 2006. doi:10.1145/1160633.1160814.

- [Nel67] Theodor H. Nelson. Getting it out of our system. In *Information Retrieval: A Critical Review*, pages 191–210. Thompson Books, Washington, D.C., 1967.
- [New02] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional, 2002.
- [NHA⁺21] Natalia Norori, Qiyang Hu, Florence Marcelle Aellen, Francesca Dalia Faraci, and Athina Tzovara. Addressing bias in big data and AI for health care: A call for open science. *Patterns*, 2(10), October 2021. doi:10.1016/j.patter.2021.100347.
- [NL18] Narissa J. Nonzee and Thanh Ha Luu. *The Drug Shortage Crisis in the United States: Impact on Cancer Pharmaceutical Safety*, page 75–92. Springer International Publishing, December 2018. doi:10.1007/978-3-319-43896-2_6.
- [Nor08] Donald A. Norman. The way i see it: Signifiers, not affordances. *Interactions*, 15(6):18–19, November 2008. doi:10.1145/1409040.1409044.
- [Nor13] Donald A. Norman. *The Design of Everyday Things*. MIT Press, 2013.
- [OAS19] OASIS Standard. MQTT version 5.0. Standard, OASIS, March 2019. OASIS Standard, 7 March 2019. URL: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.
- [OC19] Andrea Omicini and Roberta Calegari. Injecting (micro)intelligence in the IoT: Logic-based approaches for (M)MAS. In Donghui Lin, Toru Ishida, Franco Zambonelli, and Itsuki Noda, editors, *Massively Multi-Agent Systems II*, page 21–35, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-20937-7_2.

- [olw12] OWL 2 web ontology language document overview (second edition). Technical report, W3C, December 2012. <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/>.
- [ope21] openEHR Foundation. openEHR architecture overview. Technical Specification BASE Release 1.2.0, openEHR Foundation, 2021. Status: STABLE. © 2003–2021 openEHR Foundation. Accessed: 2025-10-14. URL: https://specifications.openehr.org/releases/BASE/latest/architecture_overview.html.
- [ORV08] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, December 2008. doi:10.1007/s10458-008-9053-x.
- [PBB20] Marina Papalexi, David Bamford, and Liz Breen. Key sources of operational inefficiency in the pharmaceutical supply chain. *Supply Chain Management: An International Journal*, 25(6):617–635, 2023/11/20 2020. doi:10.1108/SCM-02-2019-0076.
- [PBBC24] Danilo Pianini, Martina Baiardi, Samuele Burattini, and Giovanni Ciatto. Multi-paradigm integration for the BDI resurgence. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2024 - Companion, Aarhus, Denmark, September 16-20, 2024*, pages 27–28. IEEE, 2024. doi:10.1109/ACSOS-C63493.2024.00025.
- [PBL05] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 149–174. Springer, 2005. doi:10.1007/0-387-26350-0_6.

- [PG18] Aaron Parecki and Julien Genestoux. WebSub. W3C recommendation, W3C, January 2018. URL: <https://www.w3.org/TR/2018/REC-websub-20180123/>.
- [PGS03] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with INGENIAS. In Vladimír Mařík, Michal Pěchouček, and Jörg Müller, editors, *Multi-Agent Systems and Applications III*, page 394–403, Berlin, Heidelberg, 2003. Springer. doi:10.1007/3-540-45023-8_38.
- [PH] Pablo Pico-Valencia and Juan A. Holgado-Terriza. Agentification of the internet of things: A systematic literature review. 14(10). doi:10.1177/1550147718805945.
- [Pir21] Vartan Piroumian. Digital twins: Universal interoperability for the digital age. *Computer*, 54(1):61–69, 2021. doi:10.1109/MC.2020.3032148.
- [PJGBF⁺23] Miguel Pedrera-Jiménez, Noelia García-Barrio, Santiago Frid, David Moner, Diego Boscá-Tomás, Raimundo Lozano-Rubí, Dipak Kalra, Thomas Beale, Adolfo Muñoz-Carrero, and Pablo Serrano-Balazote. Can OpenEHR, ISO 13606, and HL7 FHIR work together? An agnostic approach for the selection and application of electronic health record standards to the next-generation health data spaces. *Journal of Medical Internet Research*, 25(1):e48702, December 2023. Company: Journal of Medical Internet Research Distributor: Journal of Medical Internet Research Institution: Journal of Medical Internet Research Label: Journal of Medical Internet Research publisher: JMIR Publications Inc., Toronto, Canada. doi:10.2196/48702.
- [PK21] Paolo Pareti and George Konstantinidis. A review of SHACL: from data validation to schema reasoning for RDF graphs. In Mantas Simkus and Ivan Varzinczak, editors, *Reasoning Web. Declarative Artificial Intelligence - 17th International Summer School*

