

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Enabling General-Purpose Digital Twins Using Artificial Intelligence and
Distributed Systems**

AYMAN MOHAMED ABOELMAATY ABOELHASSAN

Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie industriel

Mai 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Enabling General-Purpose Digital Twins Using Artificial Intelligence and
Distributed Systems**

présentée par **Ayman Mohamed Aboelmaaty ABOELHASSAN**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Martin TRÉPANIÉ, président

Soumaya YACOUT, membre et directrice de recherche

Jinghui CHANG, membre

Elisa NEGRI, membre externe

DEDICATION

Thank you God the Almighty for your guidance, protection, and the countless blessings
bestowed upon me.

To my family for being in my life and for their continuous support.

ACKNOWLEDGEMENTS

Firstly, I express my sincere gratitude to my supervisor, Soumaya Yacout, for her patience and support during my Ph.D. I have learned a lot from her since we have different backgrounds. She actively supported me to engage in numerous activities to gain more academic and research experiences. She believed in my capabilities and opened the way for creative ideas. I appreciate her understanding and support throughout hard times.

I am also thankful to the jury committee for their meaningful feedback and acceptance to attend my Ph.D. defense as jury members and representatives of the director of graduate studies.

Finally, I would like to thank my colleagues and friends for their emotional support through difficult times and happy times.

RÉSUMÉ

Depuis le début de l'ère de l'industrie 4.0, les fabricants investissent dans la numérisation des systèmes de production et l'intégration de l'intelligence. Les perturbations de la chaîne d'approvisionnement et les variations de la demande du marché ont incité les fabricants à améliorer leur processus décisionnel actuel. Des systèmes de décision adaptatifs à réponse rapide évaluent les alternatives et ajustent les plans de production une fois les changements détectés. Le jumeau numérique (DT) est un concept important qui utilise des modèles de simulation et l'intelligence artificielle (IA) pour fournir une aide à la décision en temps réel. L'industrie 4.0 et l'intégration des technologies de l'information et de la communication (TIC) dans les systèmes de production ont fourni l'infrastructure nécessaire aux systèmes DT. Les DT aident les responsables de production à dépasser leurs limites actuelles avec une meilleure observabilité de leurs systèmes de production. Plusieurs applications DT ont été développées et étudiées dans la littérature; cependant, un petit nombre de publications ont considéré la généralité de la conception de leur système DT. De plus, le concept de DT manque de standardisation malgré l'intérêt pour la DT des communautés académiques et industrielles. Plusieurs concepts de l'industrie 4.0 ont été proposés avec des fonctionnalités similaires et enchevêtrées, telles que le système cyber-physique (CPS), le shell d'administration d'actifs (AAS) et l'ombre numérique (DS).

Dans cette recherche, un cadre DT à usage général est développé à l'aide de concepts informatiques. Une définition de DT basée sur la fonctionnalité est proposée pour séparer DS et DT. La définition proposée permet à différents outils d'aide à la décision de fonctionner à partir d'une représentation numérique en temps réel du système physique. Le CPS, les systèmes distribués et l'IA sont des technologies habilitantes clés pour le cadre DT proposé. CPS fournit l'infrastructure de connectivité cyber-physique requise pour intégrer les DT dans les systèmes de production. Les concepts de système distribué permettent la flexibilité et la généralité de la conception du cadre proposé, comme la conteneurisation, les microservices et la communication Pub/Sub. Les algorithmes d'IA combinent raisonnement et inférence pour créer des systèmes de recommandation intelligents. Le cadre DT proposé permet aux concepteurs d'ajuster les modèles numériques (DM) pendant l'exécution pour refléter les modifications du système de production. De nouveaux DM et applications sont déployés sur les systèmes DT existants sans perturber leurs services. La conteneurisation et la séparation entre les DM et les applications DT permettent à différents outils d'aide à la décision de fonctionner simultanément.

Une méthodologie est proposée pour le développement du système DT. L'approche de développement agile est adoptée pour concevoir la méthodologie proposée comme un processus itératif par étapes. Les analystes commerciaux, les responsables de production et les opérateurs participent à la définition des rôles DT et à l'amélioration constante du système DT. La méthodologie proposée soutient des cycles de production rapides et une amélioration continue. Plusieurs cas d'utilisation industrielle ont été étudiés dans le cadre de cette recherche, tels qu'un système de transport de matériaux de mine à ciel ouvert, un système de compression de gaz pour le transport offshore, un système de production de semi-conducteurs, et un système d'entrepôt intelligent. Ces cas d'utilisation ont été utilisés pour valider la flexibilité et la généralité du framework DT proposé. Ces cas d'utilisation étudiés les avantages de l'intégration de l'intelligence dans les systèmes de production et les techniques de conception de modèles nécessaires pour créer des modèles DS et des applications DT.

ABSTRACT

Since the start of the industry 4.0 era, manufacturers are investing in production system digitalization and intelligence integration. Supply-chain disruptions and market demand variations influenced manufacturers to improve their current decision-making process. Adaptive fast-response decision systems evaluate alternatives and adjust production plans once changes are detected. Digital twin (DT) is a prominent concept that utilizes simulation models and Artificial intelligence (AI) to provide real-time decision support. Industry 4.0 and the integration of Information and communication technologies (ICT) into production systems provided the needed infrastructure for DT systems. DTs assist production managers to surpass their current limitations with better observability of their production systems. Multiple DT applications were developed and studied in the literature; however, a handful number of publications considered the generality of their DT system design. Additionally, the DT concept lacks standardization despite the interest in DT from both academic and industrial communities. Multiple industry 4.0 concepts were proposed with similar and entangled functionality, such as Cyber-physical system (CPS), Asset administration shell (AAS), and Digital shadow (DS).

In this research, a general-purpose DT framework is developed using computer science concepts. A functionality-based definition of DT is proposed to separate between DS and DT. The proposed definition enables different decision-support tools to operate using a real-time digital representation of the physical system. CPS, distributed systems, and AI are key enabling technologies for the proposed DT framework. CPS provides the cyber-physical connectivity infrastructure required to integrate DTs into production systems. Distributed system concepts enable the flexibility and generality of the proposed framework design, such as containerization, Microservices, and Pub/Sub communication. AI algorithms combine reasoning and inference for creating smart recommendation systems. The proposed DT framework allows designers to adjust Digital models (DMs) during runtime to reflect production system changes. New DMs and applications are deployed to existing DT systems without disrupting their services. Containerization and the separation between DMs and DT applications enable different decision-making tools to operate concurrently.

A methodology is proposed for DT system development. The agile development approach is adopted to design the proposed methodology as a stepwise iterative process. Business analysts, production managers, and operators are involved in defining the DT roles and constantly improving the DT system. The proposed methodology supports fast production

cycles and continuous improvement. Multiple industrial use cases were studied through this research, such as an open-mine pit material transportation system, a gas compression system for offshore transportation, a semiconductor production system, and an intelligent warehouse system. These use cases were used to validate the proposed DT framework's flexibility and generality. These use cases studied the benefits of embedding intelligence into production systems, and the model design techniques needed to build DS models and DT applications.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
TABLE OF CONTENTS	ix
LIST OF TABLES	xiv
LIST OF FIGURES	xv
LIST OF SYMBOLS AND ACRONYMS	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Basic concepts	1
1.1.1 Digital Twin	1
1.1.2 Distributed system	5
1.1.3 Artificial intelligence	6
1.2 Elements of the problem	6
1.2.1 Industrial adaptation of DT	6
1.2.2 Lack of standardization	7
1.2.3 Lack of generality	7
1.2.4 Potential benefits of distributed systems and AI	8
1.3 Research objectives	8
1.4 Research questions	10
1.5 Research hypotheses	11
1.6 Points of originality	12
CHAPTER 2 THESIS ORGANIZATION	15
CHAPTER 3 LITERATURE REVIEW	18
3.1 Digital Twin	18
3.1.1 Digital Twin Definition	18

3.1.2	Digital Twin Frameworks	19
3.1.3	Digital Twin Review Papers	21
3.2	Digital Shadow	21
3.3	Distributed Systems in Industrial Domain	22
3.3.1	Service Oriented Architecture	22
3.3.2	Event-Driven Architecture	23
3.3.3	Containerization	23
3.4	Literature Synthesis	24
CHAPTER 4 ARTICLE 1: A FRAMEWORK FOR DIGITAL TWIN DEPLOYMENT IN PRODUCTION SYSTEMS		25
4.1	Abstract	26
4.2	Introduction	26
4.2.1	Production Systems	27
4.2.2	Related Work	28
4.2.3	Contribution	28
4.3	Digital Shadow	29
4.4	Digital Twin	31
4.5	Integration in Automation Process	33
4.5.1	Feedback to Physical World	33
4.5.2	Integration with MES/ERP	33
4.6	Conclusion	34
CHAPTER 5 ARTICLE 2: EMBEDDING REINFORCEMENT LEARNING IN SIMULATION		35
5.1	Abstract	36
5.2	Introduction	36
5.3	Background	38
5.3.1	Decision Support using Simulation	38
5.3.2	Agent-based Modeling	38
5.3.3	Reinforcement Learning	38
5.3.4	Reinforcement Learning in Simulation	39
5.4	Embedding Reinforcement Learning	39
5.4.1	User Experience	40
5.4.2	Hyperparameter Optimization	41
5.5	Truck Dispatching Use Case	42
5.5.1	System Description	42

5.5.2	Business Objective	43
5.5.3	Reference Heuristic Rule	43
5.6	Results and Discussion	43
5.6.1	Reinforcement Learning (RL) Results	43
5.6.2	Benefits of Embedding RL in AnyLogic	46
5.7	Conclusion	47
CHAPTER 6 ARTICLE 3: A DIGITAL SHADOW FRAMEWORK USING DIS-		
TRIBUTED SYSTEM CONCEPTS		48
6.1	Abstract	49
6.2	Introduction	49
6.2.1	Motivation	50
6.2.2	Contribution	51
6.2.3	Organization	51
6.3	Literature Review	52
6.3.1	Digital Shadow	52
6.3.2	Digital Twin	53
6.4	Distributed Systems	56
6.4.1	Event-Driven Architecture	56
6.4.2	Service Oriented Architecture	57
6.4.3	Containerization	58
6.5	Digital Shadow Framework	59
6.5.1	Digital Shadow Coordinator	59
6.5.2	Pub/Sub Communication channel	60
6.5.3	Source Gateway	63
6.5.4	Digital Model Manager	64
6.5.5	Digital Model	64
6.6	Case Study	69
6.6.1	Source Gateway	70
6.6.2	Gas Pressure Model	71
6.6.3	Gas Compressor Model	73
6.6.4	Gas Compressibility Model	75
6.6.5	Lube Oil Reservoir Model	75
6.6.6	Visualization	76
6.7	Discussion	77
6.7.1	DS Throughput	77

6.7.2	Scalability	78
6.7.3	Fault Tolerance	79
6.8	Conclusion	79
6.9	Future Work	80
CHAPTER 7 ARTICLE 4: GENERAL PURPOSE DIGITAL TWIN FRAMEWORK USING DIGITAL SHADOW AND DISTRIBUTED SYSTEM CONCEPTS		81
7.1	Abstract	82
7.2	Introduction	82
7.2.1	Motivation	82
7.2.2	Contribution	83
7.2.3	Organization	84
7.3	Related Work	84
7.3.1	Digital Twin	84
7.3.2	Digital Shadow	85
7.3.3	DT Models Development	86
7.4	Digital Twin Framework	86
7.4.1	DT Manager	88
7.4.2	Pub/Sub Communication Channel	90
7.4.3	Digital Shadow	91
7.4.4	DT Application	93
7.5	Designing Models for DT vs DS	94
7.5.1	Digital Shadow Model	94
7.5.2	Digital Twin Application	96
7.6	Semiconductor Production System Use Case	98
7.6.1	Source Gateway	98
7.6.2	Digital Shadow	99
7.6.3	Digital Twin	102
7.7	Conclusion and Outlook	108
7.8	Future Work	110
7.9	Author's Contribution	110
CHAPTER 8 ARTICLE 5: A DIGITAL TWIN DEVELOPMENT METHODOLOGY FOR MANUFACTURING SYSTEMS		111
8.1	Abstract	112
8.2	Introduction	112
8.2.1	Designing digital twin systems	114

8.2.2	Virtual commissioning and operation management using digital twins	115
8.2.3	Research objectives and contribution	116
8.3	Methodology	116
8.3.1	Planning Phase	117
8.3.2	Transformation Phase	119
8.3.3	Modeling Phase	120
8.3.4	Shadowing Phase	120
8.3.5	Twinning Phase	121
8.4	Experimental Testbed	121
8.4.1	Planning Phase	122
8.4.2	Transformation Phase	125
8.4.3	Modeling Phase	126
8.4.4	Shadowing Phase	128
8.4.5	Twinning Phase	129
8.5	Discussion	130
8.6	Conclusion and outlook	131
8.7	Future work	133
8.8	Statements and declarations	134
8.8.1	Competing interests	134
8.8.2	Funding	134
8.9	Author's Contribution	134
CHAPTER 9 GENERAL DISCUSSION		135
9.1	Research objectives	136
9.2	Research questions	136
9.3	Research hypotheses	140
9.4	Summary	141
9.4.1	Implications	142
9.5	Challenges and limitations	142
9.6	Future Research	143
CHAPTER 10 CONCLUSION AND RECOMMENDATIONS		146
REFERENCES		147

LIST OF TABLES

Table 2.1	Research objectives and questions addressed by the research articles .	16
Table 5.1	Heuristic Rule Results	44
Table 6.1	Overview of DT Applications in Industrial Domain	55
Table 6.2	Description of Tags Pulled from Cognite data fusion (CDF)	72
Table 6.3	DS Components' Performance	78
Table 6.4	Kafka Cluster Performance	78
Table 7.1	Lot Agent Elements Description	102
Table 7.2	Timer Agent Elements Description	103
Table 7.3	Recipe Agent Elements Description	104

LIST OF FIGURES

Figure 1.1	Digital twin virtual representation of the physical system	2
Figure 1.2	DT components and their relationship	3
Figure 1.3	DT types and topologies	4
Figure 1.4	Dataflow difference between DM, DS, and DT	4
Figure 1.5	Fidelity difference between DM, DS, and DT	5
Figure 1.6	UML class diagram describing the relation between the research objectives, questions, and hypotheses	10
Figure 2.1	Graphical description of the thesis organization	15
Figure 4.1	Traditional Approach for data gathering, management, and planning	27
Figure 4.2	A Digital Shadow Architecture	29
Figure 4.3	A Digital Twin Architecture	32
Figure 4.4	A controlling Digital Twin	33
Figure 5.1	RL library components' interface with AnyLogic models	40
Figure 5.2	<i>Hyperparameter Optimizer</i> interface with an AnyLogic model	41
Figure 5.3	Truck transportation environment	42
Figure 5.4	Percentage of total trucks dispatched in the 1st congestion scenario .	44
Figure 5.5	Percentage of total trucks dispatched in the 2nd congestion scenario .	45
Figure 5.6	Percentage of total trucks dispatched in the 3rd congestion scenario .	46
Figure 6.1	Digital Shadow System Architecture	60
Figure 6.2	UML Sequence Diagram of DM Creation	61
Figure 6.3	An example of Pub/Sub Communication Channel using Apache Kafka®	62
Figure 6.4	UML Sequence Diagram of DMM-DM Initialization	65
Figure 6.5	UML Sequence Diagram of DMM-DM Communication for Discrete-Time DM	67
Figure 6.6	UML Sequence Diagram of DMM-DM Communication for Continuous-Time DM	68
Figure 6.7	Digital Model Interface with the Digital Model Manager	69
Figure 6.8	First Stage Natural Gas Compressor System [1]	70
Figure 6.9	Distributed System Servers	70
Figure 6.10	Digital Models Used to Represent the System	71
Figure 6.11	Compressor's Pressure and Temperature Sensors Measurements . . .	72
Figure 6.12	ANN Model Estimate of the Compressed Gas Pressure During DS Operation	73

Figure 6.13	Pressure-Temperature Model Estimate of the Compressed Gas Pressure During DS Operation	74
Figure 6.14	Lube Oil Reservoir AnyLogic® Model	76
Figure 7.1	Functionality-based Separation between Digital Twin vs Digital Shadow	86
Figure 7.2	Digital Twin Development Process	87
Figure 7.3	Digital Twin Framework Architecture	88
Figure 7.4	Example of a Kubernetes (K8s) Cluster	89
Figure 7.5	DT Manager connection to K8s cluster	89
Figure 7.6	Digital Shadow Architecture [2]	91
Figure 7.7	Tightly-Coupled Source Gateway	92
Figure 7.8	Loosely-Coupled Source Gateway	92
Figure 7.9	DS Model Execution Flow Chart	95
Figure 7.10	DT Simulation Model Execution Flow Chart	97
Figure 7.11	Product DM Ontology Graph	100
Figure 7.12	Equipment Agent Process in Production System DM	101
Figure 7.13	Lot Agent Description in Production System DM	102
Figure 7.14	Timer Agent Description in Production System DM	103
Figure 7.15	Recipe Agent Description in Production System DM	103
Figure 7.16	Screenshot of an Equipment State in Production System DM	105
Figure 7.17	Example of an Work-in Progress (WIP) Lot Information Displayed using Node-Red Business intelligence (BI) dashboard	106
Figure 7.18	Dispatching Recommendation DT Logic Flowchart	106
Figure 7.19	A Screenshot for Dispatching Recommendation DM Web-Based Interface	106
Figure 7.20	Equipment Agent Process in Deadline Estimation DM	108
Figure 7.21	Screenshot of the start screen of Deadline Estimation DM	108
Figure 7.22	Screenshot of an Equipment State in Deadline Estimation DM	109
Figure 8.1	Digital twin as a part of cyber-physical system	113
Figure 8.2	Dataflow in DM, DS, and DT	114
Figure 8.3	A summary of a collection of digital twin design methodologies in literature	115
Figure 8.4	The proposed digital twin design methodology	118
Figure 8.5	Schematic diagram of the warehouse layout	122
Figure 8.6	Functional flow block diagram of the CPS of the warehouse	123
Figure 8.7	The CPS control system architecture diagram	124
Figure 8.8	The CPS communication network topology	125
Figure 8.9	Digital twin system integrated into the CPS control system	126

Figure 8.10	<i>Product DM</i> ontology diagram	127
Figure 8.11	<i>Warehouse DM</i> screenshot of the model's 3D view	128
Figure 8.12	UML sequence diagram for products reordering	130
Figure 8.13	UML sequence diagram for moving products after inspection	131
Figure 8.14	A screenshot of the CPS warehouse dashboard in <i>Warehouse DM</i>	132
Figure 8.15	A screenshot of the product dashboard in <i>Warehouse DM</i>	133
Figure 8.16	A screenshot of the AMR dashboard in <i>Warehouse DM</i>	134

LIST OF SYMBOLS AND ACRONYMS

AAS	Asset administration shell
ABM	Agent-based model
AI	Artificial intelligence
ANN	Artificial neural network
AMR	Autonomous mobile robot
API	Application programming interface
B2B	Business-to-business
BI	Business intelligence
BOM	Bill of materials
CD	Continuous deployment
CDF	Cognite data fusion
DES	Discrete event simulation
CPPS	cyber-physical production systems
CPS	Cyber-physical system
DDS	Data Distribution Service
DM	Digital model
DMM	Digital model manager
DMS	Data, model, and service
DS	Digital shadow
DT	Digital twin
DQN	Deep-Q network
ERP	Enterprise Resource Planning
FEM	Finite Element Model
FMI	Functional mock-up interface
GALS	Globally asynchronous locally synchronous
GCP	Google cloud platform
GKE	Google Kubernetes engine
GP	Gaussian process
HMI	Human-machine interface
IC	Integrated circuit
ICT	Information and communication technologies
IE	Industrial Ethernet
IIoT	Industrial internet of things

IIRA	Industrial internet reference architecture
IPC	Inter-process communication
IoP	Internet of production
IoT	Internet of things
IT	Information technology
JRE	Java runtime environment
K8s	Kubernetes
KPI	Key Performance Indicator
MES	Manufacturing Execution System
ML	Machine Learning
MQ	Message Queue
OID	Open Industrial Data
OME	Observable Manufacturing Element
OPC-UA	OPC Unified Architecture
OR	Operational Research
OT	Operational Technology
OWL	Web Ontology Language
PLM	product lifecycle management
RAMI 4.0	Reference Architecture Model Industrie 4.0
RFID	Radio-Frequency Identification
RL	Reinforcement Learning
SCADA	supervisory control and data acquisition
SITAM	Stuttgart IT-Architecture for Manufacturing
SME	small and medium-sized enterprises
SOA	Service Oriented Architecture
TPE	Tree of Parzen Estimators
TRL	Technology Readiness Level
VC	virtual commissioning
WDT	Web-based Digital Twin
WGP	German Academic Association for Production Technology
WIP	Work-in Progress

CHAPTER 1 INTRODUCTION

Supply chain disruptions grew into a worldwide problem during the Covid-19 pandemic [3]. Travel restrictions, maritime transportation delays, and sudden demand shifts pushed industrial systems to their limits. Supply shortages and volatile demand showed the drawbacks of efficient-but-rigid production systems [4, 5].

Digital transformation of production systems triggered the start of the industry 4.0 era. The industry 4.0 paradigm emphasizes the benefits of using well-known computer science technologies in production systems, such as Internet of things (IoT), AI, and cloud technologies [6]. Supply chain variability and market uncertainty hastened the adaption industry 4.0 paradigm. Smart production systems adapt to supply and demand changes using real-time monitoring systems and autonomous detection of system changes.

Adapting industry 4.0 technologies into production systems is an ongoing process [7, 8]. Government initiatives and business opportunities encourage both small and medium-sized enterprises (SME) and large enterprises to deploy state-of-the-art industry 4.0 technologies in their production systems [9]. The integration of Information technology (IT) and Operational Technology (OT) into production systems enables the usage of new technologies and services. Modern industrial controllers have evolved outside the ISA-95 automation pyramid functional specifications to keep up with the manufacturers' needs [10, 11]. IT/OT integration also migrated some industrial tools from the OT domain into the IT domain such as cloud-based supervisory control and data acquisition (SCADA) systems [12].

Digital twin (DT) is a fast-growing concept that emphasizes the importance of simulation and AI in the industrial domain [13]. DT generates a virtual replica of the physical system using knowledge-based descriptive models [14, 15]. Real-time information is captured from the physical system to update the virtual representation. DT's simulation models and AI tools provide decision support for operators. DT may also be used to adjust the production system.

1.1 Basic concepts

1.1.1 Digital Twin

NASA first used a physical twin during the apollo mission in the 1960s. An on-ground physical replica of the apollo space shuttle was used to reproduce and confirm computations calculated by the space shuttle computers [13]. Afterward, a pseudo-DT concept called *virtual product*

was proposed by Grieves in 2003 as a product lifecycle management (PLM) concept [14–16]. Grieves defined *virtual product* as a collection of digital information describing the physical product throughout its lifetime. Digital twin (DT) was introduced by NASA engineers in the 2010 technology roadmap. DT was defined as a real-time high-fidelity multi-scale multi-physics model, which is used to predict vehicle health index in real-time [17,18]. Figure 1.1 shows the connectivity between the physical system and its virtual counterpart in a DT.

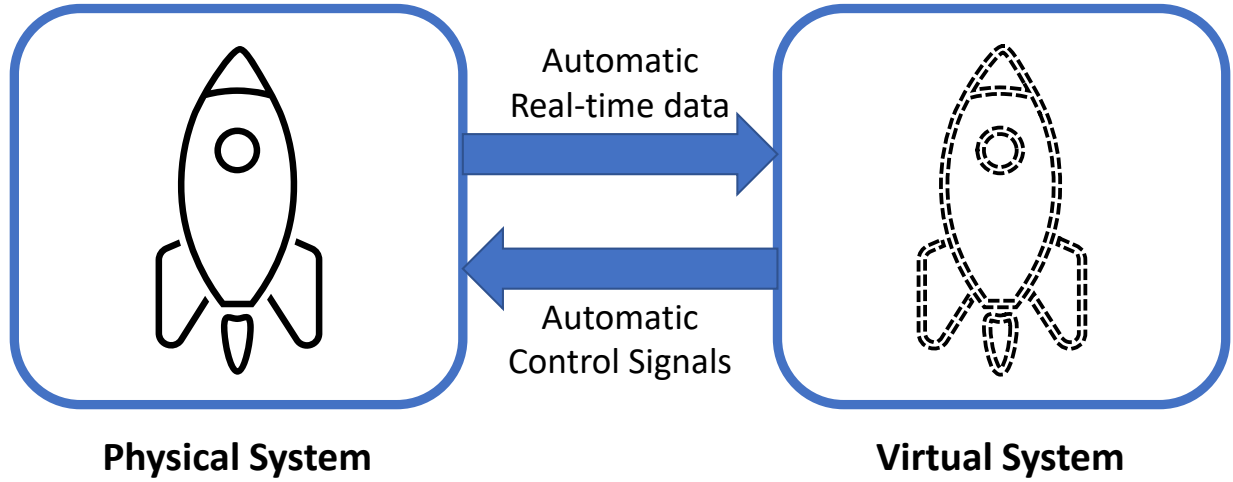


Figure 1.1 Digital twin virtual representation of the physical system

DT gained momentum in the industry 4.0 era. DT companies are expected to achieve revenue of \$183 billion by 2031 [19]. Potential benefits of DT in the industrial domain were first discussed by Rosen et al. [20]. The DT concept broadened to define DT as a digital replica of a product, machine, system, or system of systems [13, 14, 21]. DT captures its physical counterpart's state and behavior in real-time, which is aggregated over its lifetime. Simulation models, numerical models, and statistical models are used to describe the physical system behavior.

DT provide decision-support using simulation models and AI algorithms [8,14,22–24]. virtual commissioning (VC) use simulation models to predict system behavior before applying new configuration settings or deploying new products. Quality management tools use real-time data to trace the production process quality and machine condition. Operations management updates the production plan using real-time system status to improve machine utilization and reduce downtimes.

DT Components

DT is composed of multiple components according to the used DT definition. For instance, Grieves and Vickers [16] divided DT into a physical product, a virtual product, and the connection between both. Park et al. [25] divided DT into 3 components; the physical system, simulation models and inner services, and intelligent applications. While Tao et al. [26,27] divided DT into 5 components; physical components, virtual components, services and tools, collected data, and the communication between all previous components. Stark and Damerau [15] defined 8 dimensions for a DT; the physical entity, connectivity, data frequency, CPS intelligence, simulation, model abstraction, human interface, and product lifecycle context. Figure 1.2 shows a summarized description of DT components.

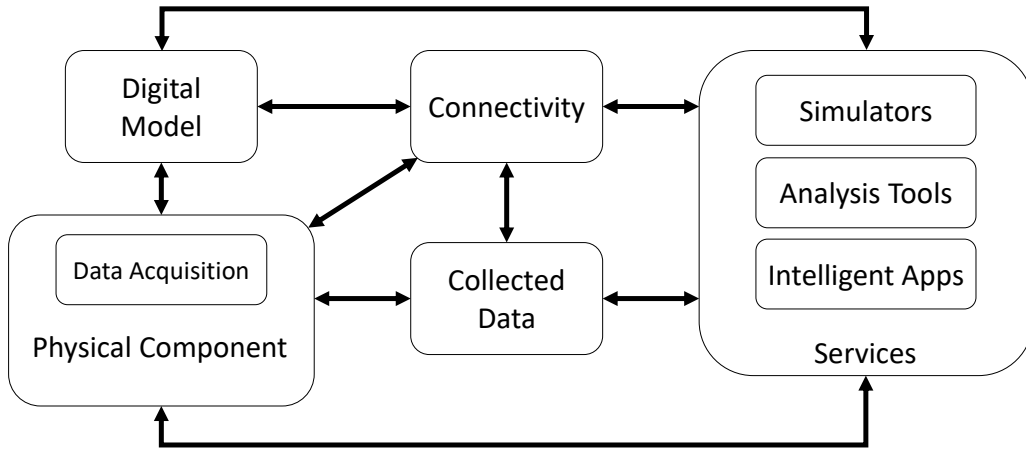


Figure 1.2 DT components and their relationship

DT may be defined as a single DT or a collection of sub-DTs [24,28]. A DT creates a digital representation using a single DM or multiple DMs of the same physical system. DT may also be designed as a set of separate sub-DTs working in collaboration or independently. DT definition allows DTs to be deployed on a cloud platform or on-site using IoT. Figure 1.3 shows different topologies used in designing and deploying DTs.

Digital Shadow

Digital shadow (DS) is an artifact concept of DT [29]. DS was introduced by German Academic Association for Production Technology (WGP) in 2016 [30,31]. WGP defined DS as an information collection platform used for collecting real-time data from multiple sources. DS enables decision-support tools by providing a virtual description of the physical system.

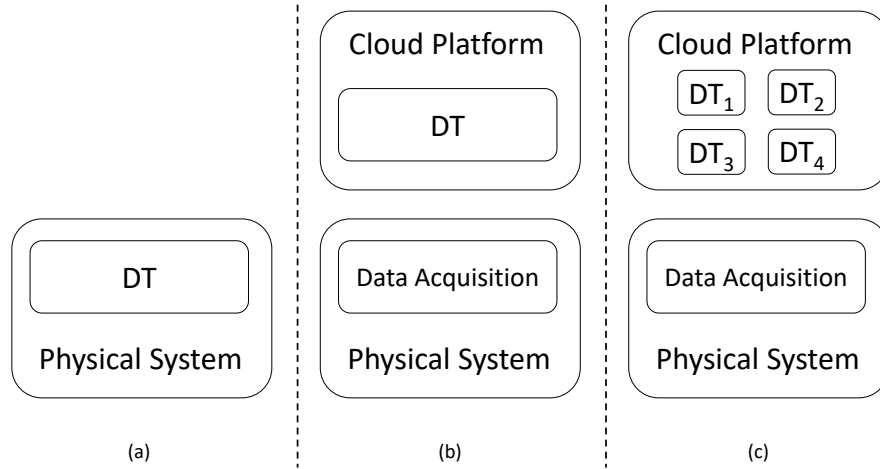


Figure 1.3 DT types and topologies

DS and DT concepts are overlapping. Both concepts collect real-time data from their corresponding physical system to create a digital representation. DS is usually described as a one-direction DT [8, 14, 29, 32, 33]. DS automatically collects real-time data from the physical system; however, DS does not directly affect the physical system. Figure 1.4 shows the difference between DS and DT in terms of dataflow automation.

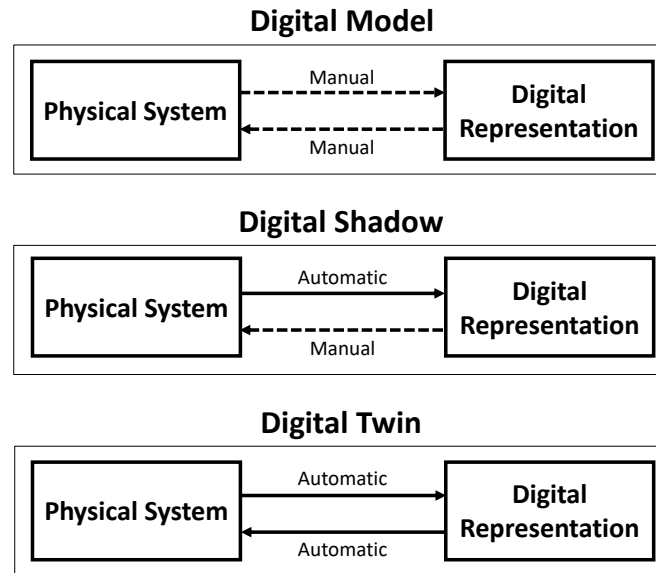


Figure 1.4 Dataflow difference between DM, DS, and DT

Other researchers define DS as a low-fidelity digital replica [15, 30, 34]. In this definition, DS focuses on data aggregation and fusion to create a digital description of the physical system. While DT is defined as a model-driven concept that creates a behavioral description of the physical system for decision support. Figure 1.5 shows the difference between DS and DT in terms of physical system description fidelity.

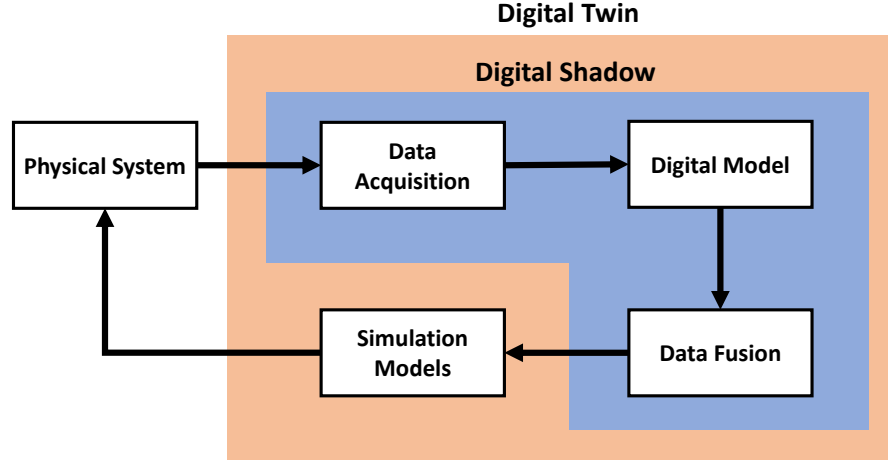


Figure 1.5 Fidelity difference between DM, DS, and DT

1.1.2 Distributed system

Distributed system is a research field in the computer science domain. It studies the coordination between connected computer systems. The goal is to build a collaborative heterogeneous system over non-reliable communication networks while appearing as a single system to end-users [35]. The main challenges in this research field are; components synchronization, applications concurrency and parallelism, and multiple independent failure sources. Multiple technologies are developed in this field such as virtualization, containers, code migration, Service Oriented Architecture (SOA), microservices, and service replication.