- 2021, Leuven, Belgium, September 8-15, 2021, Tutorial Lectures, volume 13100 of *Lecture Notes in Computer Science*, pages 115–144. Springer, 2021. doi:10.1007/978-3-030-95481-9_6.
- [Pla19] Plattform Industrie 4.0 Working Group “Asset Administration Shell”. Details of the asset administration shell – part 1: The exchange of information between partners in the value chain of industrie 4.0 (version 2.0). Technical report, Plattform Industrie 4.0, November 2019. Version 2.0, produced August 2018–November 2019. URL: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V2.html.
- [PLR⁺25] Eric G Poon, Christy Harris Lemak, Juan C Rojas, Janet Gup-till, and David Classen. Adoption of artificial intelligence in healthcare: survey of health system priorities, successes, and chal-lenges. *Journal of the American Medical Informatics Association*, 32(7):1093–1100, 2025. doi:10.1093/jamia/ocaf065.
- [PLT20] Rashik Parmar, Aija Leiponen, and Llewellyn D. W. Thomas. Building an organizational digital twin. *Busi-ness Horizons*, 63(6):725–736, November 2020. doi:10.1016/j.bushor.2020.08.001.
- [PLW⁺24] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Trans. Knowl. Data Eng.*, 36(7):3580–3599, 2024. doi:10.1109/TKDE.2024.3352100.
- [PMB⁺25a] Marco Picone, Matteo Martinelli, Samuele Burattini, Andrea Giu-lianelli, and Alessandro Ricci. The two faces of interoperability: Bridging cyber and physical spaces with digital twins. In *2025 21st International Conference on Distributed Computing in Smart Sys-tems and the Internet of Things (DCOSS-IoT)*, pages 1–8, 2025. doi:10.1109/DCOSS-IoT65416.2025.00078.

- [PMB⁺25b] Marco Picone, Riccardo Morandi, Antonello Barbone, Samuele Burattini, Mattia Fogli, Nicola Bicocchi, Carlo Giannelli, and Alessandro Ricci. Harmonizing physical and digital twins life-cycles. In *22nd IEEE International Conference on Software Architecture, ICSA - Companion, Odense, Denmark, March 31 - April 4, 2025*, pages 197–204. IEEE, 2025. doi:10.1109/ICSA-C65153.2025.00039.
- [PMN⁺24] Elena Pretel, Alejandro Moya, Elena Navarro, Víctor López-Jaquero, and Pascual González. Analysing the synergies between multi-agent systems and digital twins: A systematic literature review. *Information and Software Technology*, 174:107503, October 2024. doi:10.1016/j.infsof.2024.107503.
- [PMV13] Danilo Pianini, Sara Montagna, and Mirko Viroli. Chemical-oriented simulation of computational systems with ALCHEMIST. *J. Simulation*, 7(3):202–215, 2013. doi:10.1057/jos.2012.27.
- [PMZ21] Marco Picone, Marco Mamei, and Franco Zambonelli. WLDT: A general purpose library to build iot digital twins. *SoftwareX*, 13:100661, 2021. doi:10.1016/j.softx.2021.100661.
- [PPRK19] Alexander Perzylo, Stefan Profanter, Markus Rickert, and Alois C. Knoll. OPC UA nodeset ontologies as a pillar of representing semantic digital twins of manufacturing resources. In *24th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2019, Zaragoza, Spain, September 10-13, 2019*, pages 1085–1092. IEEE, 2019. doi:10.1109/ETFA.2019.8868954.
- [PRC⁺22] Javier Palanca, Jaime Andres Rincon, Carlos Carrascosa, Vicente Julián, and Andrés Terrasa. A flexible agent architecture in SPADE. In Frank Dignum, Philippe Mathieu, Juan Manuel Corchado, and Fernando de la Prieta, editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex*

- Systems Simulation. The PAAMS Collection - 20th International Conference, PAAMS*, volume 13616 of *Lecture Notes in Computer Science*, pages 320–331, L'Aquila, Italy, 10 2022. Springer. doi:10.1007/978-3-031-18192-4_26.
- [PW02] Lin Padgham and Michael Winikoff. Prometheus: a methodology for developing intelligent agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, page 37–38, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/544741.544749.
- [PZVN⁺25] Elena Pretel, Luis Zhinin-Vera, Elena Navarro, Víctor López-Jaquero, and Pascual González. MAS4DT: A novel proposal for developing digital twins following a multi-agent system approach. *Journal of Systems and Software*, 222:112344, April 2025. doi:10.1016/j.jss.2025.112344.
- [PZY⁺20] Yang Peng, Ming Zhang, Fangqiang Yu, Jinglin Xu, and Shang Gao. Digital twin hospital buildings: An exemplary case study through continuous lifecycle integration. *Advances in Civil Engineering*, 2020(1):8846667, 2020. doi:10.1155/2020/8846667.
- [QGZ⁺] Weiwei Qian, Yu Guo, Hao Zhang, Shaohua Huang, Litong Zhang, Hailang Zhou, Weiguang Fang, and Shanshan Zha. Digital twin driven production progress prediction for discrete manufacturing workshop. 80:102456. doi:10.1016/J.RCIM.2022.102456.
- [QL08] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in Com-*