Virtualization is a distributed system technology that hides the details of the computational resources from the hosted software applications [36–38]. This enables resource sharing and application isolation on cloud platforms. Containerization is an operating-system-level virtualization approach [38, 39]. Containers are multiple isolated virtual environments running on the same operating system. In this research, containerization is used in the modularization of DT components. Digital models (DMs) are containerized to enable their deployment on different systems with the needed dependencies as standalone applications. The containerized DMs are then deployed on the DT framework system.

Service Oriented Architecture (SOA) is a distributed system architecture for connecting multiple systems and components. SOA enables the flexibility and scalability of software applications through a loosely coupled approach [40, 41]. Applications in SOA are divided into subcomponents, called services, each providing a distinct single functionality. Microservices are SOA-based architecture with higher granularity and self-contained. Application programming interface (API) is used to communicate with a microservices component to execute its service functionality.

1.1.3 Artificial intelligence

Artificial intelligence (AI) is the domain of studying machine-based intelligence [42]. AI is a large domain that contains multiple sub-fields such as computer vision, speech recognition, and natural language processing. Researchers create AI applications using multiple problem-solving algorithms such as mathematical optimization, search algorithms, formal logic, and probability.

Machine Learning (ML) is an AI sub-field that focuses on building algorithms for teaching machines [43]. Sample data is used in training ML models to generate predictions without directly explaining implicit data patterns. Statistical learning, Artificial neural network (ANN), and mathematical optimization are studied to design predictive models. Training data properties divide ML into supervised, unsupervised, and semi-supervised models. Supervised ML models are trained on labeled data to predict the label of new unidentified data. Unsupervised ML models are trained on unlabeled data in order to divide them into segments and clusters. Semi-supervised ML models are trained on partially labeled data. Reinforcement Learning (RL) is a type of semi-supervised ML, where RL agents are trained to take actions based on feedback reward describing the goodness of the RL actions.

1.2 Elements of the problem

1.2.1 Industrial adaptation of DT

The industrial domain is increasingly adopting DT for its potential benefits. DT gained momentum along with the rise of the industry 4.0 paradigm. DT market is expected to reach \$183 billion in revenue by 2031 [19]. DT concept is being used in many manufacturing systems to monitor physical assets and reduce downtimes [44]. Despite the motivation to utilize DT, industrial applications focus on real-time monitoring of assets and data analysis. Simulation models in such applications are limited to visualization of the manufacturing system components and monitoring their state. The focus on real-time monitoring restricts

the operations performed by DTs to operator training, reporting, and visualization.

Tool dependency is also a constraining aspect in the industrial domain [45,46]. Custom DT solutions are developed for manufacturing systems to perform case-specific operations. The custom-designed approach in developing DTs limits the use of such DTs to their corresponding systems. Expanding these DTs systems to other domains requires immense modifications. While integrating other tools requires vendors' support.

1.2.2 Lack of standardization

NASA's DT definition sets a vision for DT capabilities and goals. However, the broad DT vision generated multiple definitions and roles for DT in industrial systems [21,47]. DTs are defined based on their real-time monitoring properties [48,49], simulation capabilities [15,18,27], connectivity [16,25,50,51]. Moreover, the emergence of other industry 4.0 concepts such as CPS, Industrial internet of things (IIoT), and AAS made it hard to differentiate between DT and these concepts [52–55].

The standardization of DT is currently in progress. Research studies have investigated the implementation of DT in industry 4.0 architecture references such as Reference Architecture Model Industrie 4.0 (RAMI 4.0) [54–58]. DT is defined as a part of AAS in some of these studies [54,55]. Other efforts are ongoing to describe DT functionality in modern production systems [59,60]. ISO 23247 is a generic guideline for developing DTs for case-specific systems [59,61]. While IEC 62832 describes a high-level digital factory framework [60].

1.2.3 Lack of generality

Most of the published articles focus on DT applications [8]. Some studies investigate the integration of simulation models into production systems' IT network [25,62,63]. In these studies, DT applications are limited to the applied use cases. Additionally, these DTs are not scaled easily since considerable modifications are needed to add new services and tools to the DT. Other studies investigated the IT/OT integration, IoT, and DT interfacing [63–68]. This approach focuses more on real-time data capturing than decision-support applications. Hence, some DTs end up as real-time monitoring systems as in [48,49].

The lack of consensus and standardization of DTs had generated a diverse range of DTs in the literature. A general-purpose DT is required for the wide deployment of DT in the industrial domain. It provides decision-support tools for different system types using multiple modeling techniques. The envisioned DT integrates simulation models from different vendors to generate a holistic representation of the production system. It also enables dynamic

improvement to reflect changes in the real world.

1.2.4 Potential benefits of distributed systems and AI

Distributed system and AI concepts provide the technological means to build a general-purpose DT. Modularization of DT components allow multiple models and applications to operate independently. Moreover, modularization hides DT components' dependencies, reducing the needed effort to integrate new applications into a DT. Event-driven architecture and Microservices enable DT models to collaborate and exchange information, which enables building multi-physics multi-scale representation as a multi-model system. Finally, AI is used in DT applications to provide data insights and reasoning for decision-makers. AI's possible gains in the industrial domain are limitless. DT opens new opportunities for industrial engineers to apply AI in their fields.

1.3 Research objectives

The main objective is to design a flexible scalable general-purpose Digital twin framework. Flexible DT framework adapts to changes in both the physical and virtual systems. Changes in product design, production process, and system behavior are easily reflected in the DT. Additionally, flexible DTs enable DT designers to modify existing DMs during runtime. DT scalability is defined by its ability to incorporate new DMs and applications into existing DT. A general-purpose DT framework enables DT designers to integrate different simulation models and analysis tools into the same DT.

This research is based on the integration of computer science concepts into the industrial domain for achieving a general-purpose DT framework. Distributed system concepts are used to provide the ICT infrastructure needed to build the DT framework. Pub/Sub communication system provides an information-sharing system for DT components. While containerization and orchestration enable the flexibility and scalability of the deployed DT. Artificial intelligence is used to develop decision-support tools and recommendation systems as DT applications. ML models are used for pattern recognition, diagnosis, and prognosis. While RL agents are used for decision-making. AI is embedded in simulation models in order to simulate the system's behavior and predict future consequences.

Objective A: Develop a flexible tool-independent DS framework

According to NASA's DT definition, DT is a set of real-time simulation models mimicking the physical system. These simulation models are connected to the physical

system to automatically capture the system's status. The first step to create a DT is to develop its DS. DS provides automatic dataflow from the physical system to the virtual system. The physical system information is then passed to the corresponding DMs to update their states. DS enables multiple DMs to run concurrently in order to enable a multi-scale description of the physical system. Additionally, the DS framework is designed to incorporate multi-physics models using different modeling tools. DMs are designed to exchange information to generate a holistic view. Future integration into DT is considered during the DS framework.

Distributed system concepts are used in designing the DS framework. Gateways capture the physical system information, which is then sent to DMs through a Pub/Sub communication channel. Containerization is used to encapsulate DMs into container instances. Orchestration provides system flexibility through managing the availability of deployed DMs and their version management.

Objective B: Develop a DS-based general-purpose DT framework

DS is the core component of a DT. DS provides real-time model-based information for the DT decision-support applications. DMs publishes their model states to the shared Pub/Sub communication channel. DT applications then read the latest system status. The separation between DMs and DT applications enables DT designers to hide DM's complexity. it also allows multiple DT applications to work concurrently using the same DM. The DT framework is designed to enable different decision-support applications such as virtual commissioning, quality inspection, operations management, and recommendation system.

Similar to the DS framework, distributed system concepts are used in designing the DT framework. SOA and Pub/Sub communication are used to integrate DS into the DT framework. Each DT application subscribes to its required state messages at the Pub/Sub communication channel. DT applications are encapsulated using containerization as DMs. Orchestration provides Ad-Hoc capabilities to execute DT applications upon decision-maker request. Moreover, orchestration enables DT designers to add new applications to the DT without disrupting the DT availability.

Objective C: Design an agile-driven development methodology for DTs

The standardization of DT development is still an ongoing process. The goal of this objective is to design a development methodology for implementing the proposed DT framework. The development methodology takes into consideration the CPS infrastructure needed to connect a DT to the physical system. The DT goal and scope are

defined at the start of the DT project in collaboration with the different stakeholders. The methodology then provides a stepwise approach to design and implement DMs, then integrate them into a DS, and finally transform the DS into a DT. The agile development methodology is adopted to utilize its fast-paced production focus, continuous improvement approach, and stakeholders' involvement throughout the project duration.

The research objectives are achieved through answering a set of exploratory research questions. While multiple hypotheses are stated according to the distributed system and AI capabilities. These hypotheses describe the expected benefits of distributed systems utilized during the research study. Figure 1.6 presents the relations between the research objectives, questions, and hypotheses.

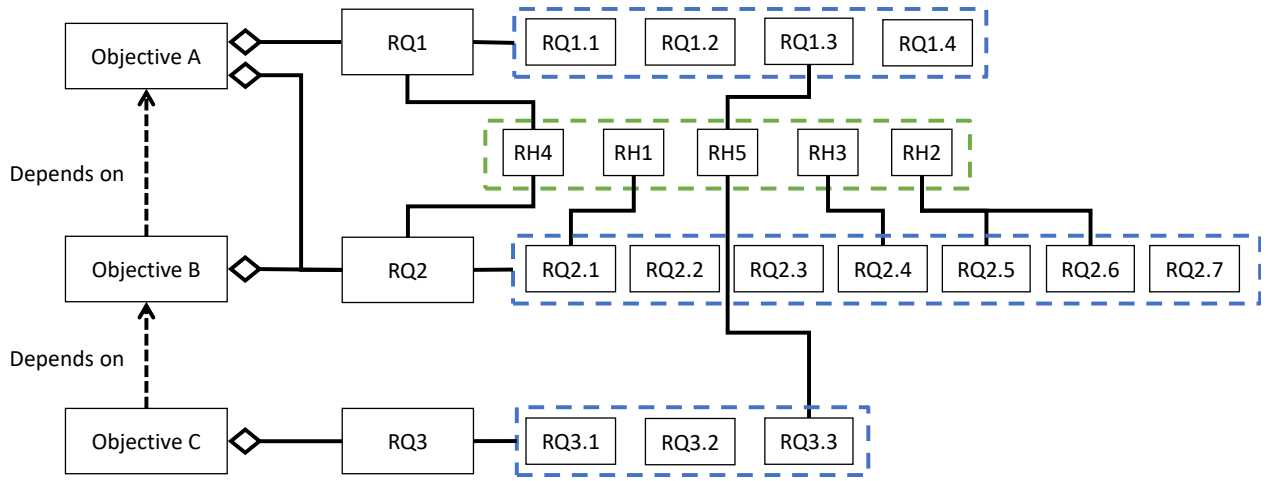


Figure 1.6 UML class diagram describing the relation between the research objectives, questions, and hypotheses

1.4 Research questions

Three main research questions are addressed in this research. Additional sub-questions are investigated throughout the study to facilitate answering the main questions.

RQ1 How to differentiate between DS and DT?

RQ1.1 What are the main functionalities of DS and DT?

RQ1.2 How to distinguish between data acquisition, modeling, and decision support?

RQ1.3 What are the main differences between DS models and DT applications?

RQ1.4 How to transform simulation models into DT applications?

RQ2 How to develop DTs using AI and distributed system concepts?

RQ2.1 Can multiple independent DMs be used to provide a multi-physics multi-scale description of the physical system?

RQ2.2 Since simulation models run faster than real-time, how to control the time flow of time-dependent models such as continuous-time models and Discrete event simulation (DES)?

RQ2.3 DS models track the physical system, therefore synchronizing DMs with the real system's state is crucial. How to synchronize DMs with the physical system?

RQ2.4 How virtualization can be applied to DMs?

RQ2.5 How to connect a DM to the Pub/Sub communication channel?

RQ2.6 Does Pub/Sub communication channel provide real-time messaging capabilities for DTs?

RQ2.7 How to embed RL into simulation models?

RQ3 How to iteratively develop and improve DTs?

RQ3.1 What are the requirements for a physical system to be DT compatible?

RQ3.2 How to systematically develop DTs?

RQ3.3 How to run multiple DT applications concurrently using the same DS model?

1.5 Research hypotheses

The research work is based on the following hypotheses:

RH1 Multi-physics simulation tools are domain-specific and expensive. Co-simulation is an alternative approach to combine multiple standalone sub-simulators, where system description is distributed over multiple independent models. Co-simulation approaches, such as Functional mock-up interface (FMI), handle the execution and interfacing with these models.

We assume that loosely-coupled DMs are capable of generating a multi-physics multi-scale description of the physical system. Defining a physical system as a system of systems reduces the DM design complexity. DS then works as a co-simulator that integrates multiple DMs.

RH2 Pub/Sub communication systems proven to have high-throughput low-latency. Moreover, Pub/Sub communication systems, such as Apache Kafka®, provide fault tolerance capabilities in order to avoid data loss due to network faults.

We assume that Pub/Sub communication systems provide a fast reliable inter-communication channel to pass information between different DT components.

RH3 Containerization of DMs is important for DM integration into the DT framework.

We assume that given the needed library dependencies, simulation models are virtualized through programable interfaces, vendor support, or virtualizing the models' executable version.

RH4 Model development is an iterative process, where continuous improvement of the model is expected to reduce the gap between reality and simulation. Hence, DMs developed for a DT are continuously modified to adapt to system changes and new situations.

We assume that orchestration systems, such as Kubernetes, handle the deployment of new versions of a DM through tracking the DM's corresponding container tag name. Each version of a DM has a unique tag name. DT design team selects the DM version by defining the container tag name at the orchestration system.

RH5 Decision-support tools' objectives vary widely. For instance, virtual commissioning (VC) simulates the system behavior to predict the consequences of changing the production process or dispatching new products. While quality inspection tools track equipment and product inspection checks over time in order to adjust machine settings and schedule maintenance operations.

The scope and objectives of a DT application determine how the application is implemented and used. We assume that DT applications are categorized into continuous and Ad-Hoc applications. The distinction was raised by the implementation differences between the two types. Continuous DT applications run in parallel to the DS in order to notify decision-makers of trends, inconsistencies, or unexpected situations. While Ad-Hoc DT applications start execution upon a decision-maker request.

1.6 Points of originality

The originality of this research is described as follows:

- Role-based separation between DS and DT.

In Chapter 4, DS and DT concepts are distinguished by their functionalities and roles in the decision-making process. The role-based separation approach clarifies the differences between both concepts and their relationship. DS uses models and simulation to generate a digital replica of the real world. While DT uses data analysis tools, simulation, and AI to support decision-makers.

- Defining DS as a core component of a DT.

In Chapter 4 and 7, a DT framework is proposed with DS as its core component. The use of DS as a component of the DT is a novel concept, which differs from the current schools of thought. DS is used to generate a digital representation of the physical system. While DT uses the generated representation to provide decision-support capabilities.

- The use of distributed system concepts as enabling technologies of general-purpose DT framework.

In Chapter 6 and 7, distributed system concepts are used to achieve a general-purpose DT. Despite the importance and benefits of distributed systems in the computer engineering domain, these concepts are still hardly used in existing DT frameworks. The proposed DT framework utilizes distributed system concepts such as Microservices, Pub/Sub communication, virtualization, and orchestration.

- Proposing a systematic approach for designing Digital models for DTs.

In Chapter 6 and 7, digital model development for DS models and DT applications are investigated. The instantiation process and initialization of DMs are defined. While the use of Digital model manager (DMM) is proposed to synchronize DM with the physical system and control its time flow. DMM also provides a systemized interface to communicate with its corresponding DM.

- Containerization of DMs.

In Chapter 6, the use of dockers to containerize DM enables the integration of DMs from different vendors into the same DT architecture. Microservices and Kafka client API allow the interaction between the containerized DM and other DT components.

- Modularization of DM and information sharing between DMs.

In Chapter 6, the use of Pub/Sub communication channel systemize information sharing between DMs. Tag-based messages add context to the information being shared. DMs receives the needed information by subscribing to a list of message tags at the Pub/Sub communication channel.

- Proving the real-time capabilities of Apache Kafka® Pub/Sub communication system to operate in a DT.

In Chapter 6, Kafka throughput and latency test is performed to prove the feasibility of using Apache Kafka® as a Pub/Sub communication system in a DT. The test results show Kafka's ability to pass information in real-time between multiple DT components.

- The definition of CPS infrastructure requirements needed to develop a DT.