- puter Science*, pages 524–538. Springer, 2008. doi:10.1007/978-3-540-68234-9_39.
- [QLZL24] Yuchong Qian, Jiawei Leng, Kai Zhou, and Yuxuan Liu. How to measure and control indoor air quality based on intelligent digital twin platforms: A case study in china. *Building and Environment*, 253:111349, 2024. doi:10.1016/j.buildenv.2024.111349.
- [QT18] Qinglin Qi and Fei Tao. Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. *IEEE Access*, 6:3585–3593, 2018. doi:10.1109/ACCESS.2018.2793265.
- [QTH⁺21] Qinglin Qi, Fei Tao, Tianliang Hu, Nabil Anwer, Ang Liu, Yongli Wei, Lihui Wang, and A.Y.C. Nee. Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*, 58:3–21, 2021. Digital Twin towards Smart Manufacturing and Industry 4.0. doi:10.1016/j.jmsy.2019.10.001.
- [QWYG] Qianzhe Qiao, Jinjiang Wang, Lunkuan Ye, and Robert X. Gao. Digital twin for machining tool condition prediction. 81:1388–1393. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019. doi:10.1016/j.procir.2019.04.049.
- [RAA⁺18] Luca Roffia, Paolo Azzoni, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, and Tullio Salmon Cinotti. Dynamic linked data: A SPARQL event processing architecture. *Future Internet*, 10(4):36, 2018. URL: <https://doi.org/10.3390/fi10040036>, doi:10.3390/FI10040036.
- [Rao96] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John W. Perram, editors, *Agents Breaking Away*, page 42–55, Berlin, Heidelberg, 1996. Springer. doi:10.1007/BFb0031845.
- [RBCC24] Alessandro Ricci, Samuele Burattini, Andrei Ciortea, and Matteo Castellucci. Agents for DDD - back and forth. In Daniela

- Briola, Rafael C. Cardoso, and Brian Logan, editors, *Engineering Multi-Agent Systems - 12th International Workshop, EMAS 2024, Auckland, New Zealand, May 6-7, 2024, Revised Selected Papers*, volume 15152 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2024. doi:10.1007/978-3-031-71152-7_11.
- [RCB⁺20] Alessandro Ricci, Angelo Croatti, Rafael H. Bordini, Jomi Fred Hübner, and Olivier Boissier. Exploiting Simulation for MAS Development and Execution - The JaCaMo-Sim Approach. In *Engineering Multi-Agent Systems - 8th International Workshop, EMAS 2020, Revised Selected Papers*, volume 12589, pages 42–60, Auckland, New Zealand, May 2020. Springer. doi:10.1007/978-3-030-66534-0_3.
- [RCB25] Alessandro Ricci, Andrei Ciortea, and Samuele Burattini. Extending CArtaGO to support the engineering of complex distributed environments at the knowledge level. *Accepted*, 2025. URL: <https://euramas.github.io/eumas2025/#EMAS%201>.
- [RCM⁺22a] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. Web of digital twins. *ACM Trans. Internet Technol.*, 22(4), nov 2022. doi:10.1145/3507909.
- [RCM22b] Alessandro Ricci, Angelo Croatti, and Sara Montagna. Pervasive and connected digital twins—a vision for digital health. *IEEE Internet Computing*, 26(5):26–32, 2022. doi:10.1109/MIC.2021.3052039.
- [RCVA22] Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. *Journal of Parallel and Distributed Computing*, 166:71–94, August 2022. doi:10.1016/j.jpdc.2022.04.004.
- [RG91] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International*

Conference on Principles of Knowledge Representation and Reasoning (KR'91), pages 473–484, 1991.