In Chapter 8, the communication infrastructure required to develop DTs is investigated, where the transformation of the physical system into a CPS is proposed as a necessity. The definition of communication infrastructure requirements and the use of IEC 61784-2 standardized Key Performance Indicators (KPIs) allows the verification of DT's real-time capabilities.

- Proposing an agile-driven stepwise development methodology for DTs.

In Chapter 8, agile development is adopted to propose a DT development methodology. The proposed methodology identifies the main steps to design, implement, and deploy DTs. Additionally, the adoption of agile development enables the iterative development and improvement of DS models and DT applications.

CHAPTER 2 THESIS ORGANIZATION

The main research objective is achieved through 4 phases of research: conceptual design of the DT framework using distributed system concepts; development of a real-time DS framework; development of DS-based DT framework; designing a DT development methodology for applying the proposed DT framework. Figure 2.1 shows the research articles' sequence throughout the phases of this research. While Table 2.1 shows the research objectives and questions targeted by each research article.

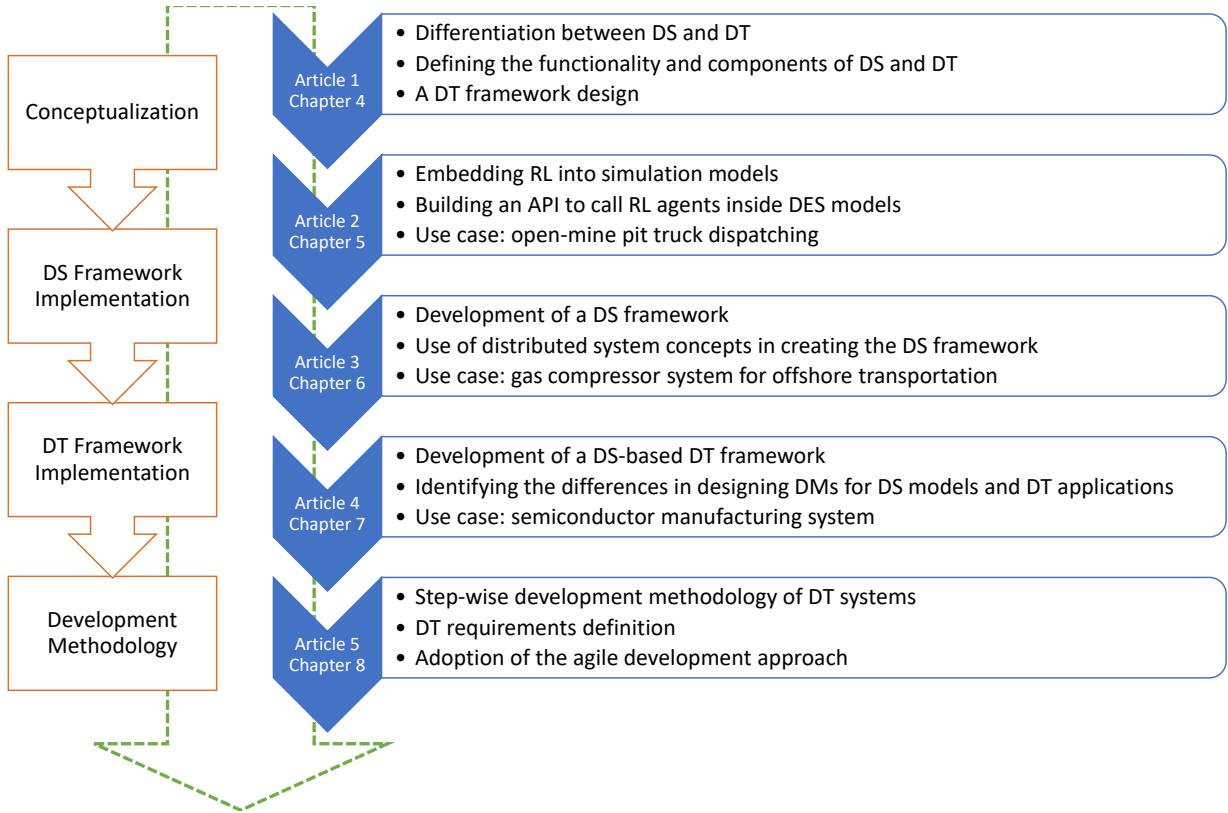


Figure 2.1 Graphical description of the thesis organization

In the first phase, the DT framework design is conceptualized. Chapter 4 addresses *RQ1.1* and *RQ1.2* by identifying the distinctions between DS and DT roles and functionalities. The conceptual design provides a roadmap for achieving both objectives *A* and *B*. In Chapter 4, the DT framework functionality and components are identified. DT concept has several definitions in literature due to the concept's generality and lack of standardization. Other concepts such as CPS and DS are intertwined in DT. Chapter 4 proposes the separation between DS and DT in terms of functionality. DS is defined as a real-time virtual replica of

Table 2.1 Research objectives and questions addressed by the research articles

Article	Objectives	Questions
Article 1	A, B	RQ1
Article 2		RQ2
Article 3	A	RQ1-2
Article 4	B	RQ1-3
Article 5	C	RQ3

a physical system. While DT is defined as a real-time decision-support sandbox. Using this definition, the roles of DS and DT are identified. Additionally, A conceptual DT framework is designed alongside the description of its components, where DS is defined as a core component of a DT.

Chapter 5 addresses *RQ2.7* in order to train and deploy RL agents into DT applications. The definition of key enabling technologies is part of the DT designing process. In Chapter 5, embedding AI into DES simulation models is studied. Traditional RL approach uses simulation models as virtual environments for RL agents' training. It considers simulation models as black boxes used for RL training only. DT highlights the importance of simulation models and their role in digitizing the physical system. Hence, Chapter 5 proposes the embedding of RL agents into simulation models in a white-box approach. Simulation models call RL agents when a decision is needed. RL agents are trained during model execution using status variables received from the model. In Chapter 5, a RL library is developed inside the simulation tool. The RL library is designed for model designers who might not be experts in the RL domain. The RL embedding approach is tested on a real-world truck dispatching problem for an open-mine pit.

In the second phase, a real-time DS framework is developed. Chapter 6 addresses *RQ1* and *RQ2* through the development of a DS base for future DT applications. Chapter 6 applies distributed system concepts in designing and implementing a DS framework to achieve objective *A*. The DS framework uses multiple Digital models to generate a digital representation of the physical system. DMs captures the system behavior from multiple perspectives using different abstraction levels. Gateways collect real-time data from the physical system. While Pub/Sub communication system shares information between different DS components. SOA, containerization, and orchestration enable the DS framework to execute and integrate multiple loosely-coupled DMs. The DS framework is applied to a natural gas compression system for an offshore gas transportation pipeline. Five DMs were developed to describe different aspects of the compression system. The use case study proved the framework's ability to integrate different simulation tools and provide a real-time representation of the system.

In the third phase, a general-purpose DT framework is developed. Chapter 7 builds upon the DS framework developed in Chapter 6. Chapter 7 continues addressing *RQ1*, *RQ2*, and *RQ3.3* to achieve objective *B*. In Chapter 7, the DS framework is used as the core component of the DT framework. The real-time digital image provided by DS is used by DT applications to identify the current system state. DT applications use simulation models, ML models, and data analysis tools to provide insights and recommendations for decision-makers. Additionally, Chapter 7 discussed the differences between simulation models deployed in DS and their counterparts used in DT. The DT framework is applied to a semiconductor production system. Two DMs are used as DS models to describe the system's behavior. While Three DT applications are developed to assist operators and decision-makers. The case study proved the framework's generality and the potential opportunities for applying AI and distributed systems in the industrial domain.

In the final phase, the DT development process is addressed. Chapter 8 addresses *RQ3* through adopting agile development to achieve objective *C*. Chapter 8 proposes a DT development methodology to transform existing physical systems to CPS systems and create its DT. The proposed methodology defines the process of DT development as a five phases project. The objectives and scope of the DT are first defined in the planning phase, alongside the physical system description. Then the physical system is transformed into a CPS in the transformation phase. Afterward, Digital models of the CPS system are designed, implemented, and verified in the modeling phase. These DMs are converted into DS models and iteratively refined in the shadowing phase. Finally, DT applications are integrated into the DT in the twining phase. The methodology adopts an agile development approach to maintain the stakeholders' involvement in all project phases, and enable continuous improvement of the DT.

CHAPTER 3 LITERATURE REVIEW

3.1 Digital Twin

The number of publications in DT field is growing exponentially [13, 48]. DT publications range from studying DT concept and technology to DT applications. The broadness of DT concept created multiple DT definitions [47, 58]. Some studies focused on DT's real-time monitoring [48, 49], where DT collects real-time data from multiple sources for data analysis and visualization. Other studies focused on DT's simulation of the physical system [15, 18, 27]. Others studied the dataflow and connectivity between multiple DTs components [16, 25, 50, 51]

3.1.1 Digital Twin Definition

The twinning concept was proposed in the 1960s [13]. NASA used a physical replica of the apollo space shuttle in replicating the shuttle's operations. An on-ground replica is used for shuttle simulation and operator training. In 2010, NASA defined DT as an objective in its 2010 technological roadmap [17]. NASA engineers defined DT as a real-time multi-scale multi-physics simulation replica used for real-time prediction of faults in the real vehicle [17, 18]. They discussed the need for a complex real-time self-aware system in deep space missions, where human intervention is inapplicable.

In the manufacturing domain, Grieves and Vickers [16] proposed a similar concept called *virtual product* in 2003 as a PLM concept [14]. A *virtual product* collects physical product information over the product's lifecycle. *Virtual product* is used in managing and optimizing product design.

The DT concept did not gain momentum till the start of the industry 4.0 era. DT opportunities in the industrial domain were discussed by Rosen et al. [20]. DT is used in industry to describe a real-time DM of a product, component, system, or system of systems. That model captures the needed information for knowledge extraction and decision-making [14].

Multiple definitions of the DT concept were studied in the industry 4.0 literature. DT is widely defined as a digital replica of a product, equipment, system, or system of systems [14, 15]. Rosen et al. [20] defined DT as the combination of physical and virtual systems where a change in one is reflected in the other. Grieves and Vickers [16] proposed DT as the collection of information providing a virtual representation of the product during its lifecycle. Tao et al. [69] compared between DT and CPS, where they defined DT as an application-driven concept, while CPS is an academic-driven concept. Brauner et al. [51] defined DT as

a digital representation of objects and processes of the real world.

DT systems are categorized according to different points of interest. Uhlenkamp et al. [14], Stark and Damerau [15], and Qi et al. [27] categorized DT systems based on their dimensions, components and properties. Schroeder et al. [28] and Phua et al. [34] categorized DT based on their functionality and proximity to the physical system.

3.1.2 Digital Twin Frameworks

Yun et al. [46] proposed a 3-tier DT framework for self-driving cars; The physical system, data middleware, and service layer. DT is designed as a middleware for real-time data collection. The physical system is connected to DT through Data Distribution Service (DDS) communication protocol. The data-based middleware layer collects the information from the physical system in real-time. The service layer simulates the system using the received data from the middleware layer. A simulation interface module is used to synchronize the simulation time with the real-time, and to exchange information with the DT and other simulations. FMI is used to encapsulate the simulators and provide a standard interface.

Tao et al. [70] proposed a DT framework under the *Made in China 2025* initiative. While they discussed the potential benefits of using DT in PLM operations in [26]. Their framework divided the DT system into 4 intercommunicating components; physical shopfloor, virtual shopfloor, services, and DT data. The physical shopfloor includes human workers, physical machines, products, and the environment. These physical components are connected to the rest of the DT system through a network of collaborative smart nodes. This network of nodes collects real-time data, process them, then sends the processed data to other DT components. The virtual shopfloor is the collection of expert-based digital models that keeps a global virtual state of the physical system. The digital models describe the physical shopfloor components logically and geometrically. These models are developed using 3D CAD, Finite Element Model (FEM), state diagrams, and rule-based models. Shopfloor services system is the collection of DT applications and tools such as simulation, visualization, production planning, and model accuracy evaluation. Shopfloor DT data is the collection of information received from all DT components.

Zhuang et al. [62] also proposed a DT framework similar to Tao et al. [70]. However, they focused on the use of Radio-Frequency Identification (RFID) for tracking both products and materials. They applied their framework to a satellite assembly line for production management and scheduling.

Park et al. [25] proposed a 3-level DT framework; physical level, services level, and application

level. The physical level contains the physical system including controllers and sensors. The services level includes the DT models and internal services. The application level represents all decision-support applications. In their framework, they separate between DT basic services performed internally as simulation and monitoring; and decision-support applications executed upon user request such as scheduling maintenance and optimizing production plans. Application level interacts with the DT using HTTP request/response. They also proposed the modeling of system components separately. The models are encapsulated using FMI to standardize the communication interface using a multi-timing system. Models with time-synchronous components communicate directly, while non-synchronous communicate using message queues.

Qamsane et al. [71] proposed a DT framework in order to provide a generalized DT framework. They claimed that simulation-only and model-only approaches provide analytical capabilities; however, these approaches do not provide a fully-functional DT system. Their framework divided the system into 3 layers; the physical layer, *software-defined controller* layer, and the application layer. DT is part of the *software-defined controller* layer. DT communicates with the physical system through an OPC Unified Architecture (OPC-UA) interface, while it communicates with the high-level applications, Manufacturing Execution System (MES), and Enterprise Resource Planning (ERP) using API interfaces. DT provides both data modeling and simulation through two separate modules. A DT manager controls the creation of DT model instances and execution of simulations. The simulators are encapsulated using FMI in order to enable the integration of different simulation tools. While DT models are created using mathematical models, topology models, finite state machines, differential equations, or discrete event models. They use multiple collaborative models to describe and track the physical system. These models communicate with each other and with the DT manager through a publish/subscribe communication system. Data shared between the models is formatted into XML by the message handler inside the model instance. They call the collection of deployed models a *digital twin pool*.

Previously discussed DT frameworks use existing standards for communication, data security, modeling, and data storage. However, standardization of DT frameworks is still an ongoing process. ISO 23247 [59] is a generic guideline for developing DT systems. ISO 23247 provides some DT implementation case studies for specific systems. However, ISO 23247 focuses on the definition of Observable Manufacturing Element (OME) and how to connect OME to DT system [61]. IEC 62832 [60] provides a high-level digital factory framework for maintaining a digital representation of a production system. IEC 62832 focuses on information sharing between different components of a digital factory. Some studies proposed the implementation of DT based on RAMI 4.0 as a part of AAS [55–58].

3.1.3 Digital Twin Review Papers

Uhlenkamp et al. [14] conducted an extensive study on the aspects of a DT framework. Cimino et al. [8] extensively studied published papers focusing on DT applications. Qi et al. [27] defined a 5 dimensions characterization of DTs and discussed the DT's enabling technological tools. Lu et al. [23] studied the technologies applied in the DT field. Tao et al. [69] comparatively studied the relationship between DT and CPS.

3.2 Digital Shadow

Digital shadow (DS) is an artifact concept of DT, which was first defined by WGP in 2016 [29–31]. WGP defines DS as a digital representation of the physical system formed using real-time information collected from multiple sources. DS does not affect the physical system. Operators utilize the DS aggregated information for system understanding and decision-making. WGP separates between DT and DS in terms of model fidelity [8, 30, 31].

Becker et al. [72] used model fidelity to differentiation between DT and DS. DT is defined as a high-fidelity replica of the physical system. While DS is a compact purpose-driven model representing part of the physical system.

Ladj et al. [73] adapted a similar definition of DS. They defined DS as a core component of DT. DS consists of ontology-based data models that capture the physical system state. DS uses data-processing algorithms to aggregate data from different sources and formats to provide a dynamic database for DT applications. Brecher et al. [74] proposed the use of a DS Caster that annotates raw data. DS Caster then updates the ontology-based data model. That definition limits DS role to data collection and processing, where engineering knowledge is added as predefined relational rules only.

Brauner et al. [51] also adapted a DS definition similar to WGP. They consider true DT as infeasible since it is hard to fully represent a physical system in a digital form. Additionally, DT requires a massive amount of data and computations to build a virtual replica, which results in high-latency operations. DS is defined as a task-driven domain-specific simulation model that describes part of the physical system in real-time. DS aggregates raw data in a single model, that is processed to extract and share knowledge. A system could have multiple DSs, each describing a part of that system.

Stark et al. [7, 15] proposed an intersected functionality between DT and DS. DS is defined as the collection of system data, while DT is defined as the digital model and its DS. DS's role in that definition is limited to real-time data acquisition.

Other studies differentiated between DT and DS according to data-flow direction. DS is a one-directional digital model that receives real-time information from the physical system. Schuh et al. [75] proposed the use of DS as a digitalization of the real manufacturing process, where a virtual copy of the factory is created using simulation models. Konstantinov et al. [76] proposed Web-based Digital Twin (WDT) is used as a DS that keeps track of the physical system, which is accessed through web browsers by technicians. Kassen et al. [77] designed data-driven DS simulation model to accommodate different types of production systems.