- [RGG14] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. SARL: a general-purpose agent-oriented programming language. In *the 2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Warsaw, Poland, 2014. IEEE Computer Society Press.
- [Ric22] Adriana Riccomagno. Sanità. una regione digitale: l’Emilia-Romagna, 2022. URL: <https://trendsanita.it/sanita-digitale-romagna/>.
- [RM24] João Luiz Rebelo Moreira. The role of interoperability for digital twins. In Tiago Prince Sales, Sybren de Kinderen, Henderik A. Proper, Luise Pufahl, Dimka Karastoyanova, and Marten van Sinderen, editors, *Enterprise Design, Operations, and Computing. EDOC 2023 Workshops*, page 139–157, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-54712-6_9.
- [RMD23] Julia Robles, Cristian Martín, and Manuel Díaz. Opentwins: An open-source framework for the development of next-gen compositional digital twins. *Comput. Ind.*, 152:104007, 2023. URL: <https://doi.org/10.1016/j.compind.2023.104007>, doi:10.1016/J.COMPIND.2023.104007.
- [RMZ⁺24] Alessandro Ricci, Stefano Mariani, Franco Zambonelli, Samuele Burattini, and Cristiano Castelfranchi. The cognitive hourglass: Agent abstractions in the large models era. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, pages 2706–2711. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024. doi:10.5555/3635637.3663262.

- [RN95] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995.
- [ROV⁺07] Alessandro Ricci, Andrea Omicini, Mirko Viroli, Luca Gardelli, and Enrico Oliva. *Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts*, volume 4389 of *Lecture Notes in Computer Science*, page 124–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-71103-2_7.
- [RPVO09] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. *Environment Programming in CArtaGO*, page 259–288. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-89299-3_8.
- [RTW23] Sebastian Rodriguez, John Thangarajah, and Michael Winikoff. A behaviour-driven approach for testing requirements via user and system stories in agent systems. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 1182–1190. ACM, 2023. doi:10.5555/3545946.3598761.
- [RUR⁺22] Julia Rosenberger, Michael Urlaub, Felix Rauterberg, Tina Lutz, Andreas Selig, Michael Bühren, and Dieter Schramm. Deep reinforcement learning multi-agent system for resource allocation in industrial internet of things. *Sensors*, 22(11), 2022. doi:10.3390/s22114099.
- [RVG⁺22] Ganesh Ramanathan, Danai Vachtsevanou, Kimberly García, Jérémy Lemée, Samuele Burattini, Kenan Bektas, and Simon Mayer. Semantic knowledge for autonomous smart farming. *IFAC-PapersOnLine*, 55(32):217–222, 2022. 7th IFAC Conference

- on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2022. doi:10.1016/j.ifacol.2022.11.142.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, 1998.
- [SBLH06] N. Shadbolt, T. Berners-Lee, and W. Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, January 2006. doi:10.1109/MIS.2006.62.
- [SC08] Leo Sauermann and Richard Cyganiak. Cool URIs for the semantic web. W3C note, W3C, December 2008. <https://www.w3.org/TR/2008/NOTE-cooluris-20081203/>.
- [SC17] Munindar P. Singh and Amit K. Chopra. The internet of things and multiagent systems: Decentralized intelligence in distributed computing. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, page 1738–1747, 2017. doi:10.1109/ICDCS.2017.304.
- [SCS⁺21] Silvana Secinaro, Davide Calandra, Aurelio Secinaro, Vivek Muthurangu, and Paolo Biancone. The role of artificial intelligence in healthcare: a structured literature review. *BMC Medical Informatics and Decision Making*, 21(1):125, April 2021. doi:10.1186/s12911-021-01488-9.
- [SFH⁺21] Maulshree Singh, Evert Fuenmayor, Eoin P. Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2):36, 2021. doi:10.3390/asi4020036.
- [SFZ16] Claudio Savaglio, Giancarlo Fortino, and Mengchu Zhou. Towards interoperable, cognitive and autonomic iot systems: An agent-based approach. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, page 58–63, December 2016. doi:10.1109/WF-IoT.2016.7845459.

- [SGM⁺22] Luca Sciullo, Lorenzo Gigli, Federico Montori, Angelo Trotta, and Marco Di Felice. A survey on the web of things. *IEEE Access*, 10:47570–47596, 2022. doi:10.1109/ACCESS.2022.3171575.
- [SGP⁺20] Claudio Savaglio, Maria Ganzha, Marcin Paprzycki, Costin Bădică, Mirjana Ivanović, and Giancarlo Fortino. Agent-based internet of things: State-of-the-art and research challenges. *Future Generation Computer Systems*, 102:1038–1053, 2020. doi:10.1016/j.future.2019.09.016.
- [SGS⁺24] Siva Sai, Aanchal Gaur, Revant Sai, Vinay Chamola, Mohsen Guizani, and Joel J. P. C. Rodrigues. Generative AI for transformative healthcare: A comprehensive study of emerging models, applications, case studies, and limitations. *IEEE Access*, 12:31078–31106, 2024. doi:10.1109/ACCESS.2024.3367715.
- [SH04] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, November 2004. doi:10.1002/0470091509.
- [Sha04] Nilay Shah. Pharmaceutical supply chains: key issues and strategies for optimisation. *Computers & Chemical Engineering*, 28(6):929–941, 2004. FOCAPO 2003 Special issue. doi:10.1016/j.compchemeng.2003.09.022.
- [SHB14] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (CoAP). RFC 7252, RFC Editor, June 2014.
- [She21] Wei Shengli. Is human digital twin possible? *Computer Methods and Programs in Biomedicine Update*, 1:100014, January 2021. doi:10.1016/j.cmpbup.2021.100014.
- [SHH⁺11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on linked data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal,