3.3 Distributed Systems in Industrial Domain

IoT, AI, and cloud technologies are playing the main role in shaping the fourth industrial revolution. Industry 4.0 emphasized the potential benefits of the digitalization of the factory [6]. AI and distributed systems have been widely studied in the software engineering discipline. IoT components are designed as application-specific low-power hardware. Cloud computing provides the required computational and storage power for industry 4.0 systems.

3.3.1 Service Oriented Architecture

Industry 4.0 reference architectures utilize SOA and microservices [53]. Industrial internet reference architecture (IIRA) uses SOA concepts for dynamic exchange of information. RAMI 4.0 uses SOA encapsulation concept to provide a common interface between its modules. Stuttgart IT-Architecture for Manufacturing (SITAM) is built on SOA architecture that defines the system as a hierarchical structure of services. Smaller initiatives such as FIWARE and ArrowHead frameworks adopted SOA for providing data-centric architectures.

Catarci et al. [55] proposed the design of DT models using SOA approach. DT collects information from the system in the form of vendor-based semantic models. The collected data are integrated and translated to a common representation by a *mediator* module. The translated information is then shared with CPS as a web-based service. DT sends data upon query request from synchronized systems or using Pub/Sub message system.

Dobaj et al. [78] proposed the development of DT as a SOA middleware component in a CPS architecture. CPS applications request information from DT level, without the need to know the physical hardware and network on the shopfloor. DT collects the shopfloor information into a global holistic view. DT provides data aggregation, service orchestration, and communication abstraction.

3.3.2 Event-Driven Architecture

DT frameworks utilize communication systems to connect different DT components. Message Queue (MQ) is an asynchronous message channel that transfers information from one component to another. MQ provides low-latency communication for one-to-one information sharing. However, MQ does not support multiple subscribers for the same message. Some MQ technologies, such as Google Pub/Sub, duplicate the message as a workaround to send the same message to multiple subscribers. Different MQ technologies are used in DT frameworks such as MQTT [25, 56, 79], MTConnect [80], AMQP [79], and RabbitMQ [50].

Pub/Sub communication systems are used in some DT frameworks as in Tao and Zhang [70] and Qamsane et al. [71]. Pub/Sub communication provides a global many-to-many information-sharing platform. DT components do not interact directly. DT publishers send tag-named messages to the Pub/Sub communication brokers. DT subscribers then receive messages based on the subscribed tag names.

Apache Kafka® is an open-source Pub/Sub communication system that has been used in DT applications in the literature. López [81] and Alaasam et al. [82] discussed the potential benefits of event-driven architectures and Pub/Sub communication. They discussed the importance of Kafka for system flexibility and scalability. Radchenko et al. [83] used Kafka to pass information to Kepler workflows in order to model the system as a structured process flow. Kafka is used as a middleware layer for collecting information from IoT sensors and passing it to Kepler workflows. Kamath et al. [79] used Kafka with InfluxDB and Grafana for time-series analysis of collected data from the real world. Kafka provided a real-time streaming pipeline to pass information from DT to analytical tools. Message queues such as MQTT and AMQP are used to collect information from the physical system. Steindl and Kastner [84] used Kafka as a backbone communication channel to connect different components of their DT system.

3.3.3 Containerization

Containerization is not widely used in the industrial domain. Radchenko et al. [83] used Docker to automate the deployment of their DT system. Docker images contained the deployment instructions for Kafka and Kepler workflows. Redeker et al. [56] proposed the use of Continuous deployment (CD) tools over Kubernetes (K8s) to enable DT system updatability. They used CD workflows to continuously test and deploy new AAS modules when modified by designers. While K8s update existing deployments without disrupting the service. Steindl and Kastner [84] discussed the importance of containerization and microservices.

3.4 Literature Synthesis

The standardization of DT is still an ongoing process. Multiple DT definitions have been proposed in the literature. These definitions agree upon the real-time capabilities of a DT and its goal to create a digital image of the real system. However, DT functionalities and roles are not clear. Moreover, the goals of other concepts such as CPS, AAS, and DS overlap with the DT. In this research study, DT concept is defined based on its roles and functionalities. This approach separates between DT and other artificial concepts as DS.

Multiple DT frameworks were proposed and discussed. A common attribute of these frameworks is designing DTs using multiple models generating a holistic view of the physical system. Distributed system concepts such as cloud computing are used in deploying DT frameworks. While SOA is used in industry 4.0 reference architectures such as IIRA, RAMI 4.0, and SITAM. However, the use of distributed systems is still limited to cloud deployment. Distributed system concepts, such as microservices, virtualization, and orchestration, enable DT capabilities. In this research study, the use of distributed system concepts is targeted to design, develop, and deploy DTs.

CHAPTER 4 ARTICLE 1: A FRAMEWORK FOR DIGITAL TWIN DEPLOYMENT IN PRODUCTION SYSTEMS

Ayman AboElHassan, Ahmed Sakr, Soumaya Yacout

Published in:

Proceedings of the Stuttgart Conference on Automotive Production (SCAP2020),
Stuttgart, Germany, November 9-10, 2020.

Part of ARENA2036 series: Advances in Automotive Production Technology – Theory and
Application.

Date of publication: 02-Jun 2021

4.1 Abstract

Digital twins represent physical systems through dynamic adaptive digital replicas. These replicas are virtual images of the functionality and interactions of the physical system and its components. Digital twin provides real-time monitoring and decision-making support. These are essential pillars in the Industry 4.0 paradigm. On the system level, multiple architectures are established for digital twin concepts in the literature. However, the roadmap for deploying a functional digital twin has not been fully recognized thus far. In this paper, we propose a framework for the deployment of a digital twin in production systems. The framework covers both levels of virtualization; digital shadowing, and digital twinning. It utilizes present technologies in network communication, data management, and knowledge extraction. We describe the main components of the digital twin, and their relation to the management schemes currently implemented in production systems. Additionally, we devise an approach for the integration of available simulation and data analytic tools for dynamic modeling and system performance evaluation. Through our proposed framework, the current technologies and tools are capable of deploying a digital twin. Consequently, learning algorithms and monitoring procedures are exploited for dynamic performance improvement.

Keywords

Industry 4.0, Digital Twin, Digital Shadow, Framework, Decision-Making Support

4.2 Introduction

Highly competitive markets and short product lifecycle push the current production systems to adapt more proactive production plans. Computer-aided planning provides the functionalities to analyze production systems in real-time and perform on-site optimization to meet production demands [85]. Moreover, customer-centric production lines need to schedule assembly-line operations for personalized products manufacturing. This need for real-time modification and resource allocation of production plans raises up the concept of cyber-physical production systems (CPPS) [86]. DT has been a highly active research topic lately because of its real-time analysis and managing capabilities. The emergence of ICT, and IoT in industrial manufacturing domain created potential economic value for the DT technology [85]. The role of DT in CPPS is to provide the real-time status of the physical system. This enables it to perform scheduling, simulation, and optimization to manage the production proactively.

DT term is usually used to define three different concepts [14]; product twin for PLM, production twin for customer-centric production systems, and system twin for CPPS.

4.2.1 Production Systems

Despite the massive focus on Industry4.0 topics in the academic domain, the industry is still studying the needed changes in their infrastructure to adopt the Industry4.0 technologies. In-service manufacturing systems still utilize the Purdue automation pyramid [7, 8]. These businesses are lagging behind due to the high cost of upgrading the manufacturing lines. The upgrading costs include installation cost, production plan interruptions, and production line stoppages.

Large investments have been specified for DT technology. Newly constructed production facilities are best suited for Industry4.0 concepts as they utilize state-of-the-art equipment. However, SME are more adaptive than standard large enterprises and complex businesses because of governmental initiatives [9].

Current production systems depend on central SCADA server to collect information from shop-floor machines and send control commands to these machines. As production systems currently use industrial Ethernet, most equipment communicate with the SCADA server through OPC-UA protocol [71]. The central server stores the collected data into Historian database. Human operators interact with the machines from on-site terminals, or off-site connection to the SCADA server. System analysis is usually performed offline on the data collected and extracted from the historian database. Figure4.1 shows the system architecture of data gathering, management, and planning in traditional production systems.

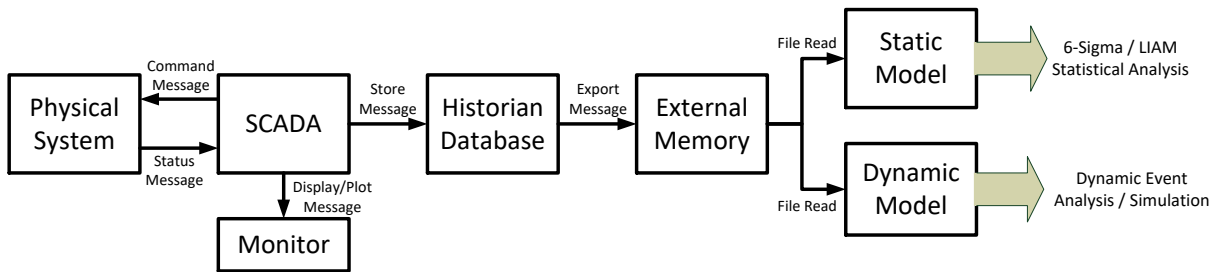


Figure 4.1 Traditional Approach for data gathering, management, and planning

4.2.2 Related Work

The number of publications are increasing rapidly in the field of Digital twin. Most of the publications concentrated on DT applications [8]. While, some publications focused on the development of the DT framework [25, 46, 70, 71].

The differentiation between DT and DS terms is still an ongoing discussion. Two main groups of thoughts exist; The first opinion differentiate between the two terms according to the data-flow direction between the virtual and physical worlds [8, 14]. In Digital shadow, the physical world sends data to its corresponding virtual model. While in Digital twin, the data-flow is bidirectional, from/to the physical world. This definition tends to describe DT as an upgraded version of DS. The second opinion uses Digital twin term as a high-fidelity model, compared to the low-fidelity of Digital shadow [8].

However, there is no clear separation between the roles of DS and DT in literature. Schuh et al. [75] defined DS a digitalization of the real manufacturing process, where a virtual copy of the factory is created using simulation models. However, they did not discuss the difference between DS and DT. While, Stark et al. [7] proposed an intersected functionality of DS and DT, for which DS term is used as the change in the digital model of the physical system over time. DT term refers to an instance of the digital model and its shadow. Where DT instance is used in system analysis and simulation.

4.2.3 Contribution

In this paper, we differentiate between DS and DT in terms of objectives and components. Digital shadow is defined as *a model-based virtual state of the physical world collected in real-time*. While digital twin is defined as *real-time decision-support sandbox based on digital shadow*. We use DS as a component of DT, which provides real-time state to DT functionalities. DS is connected to the physical world using current communication technologies. The main goal of using currently utilized communication technologies is to reduce the DT installation cost and encourage complex production systems to adapt Industry4.0 paradigm.

The rest of the paper is organized as follows; The proposed framework for DS and DT is presented in Sect. 4.3 and 4.4 respectively. The framework integration with the production system's components is discussed in Sect. 4.5. Finally, the conclusion is presented in Sect. 4.6.

4.3 Digital Shadow

Digital shadow (DS) is defined as a digital replica of a physical component, system, or system of systems. This digital replica matches its physical counterpart's state in real-time. DM describes the physical world's behavior in response to internal and external factors through time. Hence, we can define a DM as a mathematical model, behavioral model, data-driven model, or even a collection of data. The goal is to mirror the physical world in a digital form. We can summarize the objectives of a DS as follows;

1. Providing an abstract model for the physical system.
2. Providing the state of the physical system in real-time.
3. Visualize the virtual state of the system.
4. Compare the virtual state to the physical state.
5. Update the physical system's configuration in the abstract model.
6. Update the abstract model to match the physical system's behavior.

DS is divided into components according to functionality. The DS architecture is developed as a service as shown in Figure 4.2. Each component is a service that interacts with other components. These interactions take the form of publish/subscribe messaging.

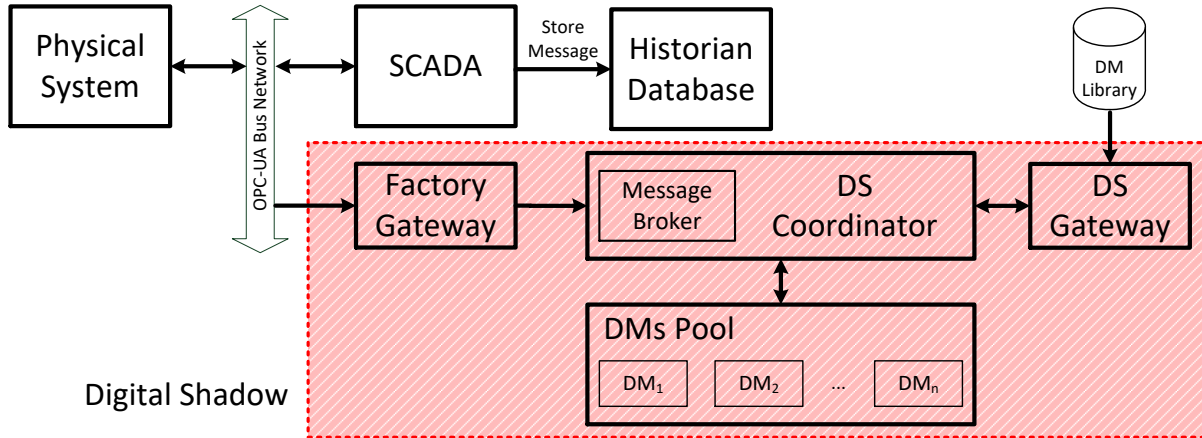


Figure 4.2 A Digital Shadow Architecture

Factory Gateway

is an OPC-UA client application, which is installed on a small board. The gateway is connected to the SCADA server through a bus network topology. Upon receiving an OPC-UA message, the gateway converts the message into a set of <key, value> messages that is sent to the DS coordinator.

The use of passive data collecting technique allows the DS to read OPC-UA messages that the SCADA server receives. Accordingly, the SCADA server is not interrupted or requested to send data to the DS all the time. This approach reduces the overhead communication of passing all the manufacturing facility's information from the SCADA server to higher layers in real-time [87].

DS Coordinator

manages the communication between the different DS components. The coordinator's message broker works as a middleware that relays messages from publishing components to subscribing DMs. The factory gateway publishes all received data onto the message broker, while DMs publish their virtual states. Each DM subscribes for the physical and virtual data that it needs. Moreover, the coordinator maintains a global state of the physical and virtual system from all the models in the DS pool. The global state is synchronized across all DMs using logical time synchronization. The DMs system time is synchronized by the coordinator using network time protocol.

DS Pool

is a collection of all DM objects that are running in the DS framework. A DM object is initiated by the DM pool manager. On construction, the DM object subscribes to a list of required keys. Then, it sends a synchronization request to the coordinator. Upon synchronization, the DM runs the model in real-time mode.

DM library

is a database of all the DMs source models. A DM object is created from one of the models provided by the library. Moreover, the library tracks the version of each DM source model. This enables the modification of an initiated DM during runtime.

DS Gateway

provides access to the global physical and virtual states. DT and user interface send data requests to the DS gateway using HTTP request protocols as a query for list of keys. The gateway then responds by the requested data from the global state provided by the coordinator. The requested data keys can refer to values published by the physical system or DMs.

4.4 Digital Twin

Digital twin (DT) is defined as a real-time decision-support sandbox, which uses real-time digital representation of the physical component, system, or system of systems. By that definition, DS is used by DT to represent the physical world in real-time. While, DT provides a higher-level intelligence. We define intelligence in this context as any knowledge-based application that uses simulation, scheduling, searching, optimization, diagnosis, prognosis, and performance evaluation to help the decision maker in improving the efficiency of the production system. Intelligence applications can range from basic pre-programmed calculations, to AI and ML.

The separation between the virtual model in DS and intelligence in DT, eases the adaptability of the DT framework for new applications. DT applications read the state and model through a systematic approach. However, this generalization requires DMs to provide both real-time and simulation-time capabilities. This simulation-time speed differs from DS speed, which is required to match the physical world's pace. DT uses the DS real-time state to apply one or more of the following objectives:

1. Analyze the physical/virtual system's performance
2. Initialize simulation tools with the current virtual state.
3. Evaluate production policies.
4. Optimize shop-floor schedules.
5. Optimize maintenance plan.
6. Estimate an order price and delivery date.
7. VC of equipment and/or product.

The DT framework is divided into components as shown in Figure4.3. Each component is an object that interacts with other components in a form of request/response messages.

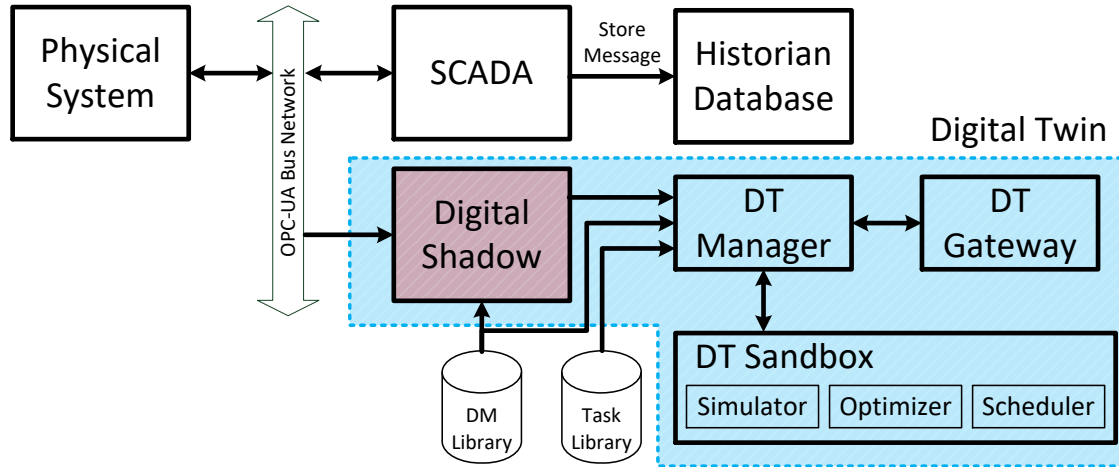


Figure 4.3 A Digital Twin Architecture

DT Manager

coordinates between the different components of the DT's framework. The manager provides the needed data to the DT sandbox and pass its results to the user. The manager loads DMs from the DM library with their DS-configuration, in order to create a real representation instance for intelligence tasks.

DT Sandbox

is a workspace for intelligence tasks. The sandbox provides necessary tools for these tasks as behavioral simulators, physics simulators, optimizers, and schedulers. Each tool is integrated into the sandbox using an API interface, which enables loading the tool with models and initial states. The sandbox initiates independent process for each task upon user request. Then, the tasks are initialized by real representation instances. Finally, task results are provided to the user through the DT manager.

Task Library

is a set of intelligence scripts and procedures that use simulation, optimization, and machine learning to perform one of the intended objectives of the DT. A task script is executed by the DT sandbox. However, the task script is called from the DT manager. The task script specifies the tool to be used from the DT sandbox, the tool settings, and the DMs to be used in the run. Historical data can be used by the DT sandbox tools.

DT Gateway

provides a user interface for the DT functionalities. Users access DT manager through a web interface using HTTP requests. This interface provides the user with the capability to view the real-time state, run intelligent tasks, review task results, and modify the DM library and task library.

4.5 Integration in Automation Process

In this section, we discuss two main aspects of the DT framework and its interaction with the other components of the automation pyramid; the feedback to the physical world, and the integration with MES/ERP.

4.5.1 Feedback to Physical World

The proposed DT framework does not explicitly define a way to control the physical system. However, the addition of a feedback connection to the SCADA server can be done through the DT gateway. In that sense, a DT system can be categorized as controlling or non-controlling. Non-controlling DT generates data insights and recommendations only. While, controlling DT uses a decision-maker module that controls the physical world through the SCADA server, as shown in Figure 4.4.

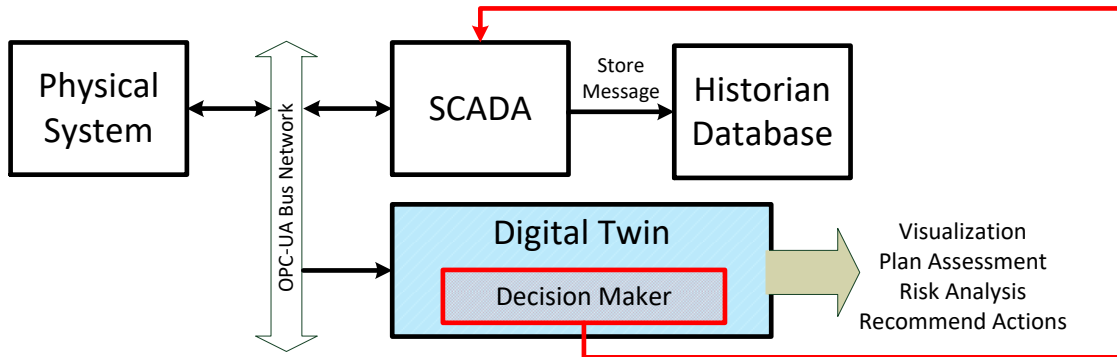


Figure 4.4 A controlling Digital Twin

4.5.2 Integration with MES/ERP

DT differs from MES and ERP. It uses the real representation of the physical system with its current state. While MES/ERP uses nominal representation of the system, which induces higher abstraction error. The DT framework extends the MES/ERP by providing multiple

operations as order dispatching, scheduling, virtual commissioning, maintenance planning, resource management, performance prediction, and estimation of price and delivery time. A MES/ERP module can directly command the DT framework through the DT gateway. The common scope between all DT applications is the use of the current state of the manufacturing facility as an initial state of the intelligence tool. Moreover, real representation of each component is used instead of nominal ones.

4.6 Conclusion

In this paper, we proposed the separation of functionalities between Digital shadow and Digital twin to describe a framework for DT deployment. Determination of roles provides a clearer view of the DT concept. Additionally, the separation of digital models from the higher intelligence allows wide range of DT applications to be applied on the same framework. We proposed a DT framework based on message-oriented communication between its components. The proposed framework provides real-time representation to the user and other high-level applications. Moreover, it provides a generic approach to extend the DT with new applications.

CHAPTER 5 ARTICLE 2: EMBEDDING REINFORCEMENT LEARNING IN SIMULATION

Ayman AboElHassan, Soumaya Yacout

Published in:

Proceedings of 26th IEEE International Conference on Emerging Technologies and Factory
Automation, ETFA, Västerås, Sweden, September 7-10, 2021.

Date of publication: 30-Nov 2021

5.1 Abstract

Reinforcement learning (RL) usage in the industrial domain is on the rise since the shift towards Industry4.0 systems. The need for faster adaptive systems is encouraging manufacturers to invest more in artificial intelligence (AI) technologies. Our aim is to add better intelligence into simulation tools by embedding RL capabilities. Discrete event simulations have been used for decision support in manufacturing systems for decades. New simulation tools such as AnyLogic have improved significantly in the past few years. In this paper, we built a RL library for AnyLogic simulation models. The RL library is developed for model designers, who may not be experts in the field of RL. We applied our RL library on a real-world use case model for a truck dispatching problem. The results show the benefits of using RL in real-world problems to find better dispatching policies. Additionally, the visualization capabilities of AnyLogic enabled us to explain the RL agent’s reaction to changes in the system.

Keywords

Discrete Event Simulation, Agent-Based Modeling, Decision-Making, Dispatching, Reinforcement Learning

5.2 Introduction

Fast decision-making is an essential aspect of manufacturing systems in Industry4.0 era. The short time-to-market and the rise of customer-centric production impose high competitiveness on current markets. Industry4.0 concepts provide a paradigm shift towards faster adaptive manufacturing systems [88, 89].

Simulation tools and data-driven models are used in Industry4.0 to provide insights into systems’ behavior [88, 90]. These decision-support tools visualize the system components for better interoperability. Moreover, these tools recommend improvements and decisions based on predetermined performance indicators. DT concept emphasizes the role of simulation models, in order to build a virtual replica of the physical system [18].

DES is a dynamic and probabilistic simulation approach that simulates the system’s state over an interval of time [91]. DES is used in industry to model and analyze manufacturing systems. DES simulation provides decision-makers with important statistics regarding the system’s performance indicators, such as resources’ utilization and work-in-process products.

DES simulation tools are in continuous improvement to keep up with the industry’s needs [90].

With the migration to Industry4.0 technologies, DES simulation tools have been adding cloud-based features. Additionally, DES simulation tools are currently in the process of developing DT enabling technologies and APIs.

Operational Research (OR) methods have been used in combination with DES simulation tools to find the best system configuration [92]. Most simulation tools have one or more OR optimization approaches integrated and automated into them. This enables model designers to run the simulation model multiple times with different settings to search for the optimal configuration. However, a new optimal configuration needs to be recomputed whenever the system’s behavior changes [93].

In contrast to OR methods, RL learns a policy through interacting with the system [94]. An optimal or sub-optimal configuration is generated by applying that learned policy. RL is a subfield of ML that focuses on training the machine to take decisions based on the system’s state. RL is used for decision-making in multiple domains, especially in planning and robotics [94]. Our main goal is to utilize RL capabilities in current DES simulation tools.

Emerging RL in DES simulation tools will ease the usage of RL in the industrial domain. RL provides fast decisions and recommendations for decision-makers, which is a step towards applying Industry4.0 concepts. Moreover, RL does not need retraining with each change in the system’s behavior, as the RL policy is updated continuously when feedback is received from the system [93]. Finally, DES simulation tools provide visualization capabilities that can be used to illustrate the RL policy. This will help in explaining the RL actions to managers and decision-makers.

In this paper, we present our approach to embed RL agents into DES simulation. We chose AnyLogic as the DES simulation tool to use, because of its flexibility to add new libraries and codes to it. Moreover, AnyLogic enables the use of Python codes inside AnyLogic models. Python is a popular programming language that is used for ML applications. We developed a RL library for AnyLogic and applied it to a real-world use case of a truck dispatching problem.

The rest of the paper is organized as follows; A brief background is introduced in Sect. 5.3. Then our RL embedding approach is illustrated in Sect. 5.4. The truck-dispatching use case scenario is presented in Sect. 5.5. The use case results and the RL visualization are discussed in Sect. 5.6. Finally, the paper’s conclusion is presented in Sect. 5.7.

5.3 Background

In this section, we state the usage of DES simulation in decision support applications. Then we briefly introduce agent-based modeling and Reinforcement Learning. Finally, we give an overview of the popular approach to combine RL and DES simulation.

5.3.1 Decision Support using Simulation

DES has been used as a decision support tool in order to provide performance indicators about manufacturing systems. The resulting information is used to evaluate different possible scenarios of the system's operations [90]. It provides a virtual environment to evaluate the effects of changes in the system's parameters, for example, changing the number of operators in a production line. DES is also used in other applications to predict the system's stability over time and its future states.

5.3.2 Agent-based Modeling

Agent-based modeling is used for building DES simulation models. The main objective is to add behavior-based logic for different components of the model [95]. This enables the implementation of complex behaviors in a systematic approach. It also gives more controllability for the agent over the simulated system.

AnyLogic simulation provides a user-friendly interface to build agent-based models. The AnyLogic agent is implemented as an independent component with a flowchart-based behavior. In this work, we use the agent-based modeling approach to add RL-based intelligence to the AnyLogic agent's behavior.

5.3.3 Reinforcement Learning

RL is a subfield of ML where the machine learns to interact with its environment based on past experiences and learning [94]. The RL agent observes the environment's state s and takes an action a according to the learned policy π . That policy is updated by feedback to the RL agent called reward r . This reward indicates the goodness of the taken action given the environment's state. The reward is usually generated by formulating a mathematical function that combines one or more predetermined performance indicators.

RL needs to be trained for a long time in the targeted environment. This can be difficult for some applications, due to the limited resources and the expensive cost of performing experimental runs. Additionally, the quality of data collected from the environment affects the RL

agent’s ability to understand the impact of its actions. Therefore, simulated environments have been used to provide an unlimited number of replications for the RL agent with clean synthesized data. The trained RL agent’s policy is then tested and updated on the real environment [96–98].

5.3.4 Reinforcement Learning in Simulation

RL has been combined with simulation tools using different approaches in literature. Most of the research studies focus on the decision problem, RL algorithm used, and RL performance. Therefore, the interface between RL and the simulation tool is not taken into consideration in these studies. Usually, RL controls the simulation flow [93, 96–101]. In other studies, a whole simulation run is executed before updating the RL policy [102]. The system state and reward are passed to the RL through system calls [97, 98, 100], or shared Database [102].

Most of the studies use a RL script to pauses/runs the simulation model. The simulation runs till an action is needed. Then the simulation pauses and returns the control back to the RL script, in order to get the RL agent action. Finally, the RL script passes the action back to the simulation and resumes it. An API for the simulation tool is needed in order to control its flow from the RL script. Moreover, an additional code is needed to get the system’s state and reward value from the simulation model.

In the previous approach, the RL policy is updated at the end of a whole simulation run, called an epoch. This enables the RL agent to test its policy for a considerable amount of time. However, the training process will need hundreds of simulation runs in order to train the RL agent, which leads to high computational cost. Additionally, the simulation model must be developed using a finite horizon modeling approach, where the simulation must end at some point.

5.4 Embedding Reinforcement Learning

In our approach, we built the RL library for model designers. This requires the simulation tool to control the RL operations. Contrary to the popular approach, the simulation tool initiates the RL agent inside the simulation model. Additionally, the RL algorithm and its parameters can be selected by using hyperparameter optimization, since model designers might not be experts in the RL field. The RL library is composed of two main components; *RL Interface*, and *RL Agent*. These components are implemented as AnyLogic agents that can be used in any AnyLogic model. Figure 5.1 shows the interface between the AnyLogic model and the RL library components.

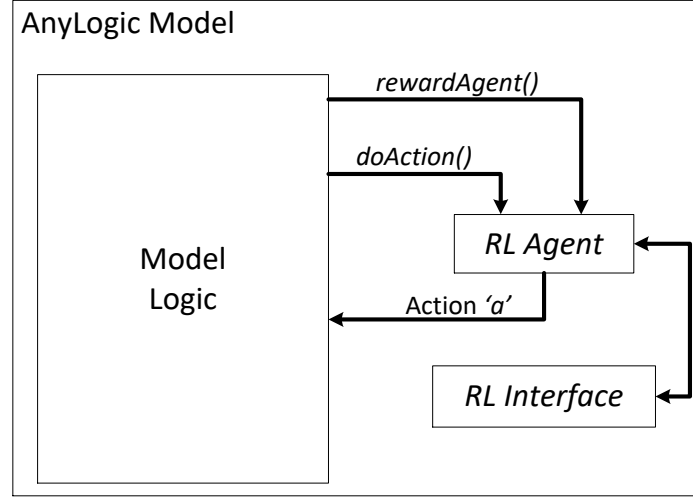


Figure 5.1 RL library components' interface with AnyLogic models

RL Interface is an AnyLogic agent that creates a RL agent. The model designer needs to set the number of observation variables and the number of actions in *RL Interface*, according to the problem definition. The RL algorithm and its parameters are defined by using AnyLogic parameters. Model designers have the ability to use the default values of *RL Interface*.

RL Agent is also an AnyLogic agent. *RL Agent* interacts with *RL Interface* to provide a systematic approach to define the system state, action, and reward. The model designer calls the function *doAction()* from *RL Agent* to get the RL agent's action, while calls the function *rewardAgent()* from *RL Agent* to reward the RL agent on its last action.

Since the system state and the reward function are problem dependent, the model designer needs to implement them according to the problem and system definitions. The model designer needs to implement two functions that are required by *RL Agent*. Firstly, *getObservations()* is an AnyLogic function that collects the current system state. Secondly, *rewardFunction()* is an AnyLogic function that computes the reward of the last action when called. A third function is also available *noRewardValue()*, which passes a default reward value in the case of temporal difference. The *noRewardValue()* function passes zero reward by default, but it can be modified by the model designer.

5.4.1 User Experience

As our main goal is to make the use of RL agents user-friendly for model designers, all the information needed by the RL library components are passed from inside the AnyLogic

model. This ensures that model designers do not need any additional tools to use RL agents in their models.

The RL algorithm is implemented in snippets using Python and AnyLogic's Pypeline library, inside the RL library components. This approach hides the RL code from model designers. Additionally, it does not require model designers to have prior knowledge of Python.

5.4.2 Hyperparameter Optimization

Hyperparameter Optimizer is an additional component of the RL library. *Hyperparameter Optimizer* is a Java class compatible with AnyLogic experiments. It enables model designers to search for the most optimum RL parameters to use in their model. Figure 5.2 shows the interface between the AnyLogic model and *Hyperparameter Optimizer* in an AnyLogic simulation.

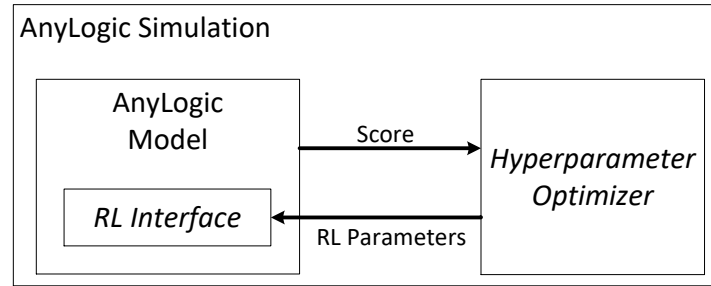


Figure 5.2 *Hyperparameter Optimizer* interface with an AnyLogic model

Hyperparameter Optimizer simulates the AnyLogic model multiple times with different RL parameters' configuration to find the optimum RL parameters values. In order to evaluate the RL parameters' performance, the model designer needs to compute a performance indicator and set it in an AnyLogic variable. That AnyLogic variable is then used to give a score for each RL parameters configuration.