- Natasha Fridman Noy, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 601–616. Springer, 2011. doi:10.1007/978-3-642-25073-6_38.
- [Sho93] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, March 1993. doi:10.1016/0004-3702(93)90034-9.
- [SKG⁺21] Jan Schoenfelder, Sebastian Kohl, Manuel Glaser, Sebastian McRae, Jens O. Brunner, and Thomas Koperna. Simulation-based evaluation of operating room management policies. *BMC Health Services Research*, 21(1), March 2021. doi:10.1186/s12913-021-06234-5.
- [SKS08] Ilias Sakellariou, Petros Kefalas, and Ioanna Stamatopoulou. Enhancing NetLogo to simulate BDI communicating agents. In John Darzentas, George A. Vouros, Spyros Vosinakis, and Argyris Arnellos, editors, *Artificial Intelligence: Theories, Models and Applications*, page 263–275, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-87881-0_24.
- [SLPD21] Concetta Semeraro, Mario Lezoche, Hervé Panetto, and Michele Dassisti. Digital twin paradigm: A systematic literature review. *Computers in Industry*, 130:103469, 2021. doi:10.1016/j.compind.2021.103469.
- [SLV⁺19] Jack Sleuters, Yonghui Li, Jacques Verriet, Marina Velikova, and Richard Doornbos. A digital twin method for automated behavior analysis of large-scale distributed iot systems. In *2019 14th Annual Conference System of Systems Engineering (SoSE)*, pages 7–12, 2019. doi:10.1109/SYSOSE.2019.8753845.
- [SMC13] Stefan Schulz and Catalina Martínez-Costa. How ontologies can improve semantic interoperability in health care. In David Ri-

- año, Richard Lenz, Silvia Miksch, Mor Peleg, Manfred Reichert, and Annette ten Teije, editors, *Process Support and Knowledge Representation in Health Care*, page 1–10, Cham, 2013. Springer International Publishing. doi:10.1007/978-3-319-03916-9_1.
- [SPL16] Dharendra Singh, Lin Padgham, and Brian Logan. Integrating BDI Agents with Agent-Based Simulation Platforms. *Autonomous Agents and Multi-Agent Systems*, 30(6):1050–1071, November 2016. doi:10.1007/s10458-016-9332-x.
- [Sta21] Christian Stary. Digital twin generation: Re-conceptualizing agent systems for behavior-centered cyber-physical system development. *Sensors*, 21(4):1096, January 2021. doi:10.3390/s21041096.
- [STM⁺22] Luca Sciallo, Angelo Trotta, Federico Montori, Luciano Bononi, and Marco Di Felice. Wotwins: Automatic digital twin generator for the web of things. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, page 607–612, 2022. doi:10.1109/WoWMoM54355.2022.00095.
- [SW18] Victor Singh and K. E. Willcox. Engineering design with digital thread. *AIAA Journal*, 56(11):4515–4528, 2018. doi:10.2514/1.J057255.
- [TBC⁺16] Patrick Taillandier, Mathieu Bourgeois, Philippe Caillou, Carole Adam, and Benoit Gaudou. A BDI agent architecture for the GAMA modeling and simulation platform. In Luis G. Nardin and Luis Antunes, editors, *Multi-Agent Based Simulation XVII - International Workshop, MABS 2016, Revised Selected Papers*, volume 10399 of *Lecture Notes in Computer Science*, pages 3–23, Singapore, May 2016. Springer. doi:10.1007/978-3-319-67477-3_1.

- [TCQ⁺18] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94(9–12):3563–3576, February 2018. doi:10.1007/s00170-017-0233-1.
- [TFMCV⁺20] Guilherme Luz Tortorella, Flávio Sanson Fogliatto, Alejandro Mac Cawley Vergara, Roberto Vassolo, and Rapinder Sawhney. Healthcare 4.0: trends, challenges and research directions. *Production Planning & Control*, 31(15):1245–1260, November 2020. doi:10.1080/09537287.2019.1702226.
- [TGM⁺23] Prasad Talasila, Cláudio Gomes, Peter Høgh Mikkelsen, Santiago Gil Arboleda, Eduard Kamburjan, and Peter Gorm Larsen. Digital twin as a service (DTaaS): A platform for digital twin developers and users. In *2023 IEEE Smart World Congress (SWC)*, page 1–8, August 2023. URL: <https://ieeexplore.ieee.org/abstract/document/10448890>, doi:10.1109/SWC57546.2023.10448890.
- [TH25] Dominik Tomaszuk and Timothée Haudebourg. RDF 1.2 schema. W3C working draft, W3C, February 2025. <https://www.w3.org/TR/2025/WD-rdf12-schema-20250224/>.
- [THXL24] Nirnaya Tripathi, Heidi Hietala, Yueqiang Xu, and Reshani Liyanage. Stakeholders collaborations, challenges and emerging concepts in digital twin ecosystems. *Inf. Softw. Technol.*, 169:107424, 2024. doi:10.1016/J.INFSOF.2024.107424.
- [TLMM23] Kunihiro Toumura, Michael Lagally, Michael McCool, and Ryuichi Matsukura. Web of things (wot) architecture 1.1. W3C recommendation, W3C, December 2023. <https://www.w3.org/TR/2023/REC-wot-architecture11-20231205/>.

- [TLWD18] Hao Tang, Di Li, Shiyong Wang, and Zhijie Dong. CASOA: an architecture for agent-based manufacturing system in the context of industry 4.0. *IEEE Access*, 6:12746–12754, 2018. doi:10.1109/ACCESS.2017.2758160.
- [TQ19] Fei Tao and Qinglin Qi. Make more digital twins. *Nature*, 573(7775):490–491, 2019. doi:10.1038/d41586-019-02849-1.
- [TWS20] Behnam Torabi, Rym Z. Wenkstern, and Robert Saylor. A collaborative agent-based traffic signal system for highly dynamic traffic conditions. *Autonomous Agents and Multi-Agent Systems*, 34(1):17, January 2020. doi:10.1007/s10458-019-09434-w.
- [TXQ⁺22] Fei Tao, Bin Xiao, Qinglin Qi, Jiangfeng Cheng, and Ping Ji. Digital twin modeling. *Journal of Manufacturing Systems*, 64:372–389, 2022. doi:10.1016/j.jmsy.2022.06.015.
- [TZLN18] Fei Tao, Meng Zhang, Yushan Liu, and A.Y.C. Nee. Digital twin driven prognostics and health management for complex equipment. *CIRP Annals*, 67(1):169–172, 2018. doi:10.1016/j.cirp.2018.04.055.
- [Uhr02] Adelinde M. Uhrmacher. Simulation for agent-oriented software engineering. San Antonio, TX, USA, 2002. URL: <https://pdfs.semanticscholar.org/b022/be67e1b2ff4f1f482ce3e4352608101ebcd8.pdf>.
- [Uni25] United Nations Statistics Division. The sustainable development goals report 2025, July 2025. Accessed: 2025-08-08. URL: <https://unstats.un.org/sdgs/report/2025>.
- [US98] Adelinde M. Uhrmacher and Bernd Schattenberg. Agents in discrete event simulation. In *European Simulation Symposium-ESS*, volume 98, pages 129–136. Citeseer, 1998.
- [UW18] Adelinde M. Uhrmacher and Danny Weyns. *Multi-Agent Systems: Simulation and Applications*. CRC Press, October 2018.

- [VCML23] Danai Vachtsevanou, Andrei Ciortea, Simon Mayer, and Jérémy Lemée. Signifiers as a first-class abstraction in hypermedia multi-agent systems. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '23*, page 1200–1208, Richland, SC, May 2023. International Foundation for Autonomous Agents and Multiagent Systems.
- [VDVD03] Rudi Van De Velde and Patrice Degoulet. *Clinical Information Systems: A Component-Based Approach*. Health Informatics. Springer New York, New York, NY, 2003. doi:10.1007/b97438.
- [vGR10] Tamara van Gog and Nikol Rummel. Example-based learning: Integrating cognitive and social-cognitive research perspectives. *Educational Psychology Review*, 22(2):155–174, May 2010. doi:10.1007/s10648-010-9134-7.
- [Vic97] B. C. Vickery. Ontologies. *Journal of Information Science*, 23(4):277–286, August 1997. doi:10.1177/016555159702300402.
- [VLK⁺22] Carina Nina Vorisek, Moritz Lehne, Sophie Anne Ines Klopfenstein, Paula Josephine Mayer, Alexander Bartschke, Thomas Haese, and Sylvia Thun. Fast healthcare interoperability resources (FHIR) for interoperability in health research: Systematic review. *JMIR Medical Informatics*, 10(7):e35724, 2022. Company: JMIR Medical Informatics Distributor: JMIR Medical Informatics Institution: JMIR Medical Informatics Label: JMIR Medical Informatics publisher: JMIR Publications Inc., Toronto, Canada. doi:10.2196/35724.
- [WAH⁺23] Alfred Winter, Elske Ammenwerth, Reinhold Haux, Michael Marschollek, Bianca Steiner, and Franziska Jahn. *Technological Perspective: Architecture, Integration, and Standards*, page 51–152. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-12310-8_3.