Hyperparameter Optimizer uses Tree of Parzen Estimators (TPE) algorithm to search for the optimum parameter values. The TPE algorithm directs the optimizer to better explore the search space. Gaussian process (GP) based approach estimates the expected reward of the RL parameters. The exploration is done iteratively, where a sample of the search space is tested, then a new sample is selected using the updated GP model.

5.5 Truck Dispatching Use Case

We applied our approach to a truck dispatching problem. This dispatching problem is inspired by a real-world use case provided by SimWell Consulting & Technologies Inc. The goal is to train a RL agent to dispatch a set of trucks inside a simulated environment. The trucks transport payloads from one of many source points to one of many destination points. Then the truck goes back to one of the sources to transport a new payload. The environment has a set of 5 sources and 5 destinations. These points are located at different locations and connected with a set of roads. Figure 5.3 shows an overview map of the simulated environment sources and destinations.

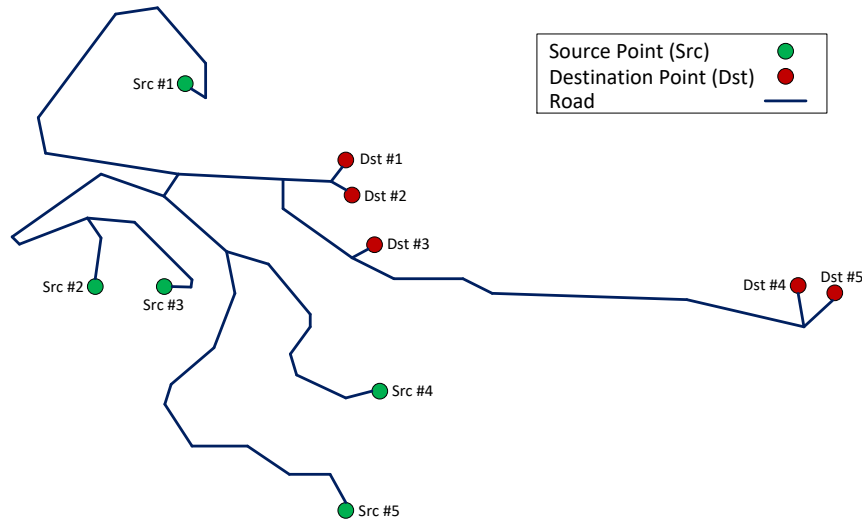


Figure 5.3 Truck transportation environment

5.5.1 System Description

Sources and Destinations take a random delay time to load and unload a truck. The loading and unloading time follows a triangular probability distribution. Trucks take more time at source points with an average loading time of 20 minutes, while unloading a truck payload takes an average of 10 minutes. Moreover, a truck takes transportation time to move from one point to another depending on the distance between the two points. If the source or the destination is currently busy serving a truck, all trucks that arrive wait in that point's queue until being served. Finally, each source and destination is taken out of service due to maintenance according to a random uptime and maintenance time.

The simulated environment has three congestion scenarios; not congested system with 15

trucks, near system capacity with 30 trucks, and over system capacity with 50 trucks. The three different scenarios were used to train the RL agent to evaluate the RL agent's behavior. Moreover, the simulated environment has two types of trucks, where each truck type has a different payload capacity. Trucks of type 1 have a maximum capacity of 50 tons, while trucks of type 2 have a maximum capacity of 35 tons. At the start of each simulation run, the truck type is assigned to each truck at random with a 50:50 probability. This will encourage the RL to learn different dispatching behavior according to the truck's type and the system congestion.

5.5.2 Business Objective

From a business point of view, the objective is to increase the productivity of the site by improving the truck dispatching policy. This objective is mapped mathematically into increasing the overall transported tons during a period of one month. The RL reward function is defined as the rate of tons transported per action, which is computed as the tons loaded/unloaded divided by the time duration to perform the dispatching action. The time duration includes the traveling time, waiting time, and the loading/unloading time.

5.5.3 Reference Heuristic Rule

In the real system, the shortest queue is applied as a heuristic rule to select the truck's next point. This rule is used to distribute the trucks over the waiting queues to decrease the truck waiting time. Additionally, the rule considers in-service sources and destinations only. This means that the truck will not be sent to an in-maintenance point, even if there are no trucks waiting at that point's queue.

5.6 Results and Discussion

The heuristic rule described in Sect. 5.5.3 is used as a reference to measure the performance improvement in terms of overall tons transported. The Table 5.1 shows the total transported materials in tons over a one-month period. The simulation results are generated for the three congestion scenarios presented in Sect. 5.5.

5.6.1 RL Results

In the 1st scenario, in which the system is not congested, the number of trucks waiting in a queue is not large. This results in lower waiting time, which increases the impact of traveling

Table 5.1 Heuristic Rule Results

Congestion Scenario	Overall Tons (x1000)
Scenario 1 (15 Trucks)	332
Scenario 2 (30 Trucks)	505
Scenario 3 (50 Trucks)	533

distance on the total time taken by a truck, from a source to a destination and back again. The RL agent learns to reduce the traveling time overhead by not selecting the furthest source/destination points (Src5, Dst4, and Dst5), as shown in Figure 5.4. The RL agent also prefers some sources/destinations over others. The RL agent preferred trucks of type 1 over type 2 when dispatching trucks to the nearest destinations. By analyzing the trucks waiting in a queue at different timestamps, trucks of the same type tend to cluster together. This means that most of the trucks waiting at a source/destination at a certain timestamp will belong to the same truck type. The RL ability to consider the traveling time and the truck type effect increased the overall transported tons by 8.7% compared to the heuristic rule.

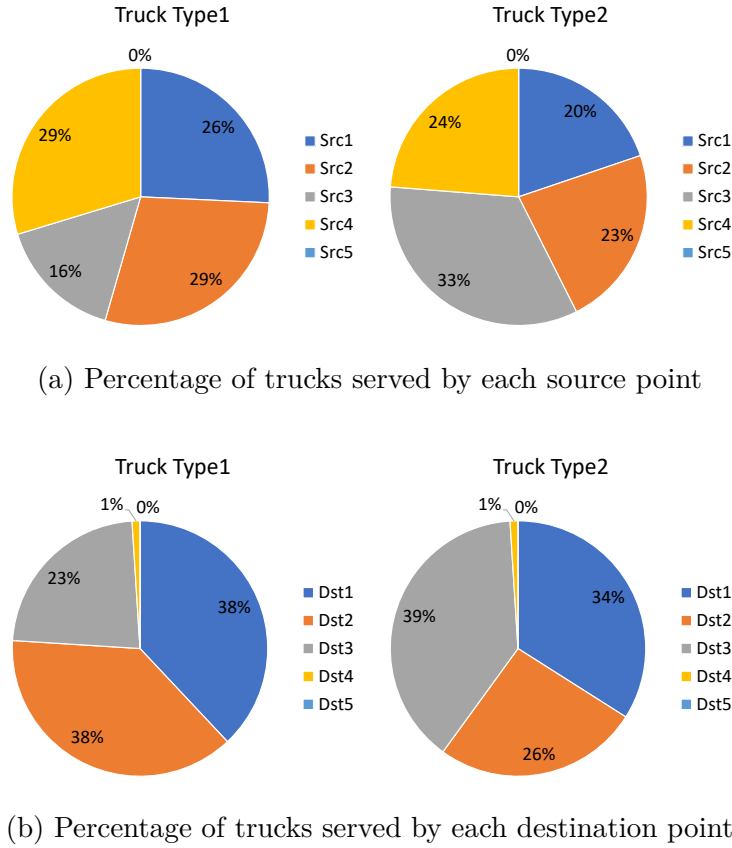


Figure 5.4 Percentage of total trucks dispatched in the 1st congestion scenario

In the 2nd scenario, in which the system is near its capacity, the average waiting time of a truck would exceed the traveling time overhead. This led the RL agent to learn to use all the sources, and use the furthest destinations at congestion times, as shown in Figure 5.5. By analyzing the trucks waiting in a queue at different timestamps, the RL agent dispatched the trucks according to their type as in the 1st scenario. The learned policy increased the overall transported tons by 4.1% compared to the heuristic rule.

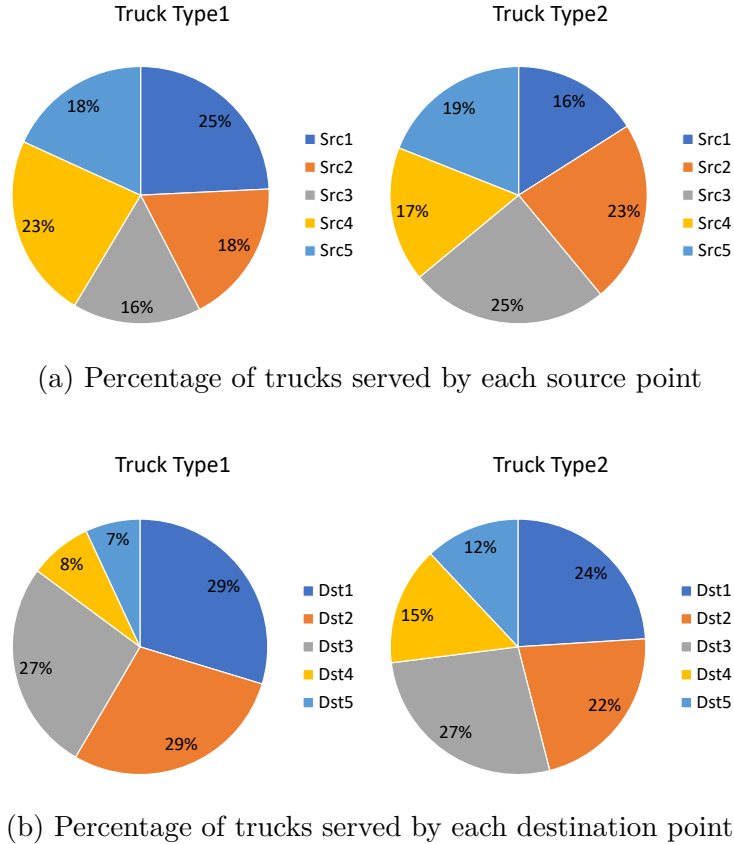
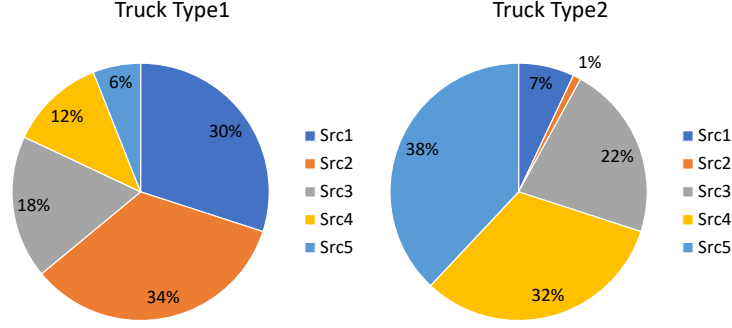
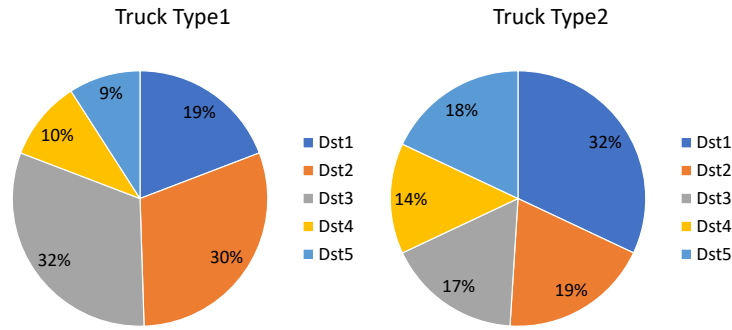


Figure 5.5 Percentage of total trucks dispatched in the 2nd congestion scenario

In the 3rd scenario, in which the system is over its capacity, all sources and destinations are used as shown in Figure 5.6. However, we can see that the RL agent learned to dispatch trucks to specific sources and destinations based on the truck's type. The RL agent preferred trucks of type 1 over type 2 when dispatching trucks to the nearest sources and destinations. This policy still increased the overall transported tons by 2.7%, despite the long waiting times.



(a) Percentage of trucks served by each source point



(b) Percentage of trucks served by each destination point

Figure 5.6 Percentage of total trucks dispatched in the 3rd congestion scenario

5.6.2 Benefits of Embedding RL in AnyLogic

Visualizing the system helps in explaining the learned RL policies and justifying its results. The visualization capabilities in AnyLogic enabled us to analyze the RL agent's behavior over time. It also allowed us to study the system's dynamics and the RL agent's reaction to such events as the sudden failure of a source or a destination.

Training the RL agent on the three different scenarios was done on the same simulation model. Changing the system behavior in AnyLogic models was done by simply changing an AnyLogic parameter. This allows the model designer to investigate different scenarios and train new RL agents without any additional effort.

Finally, the RL algorithm and its parameters were not mentioned in the results as it is irrelevant to both the model designer and business stakeholders. The hyperparameter optimizer described in Sect. 5.4.2 enables non-experts to use RL without any prior knowledge. The RL results were generated in a few hours using a single commodity computer.

5.7 Conclusion

RL is an important decision-support tool in the Industry4.0 era. The current improvements in DES simulation tools, such as AnyLogic, enabled the embedding of intelligent agents into DES simulations. In this paper, we took advantage of these improvements and programming flexibility to develop a RL library. The RL library provides a user-friendly interface to use RL agents in simulation models. We applied our RL library on a truck dispatching use-case modeled using AnyLogic. The RL agent learned to adjust its policy according to the congestion in the system and truck type. Visualization features in AnyLogic enabled us to better understand the learned RL behavior.

**CHAPTER 6 ARTICLE 3: A DIGITAL SHADOW FRAMEWORK USING
DISTRIBUTED SYSTEM CONCEPTS**

Ayman AboElHassan, Soumaya Yacout

Published in:

Journal of Intelligent Manufacturing

Date of publication: 27-Sep 2022

6.1 Abstract

Digital Twin (DT) is a research topic that gained momentum in the Industry 4.0 era. The goal of DT is to create a virtual real-time intelligent system that is a typical twin of the physical system. DT provides analysis, prognosis, planning, and rapid response when needed. Digital Shadow (DS) is an artifact concept of DT that provides a real-time replica of the physical system. Information in a DS is passed in one direction only, from the physical system to the virtual one. While in DT, the information goes in both directions. The definition and roles of DS and DT are overlapping. However, DS can be defined as the main component of a DT system. In this paper, DS roles are specified. Based on these roles, A DS framework architecture is proposed, and the communication system between its components. The proposed framework design is built using distributed system concepts such as event-driven architecture, microservices, and containerization. These concepts are well-defined and utilized in the software engineering domain. The originality of the proposed framework is the definition of a systematic approach for designing and integrating Digital Models (DMs) from different vendors and domains. An experiment is designed to prove the framework's ability to shadow a physical system in real-time. Multiple DMs are implemented and deployed on the proposed DS framework. These DMs are used to shadow a natural gas compressor system. Experimental results prove the practicality of our proposed DS framework to operate in real-time.

Keywords

Industry 4.0, Digital Twin, Digital Shadow, Distributed Systems, Modeling, Simulation

6.2 Introduction

DT concept has become a focal point since the announcement of Industry 4.0. DT was first defined by NASA researchers as "*integrated multi-physics multi-scale probabilistic simulation that mirrors the physical system*" [17]. DT definition had broadened to become a real-time digital replica of a component, a product, or a production system [14,15]. The digital replica provides statistical insights, future predictions, and decision-support capabilities. An exponential number of research articles are being published annually on the topic of DT and its applications [48]. However, DT definition and related terms lack conceptual standardization and consensus [47,58].

DS is a by-product concept of DT [29]. DS is defined as a digital replica that shadows

the physical system [14, 75]. The definition of DT and DS remains intertwined and their roles are overlapping. Role-based separation between DS and DT is proposed in a previous publication [103]. DS is defined as a "*digital replica of the physical system that matches its physical counterpart's state in real-time*". While DT is defined as a "*real-time decision-support sandbox*". DS is a core component of DT, which provides a real-time representation of the physical system to DT. DT uses DS digital replica to improve the physical system's performance in real-time.

6.2.1 Motivation

Multiple DT frameworks have been proposed in the literature; however, these frameworks lack generality and/or practicality. Creating a realistic general-purpose DT system is challenging. General purpose DT utilizes different modeling approaches to describe a system. These models are integrated into the DT framework despite their different interfaces and time flow. For example, collaborative robotic arm DT integrates a continuous-time kinematics model and an event-driven collision avoidance model. General purpose DTs are not limited to specific modeling tools or vendors, they are able to share information between different models from different vendors. Additionally, any model should be integrable into the general purpose DT. The separation between DT applications and simulation models eases the generalization of the DT framework [25, 71]. DS provides a virtual space for models to simulate the physical system. Then DT systematically pulls information needed for decision-support applications [103].