- [WC01] Michael Wooldridge and Paolo Ciancarini. Agent-oriented software engineering: The state of the art. In Paolo Ciancarini and Michael J. Wooldridge, editors, *Agent-Oriented Software Engineering*, page 1–28, Berlin, Heidelberg, 2001. Springer. doi: 10.1007/3-540-44564-1_1.
- [WDD21] Hang Wan, Michael David, and William Derigent. Modelling digital twins as a recursive multi-agent architecture: application to energy management of communicating materials. *IFAC-PapersOnLine*, 54(1):880–885, January 2021. doi:10.1016/j.ifacol.2021.08.104.
- [WDW⁺23] Huan Wang, Xiaoqiang Di, Yan Wang, Bin Ren, Ge Gao, and Junyi Deng. An intelligent digital twin method based on spatio-temporal feature fusion for IoT attack behavior identification. *IEEE J. Sel. Areas Commun.*, 41(11):3561–3572, 2023. doi: 10.1109/JSAC.2023.3310091.
- [Wil07] Erik Wilde. Putting things to REST. Ucb ischool report, School of Information, University of California, Berkeley, November 2007. URL: <https://escholarship.org/uc/item/1786t1dm>.
- [Win05] Michael Winikoff. JACKTM intelligent agents: An industrial strength platform. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 175–193. Springer, 2005.
- [Win12] Michael Winikoff. Challenges and directions for engineering multi-agent systems. *CoRR*, abs/1209.1428, 2012. URL: <http://arxiv.org/abs/1209.1428>, arXiv:1209.1428.
- [Win17] Michael Winikoff. Debugging agent programs with why? questions. In *Proceedings of the 16th Conference on Autonomous*

- Agents and MultiAgent Systems*, AAMAS '17, page 251–259, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems. doi:10.5555/3091125.3091166.
- [WJ95] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *Knowl. Eng. Rev.*, 10(2):115–152, 1995. doi:10.1017/S0269888900008122.
- [WJ04] Norman Walsh and Ian Jacobs. Architecture of the world wide web, volume one. W3C recommendation, W3C, December 2004. <https://www.w3.org/TR/2004/REC-webarch-20041215/>.
- [WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000. doi:10.1023/A:1010071910869.
- [WOO07] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, February 2007. doi:10.1007/s10458-006-0012-0.
- [Woo09] Michael J. Wooldridge. *An Introduction to MultiAgent Systems, Second Edition*. Wiley, 2009.
- [WSDD21] Michael Winikoff, Galina Sidorenko, Virginia Dignum, and Frank Dignum. Why bad coffee? Explaining BDI agent behaviour with valuing. *Artificial Intelligence*, 300:103554, 2021. doi:10.1016/j.artint.2021.103554.
- [WZM18] Monika Wenger, Alois Zoitl, and Thorsten Müller. Connecting PLCs with their asset administration shell for automatic device configuration. In *16th IEEE International Conference on Industrial Informatics, INDIN 2018, Porto, Portugal, July 18-20, 2018*, pages 74–79. IEEE, 2018. doi:10.1109/INDIN.2018.8472022.

- [XSLZ19] Yan Xu, Yanming Sun, Xiaolong Liu, and Yonghua Zheng. A digital-twin-assisted fault diagnosis using deep transfer learning. *IEEE Access*, 7:19990–19999, 2019. doi:10.1109/ACCESS.2018.2890566.
- [XUD⁺19] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable AI: A brief survey on history, research areas, approaches and challenges. In Jie Tang, Min-Yen Kan, Dongyan Zhao, Sujian Li, and Hongying Zan, editors, *Natural Language Processing and Chinese Computing*, pages 563–574, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-32236-6_51.
- [YBHR23] Elena Yan, Samuele Burattini, Jomi Fred Hübner, and Alessandro Ricci. Towards a multi-level explainability framework for engineering and understanding BDI agent systems. In Rino Falcone, Cristiano Castelfranchi, Alessandro Sapienza, and Filippo Cantucci, editors, *Proceedings of the 24th Workshop "From Objects to Agents", Roma, Italy, November 6-8, 2023*, volume 3579 of *CEUR Workshop Proceedings*, pages 216–231. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3579/paper17.pdf>.
- [YBHR25] Elena Yan, Samuele Burattini, Jomi Fred Hübner, and Alessandro Ricci. A multi-level explainability framework for engineering and understanding BDI agents. *Auton. Agents Multi Agent Syst.*, 39(1):9, 2025. doi:10.1007/S10458-025-09689-6.
- [YBK18] Kun-Hsing Yu, Andrew L. Beam, and Isaac S. Kohane. Artificial intelligence in healthcare. *Nature Biomedical Engineering*, 2(10):719–731, October 2018. doi:10.1038/s41551-018-0305-z.
- [YDL⁺25] Farzaneh Yousefi, Reza Dehnavieh, Maude Laberge, Marie-Pierre Gagnon, Mohammad Mehdi Ghaemi, Mohsen Nadali, and Najmeh Azizi. Opportunities, challenges, and requirements for artificial intelligence (AI) implementation in primary health care

- (PHC): a systematic review. *BMC Primary Care*, 26(1):196, 2025. doi:10.1186/s12875-025-02785-2.
- [YZW21] Jiachen Yang, Jipeng Zhang, and Huihui Wang. Urban traffic control in software defined internet of things via a multi-agent deep reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3742–3754, 2021. doi:10.1109/TITS.2020.3023788.
- [YZX21] Xinyu Yi, Yuxiao Zhou, and Feng Xu. TransPose: real-time 3D human translation and pose estimation with six inertial sensors. *ACM Trans. Graph.*, 40(4):86:1–86:13, 2021. doi:10.1145/3450626.3459786.
- [ZCF⁺23] Antoine Zimmermann, Andrei Ciortea, Catherine Faron, Eoin O’Neill, and María Poveda-Villalón. Pody: A Solid-Based Approach to Embody Agents in Web-Based Multi-Agent-Systems. In Andrei Ciortea, Mehdi Dastani, and Jieting Luo, editors, *Engineering Multi-Agent Systems - 11th International Workshop, EMAS 2023, London, UK, May 29-30, 2023, Revised Selected Papers*, volume 14378 of *Lecture Notes in Computer Science*, pages 220–229. Springer, 2023. doi:10.1007/978-3-031-48539-8_15.
- [ZFY⁺19] Shuwan Zhu, Wenjuan Fan, Shanlin Yang, Jun Pei, and Panos M. Pardalos. Operating room planning and surgical case scheduling: a review of literature. *Journal of Combinatorial Optimization*, 37(3):757–805, 2019. doi:10.1007/s10878-018-0322-6.
- [ZGK⁺24] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org, 2024.
- [ZGP⁺16] Jon Zugazagoitia, Cristiano Guedes, Santiago Ponce, Irene Ferrer, Sonia Molina-Pinelo, and Luis Paz-Ares. Current challenges in

- cancer treatment. *Clinical Therapeutics*, 38(7):1551–1566, 2016. doi:10.1016/j.clinthera.2016.03.026.
- [ZLK22] Xiaochen Zheng, Jinzhi Lu, and Dimitris Kiritsis. The emergence of cognitive digital twin: vision, challenges and opportunities. *International Journal of Production Research*, 60(24):7610–7632, December 2022. doi:10.1080/00207543.2021.2014591.
- [ZLQ24] Longyu Zhou, Supeng Leng, and Tony Q. S. Quek. Hierarchical digital-twin-enhanced cooperative sensing for UAV swarms. *IEEE Internet Things J.*, 11(20):33204–33216, 2024. doi:10.1109/JIOT.2024.3428476.
- [ZMZ⁺23] Long Zhang, Xiaozheng Ma, Zirui Zhuang, Haitao Xu, Vishal Sharma, and Zhu Han. Q-Learning aided intelligent routing with maximum utility in cognitive UAV swarm for emergency communications. *IEEE Trans. Veh. Technol.*, 72(3):3707–3723, 2023. doi:10.1109/TVT.2022.3221538.
- [ZQLL17] Yingfeng Zhang, Cheng Qian, Jingxiang Lv, and Ying Liu. Agent and cyber-physical system based self-organizing and self-adaptive intelligent shopfloor. *IEEE Trans. Ind. Informatics*, 13(2):737–747, 2017. doi:10.1109/TII.2016.2618892.
- [ZTZZ21] Tong Zhou, Dunbing Tang, Haihua Zhu, and Zequn Zhang. Multi-agent reinforcement learning for online scheduling in smart factories. *Robotics Comput. Integr. Manuf.*, 72:102202, 2021. doi:10.1016/J.RCIM.2021.102202.
- [ZWX⁺23] Long Zhang, Han Wang, Hongmei Xue, Hongliang Zhang, Qilie Liu, Dusit Niyato, and Zhu Han. Digital twin-assisted edge computation offloading in industrial internet of things with NOMA. *IEEE Trans. Veh. Technol.*, 72(9):11935–11950, 2023. doi:10.1109/TVT.2023.3270859.