Most DT frameworks designs are described on system-level without implementation details as in [46, 70, 71, 104]. Such articles focus on DT concepts and data flow between the framework components. DT components are defined based on functional capabilities and integration into the industrial automation pyramid.

Other DT frameworks are tightly-coupled to the applied case study and the utilized modeling tools as in [25, 62, 63]. This approach generates a realistic DT framework that can be implemented and tested on real-world problems. However, focusing on a specific problem reduces DT generality and ability to expand into other domains. DT framework implementation requires modifications in order to include additional modeling tools.

Many published studies focus on the applications of real-time modeling. These studies either connect the model directly to the physical system through a direct communication interface as in [63–68]; or depend on previously developed DT frameworks as their enabling technology, as in [105–108]. Other studies do not use simulation models in their DT application [48, 49]. In that sense, some of the systems described as DT systems in literature are in reality DS

systems or real-time monitoring systems.

6.2.2 Contribution

In this paper, a general-purpose DS framework is proposed using distributed system concepts. DS framework provides a real-time digital replica of the real world. Gateways acquire raw data from the physical system in real-time. The raw data is then published to a Pub/Sub communication channel, where it becomes available for any model in the framework. Microservice architecture is used to design DS components as services. Multiple DMs run simultaneously and exchange information over the Pub/Sub communication channel.

In comparison with previous studies, the proposed DS framework is designed based on distributed system concepts. The originality of the proposed framework design is in the modularization of DS models and information sharing between different components. The framework is based on a microservices architecture to adapt different simulation and modeling tools from different vendors and domains. DS model interface is defined using JSON files generated from ontology-based knowledge graphs. This enables automated deployment of DS models on the proposed framework.

The originality and contribution of this paper are summarized as the key elements of the proposed DS framework architecture;

1. A systematic approach for DM interface
2. Modularization of DM and information sharing between DMs
3. Encapsulating DM in Docker containers
4. Use of Kafka for high-throughput low-latency communication
5. Use of Kubernetes for automated deployment of DS components

6.2.3 Organization

The rest of the paper is organized as follows; Literature review of DS and DT concepts is presented in Sect. 6.3. Then a brief background of distributed systems and their use in DT is presented in Sect. 6.4. The proposed DS framework is described in Sect. 6.5. The framework is applied in an industrial case study presented in Sect. 6.6. Multiple aspects of the proposed framework are discussed in Sect. 6.7. Finally, conclusion and future work are presented in Sect. 6.8 and Sect. 6.9.

6.3 Literature Review

In this section, Different definitions of DS in literature are discussed in relation to DT. Then, a brief review is presented of some DT architectures in literature. This is due to the high publication rate in the topic of DT compared to DS.

6.3.1 Digital Shadow

WGP was the first to define Digital shadow (DS) in 2016 [30,31]. In that definition, DS integrates real-time information collected from multiple sources into a general image of the physical system. This image is used by decision-support tools to provide real-time recommendations for managers. WGP separates between DT and DS in terms of model fidelity [8,30,31]. DT requires high-fidelity models that describe the system in fine detail for decision-making processes. While DS uses low-fidelity models to describe the system using an acceptable abstraction level.

Ladj et al. [73] adopted a similar definition of DS. They defined DS as a core component of DT. DS consists of ontology-based data models that capture the physical system state. DS uses data-processing algorithms to aggregate data from different sources and formats to provide a dynamic database for DT applications. Brecher et al. [74] proposed the use of a DS Caster that processes and annotates raw data. DS Caster then updates the ontology-based data model. That definition limits the DS role to data collection and processing, where engineering knowledge is added as predefined relational rules only. In our proposed definition, DS provides more depth into the system behavior through a wider range of modeling techniques and data analytics. Additionally, defining DS as multiple models sharing information in-between enables the development of a holistic view of the system.

Brauner et al. [51] also adapted a DS definition similar to WGP. They consider true DT as infeasible since it is hard to fully represent a physical system in a digital form. Additionally, DT requires a massive amount of data and computations to build a virtual replica, which results in high-latency operations. They define DS as a task-driven domain-specific simulation model that describes part of the physical system in real-time. DS aggregates raw data in a model, that is processed to extract and share knowledge. A system could have multiple DSs, each describing a part of that system. That definition of DS is similar to our proposed DM. Our proposed DS provides an aggregate view of the physical system using multiple DMs. Additionally, our proposed DS definition focuses on imitating system behavior and embedding prior knowledge.

Another definition, that is popular in literature, categorizes DT and DS according to data

flow direction between virtual and physical worlds [8, 14, 29, 32, 33]. DS receives information automatically from the physical system, but DS can not send feedback to the real world. While DT sends and receives information automatically in a bidirectional way. DT receives the latest readings from the physical system, while it sends control commands back to the physical system controller.

The lack of separation between the two concepts, DS and DT, makes their roles mixed up in literature. Stark and Damerau [15] proposed an intersected functionality between the two terms. DS is defined as the collection of system data, while DT is defined as the digital model and its DS. A DS's role in that definition is limited to real-time data acquisition similar to WGP and Ladj et al. [73]. Schuh et al. [75] proposed the use of DS for the digitalization of the manufacturing process. Simulation models are used to create a digital copy of the production line. However, they did not discuss the difference between DS and DT. Becker et al. [72] used model fidelity to differentiation between DT and DS. DT is defined as a high-fidelity replica of the physical system. While DS is defined as a compact purpose-driven model representing part of the physical system. Konstantinov et al. [76] followed the data-flow direction differentiation between DT and DS. Their WDT is used as a DS that keeps track of the physical system and is accessed through a web browser by technicians. Kassen et al. [77] also followed the data-flow direction to define DT and DS. Their data-driven DS simulation model is designed to accommodate different types of production systems.

6.3.2 Digital Twin

As DT publications are growing exponentially, the number of review studies is growing too. This helps researchers keep track of the various domains where DT is applied and the technologies used in DT. Uhlenkamp et al. [14] made an extensive study on the aspects of a DT framework. While Cimin et al. [8] made an extensive study on published papers focusing on DT applications. Qi et al. [27] defined a 5 dimensions characterization of DTs and discussed the DT's enabling technological tools. Lu et al. [23] made a literature review on the technologies applied in the DT domain. Finally, Tao et al. [69] made a comparative study about the relationship between DT and CPS.

DT have been applied for various applications in the industrial domain. Table 6.1 presents an overview of DTs applied in industrial case studies, their scope, and applications. Several DT architectures have been proposed in the literature. The common aspect between these architectures is the separation between the DT framework and the models deployed on it. Additionally, the use of communication channels to connect different components of the system together. In these architectures, data is collected and preprocessed on the factory

level and then sent to DT. The data is then stored in a shared database and/or MQ in order to be shared within the framework. When the operator requires a service, the corresponding models are initiated. These models process the collected data and send the analyzed results back to the operator.

Tao et al. [70] proposed a DT architecture under *Made in China 2025* initiative. They proposed the use of models as services that can be requested from a shop-floor service system. They later expanded their DT architecture into 5 dimensions to incorporate data collected in DT and communication between the different components [27]. Another publication discussed the use of multi-layer models to address the hierarchical relationship between objects in the shop-floor [109]. Zhuang et al. [62] also proposed a DT architecture similar to Tao et al. [70].

Park et al. [25] proposed a three-level DT architecture. The first level includes the physical system controllers and sensors. The second level contains DT models and inner services. Finally, the third level represents DT applications. In their architecture, they separate between DT basic services performed internally as simulation and monitoring; and DT applications executed upon user request such as scheduling maintenance and optimizing production plan.

Qamsane et al. [71] proposed a DT architecture in order to provide a more generalized DT framework. They argue that simulation-only and model-only approaches provide analytical capabilities; however, these approaches do not provide a fully-functional DT system. Their architecture divided the system into 3 layers, with DT as a part of the middle layer called *software-defined controller*.

Jiang et al. [50] proposed a service oriented DT framework built on the Data, model, and service (DMS). Their DMS framework defines each component of DT framework as a DMS module, where data is processed by the model and provided upon request as a service. The service part of the DMS module also provides communication capabilities to other DMS modules in the framework.

Redeker et al. [56] proposed a DT framework based on RAMI 4.0 where AAS is the core component. The framework is designed to collect the data through AAS-servers in the physical system using OPC-UA. The information is shared to business-level AAS over the cloud using MQTT and RESTful services. Similarly, Park et al. [57] used AAS as a base component in their autonomous DT. AAS asset description is used in the self-configuration of DT without human intervention.

Table 6.1 Overview of DT Applications in Industrial Domain

Publication	Scope	Tools	Application	Domain
[46]	System	N/A	Driving assistance configuration	Automotive
[25]	Equipment	Modelica, Perti-Net	Root cause analysis	Robotics
[71]	Equipment	Mathematical Model	Machine health shop-floor reconfiguration	Manufacturing
[66]	System	3D CAD, PLCSIM	Warehouse reconfiguration	Manufacturing
[106]	Equipment	3D CAD, Tecnomatix PS	Fault detection	Robotics
[107]	Equipment	3D CAD, CNN	Product identification	Robotics
[48]	Equipment, System	Dashboards	Real-time monitoring	Manufacturing
[49]	Equipment	3D CAD, Mathematical Model	Mobile robot calibration	Robotics
[79]	Equipment	N/A	Real-time monitoring	Manufacturing
[80]	Equipment	3D CAD, Mathematical Model	Tool path analysis	Manufacturing
[56]	Equipment	N/A	Condition monitoring	Manufacturing
[33]	Equipment	Mathematical Model	AGV tracking	Robotics
[64]	Equipment	URSim, Experior	Trajectory optimization	Robotics
[65]	Equipment	3D CAD	Real-time monitoring	Robotics
[105]	System	3D CAD	Production line configuration	Manufacturing
[50]	Equipment	Statistical Model	Predictive maintenance	Manufacturing
[81]	System	Artificial Neural Network	Asset health monitoring	Manufacturing
[77]	System	AnyLogic	Optimization production line	Manufacturing
[63]	Product, Equipment	CAM Simulation, ANSYS	Machining process optimizing	Manufacturing
[73]	Equipment	Web-Ontology Knowledge Model	Fault detection	Manufacturing
[68]	System	AnyLogic	Virtual commissioning Resources management	Manufacturing
[110]	System	N/A	shop-floor scheduling process optimization	Manufacturing
[111]	System	LabVIEW, ANN	Anomaly detection	Food production
[112]	Equipment	3D CAD	Human-robot collaboration	Robotics
[113]	Equipment	3D CAD	Deburring operation	Robotics
[108]	Equipment	3D CAD, Simulink	Process planning of raise boring	Mining
[67]	System	AnyLogic	Production scheduling	Food production

DT concept lacks standardization [47, 58]. This is partially due to the general definition of DT. Different systems and domains require different modeling approaches; however, generic models for multiple domains do not exist. For example, modeling an electric grid differs from modeling an assembly line. Some publications focused on the process of designing a DT to formalize a workflow for creating and deploying DTs. Ontology and knowledge graphs provide a semantic description for automating DT deployment [58]. Ontology provides meta-model information used for defining models interface and shared information. Moreover, ontology enables system interoperability that facilitates human involvement and supervision.

6.4 Distributed Systems

Distributed systems concepts are used in designing the proposed DS framework. Distributed systems domain has been studied in the software engineering discipline. Cloud computing is a sub-field of distributed systems, which became one of the main pillars of Industry 4.0. Cloud computing provides the required computational and storage power for Industry 4.0 systems. Network performance of cloud computing system affects DT system reliability [114]. However, large cloud providers, such as Microsoft Azure, are addressing the network quality needed by DT applications in their IoT solutions [115].

In this section, a background discussion is provided about distributed system concepts: event-driven architecture, service-oriented architecture, and containerization. Previous studies are highlighted for discussing the benefits and potential of applying these concepts.

6.4.1 Event-Driven Architecture

As discussed in Sect. 6.3, DT frameworks utilize communication systems to connect different DT components. Message Queue (MQ) such as MQTT [25, 56, 79], MTConnect [80], AMQP [79], and RabbitMQ [50] are used in DT frameworks. MQ is an asynchronous message channel that transfers information from one component to another. MQ provides low-latency communication for one-to-one information sharing. However, MQ does not support multiple subscribers for the same message. Some MQ systems, such as Google Pub/Sub, duplicate the message as a workaround to send the same message to multiple subscribers.

Pub/Sub communication systems are used in some DT frameworks as [70, 71]. Pub/Sub communication provides a global many-to-many information sharing platform. Publishers and subscribers do not interact directly. Publishers send tag-named messages to the Pub/Sub communication channel. Subscribers receive messages from the Pub/Sub communication channel based on the subscribed tag names. Multiple Pub/Sub communication systems

Apache Kafka® is an open-source Pub/Sub communication system used in data-stream services. Amazon Kinesis is a managed version of Kafka, which is managed and deployed on the Amazon AWS cloud system.

Apache Kafka® has been used in DT applications in the literature. López [81] discussed the potential benefits of event-driven architectures and Pub/Sub communication, where they used Kafka in their case study. They discussed its importance for system flexibility and scalability. Radchenko et al. [83] used Kafka to pass information to Kepler workflows that model the system as a structured process-flow. Kafka was used as a middleware layer for collecting information from IoT sensors and then passing it to Kepler workflows. They also discussed the potential benefits of using Kafka in [82]. Kamath et al. [79] used Kafka with InfluxDB and Grafana to provide a time-series analysis of collected data from the real world. Kafka provided a real-time streaming pipeline to pass information from DT to analytical tools. While message queues such as MQTT and AMQP are used to collect information from the physical system. Steindl and Kastner [84] used Kafka as a backbone communication channel to connect different components of the system. They also discussed the importance of Kafka in their design.

The previous studies do not fully utilize Apache Kafka® capabilities. The proposed DS framework uses Kafka as a backbone communication channel to connect components of DS framework. Kafka ensures a high-throughput low-latency communication with fault-tolerance [116,117]. The distributed nature of Kafka provides high availability as data is replicated on multiple Kafka brokers. Kafka also enables the scalability of the DS framework, as new Kafka brokers can be added to the system without service disruptions.

6.4.2 Service Oriented Architecture

Almost all Industry 4.0 architectures are built on the use of SOA and microservices [53]. IIRA uses SOA concepts for the dynamic exchange of information. While RAMI 4.0 uses SOA encapsulation concept to provide a common interface between its modules. Finally, SITAM is built on SOA architecture that defines the system as a hierarchical structure of services. Some smaller initiatives and SOA approaches, such as FIWARE and ArrowHead frameworks, were also studied for providing data-centric aspects in the industrial domain.

Catarci et al. [55] proposed the design of DT models using SOA approach. DT collects information from the system in the form of vendor-based semantic models. The collected data are integrated and translated to a common representation by a *mediator* module. The translated information is then shared with CPS as a web-based service. DT sends data upon query request from synchronized systems or using a Pub/Sub message system.

Dobaj et al. [78] proposed the development of DT as a SOA middleware component in a CPS architecture. CPS applications request information from DT level, without the need to know the physical hardware and network on the shop floor. While DT collects the shop-floor information into a global holistic view. DT in that study provides data aggregation, service orchestration, and communication abstraction.

The previous studies show the importance of adapting the SOA concept in designing a framework. SOA and microservices concepts are applied to build the proposed DS framework in a loosely-coupled architecture. This approach eases the scalability of the system as new DS modules are added separately from other modules. A module can be modified without affecting the system as the service interface hides the module's details from the rest of the framework. Alongside the use of Kafka, microservices ensure framework resilience to network faults and enable the scalability of services.

6.4.3 Containerization

Containerization is still not a widely used concept in the industrial domain despite its importance in cloud computing. Radchenko et al. [83] used Dockers to automate the deployment of their DT system. Docker images contained the deployment instructions for Kafka and Kepler workflows. Redeker et al. [56] proposed the use of CD tools over Kubernetes (K8s) to enable DT system updatability. They used CD workflows to continuously test and deploy new AAS modules when modified by designers. While K8s update existing deployments without disrupting the service. Steindl and Kastner [84] discussed the importance of containerization and microservices in designing DT components. This approach eases the addition of new services due to the loosely-coupled interaction within the system.

The rising trend of containerization in the computer science domain highlights the importance of applying it in industrial applications. The proposed framework uses containerization for DM modularization and automation of the framework deployment. Dockers and K8s are used for deploying the DS framework and modules. Dockers provide virtualization and isolation for DS modules. This ease the deployment of DS on any hardware infrastructure. The combination of module isolation with microservices prevents runtime faults to propagate to the rest of the framework. K8s is used to deploy DS components and containerized DMs on cloud. K8s provides orchestration for the deployed Docker containers. This enables scalability and service availability during failures.

6.5 Digital Shadow Framework

As previously stated in Sect. 6.2, DS is defined as a digital replica of the physical system. DS mimics the system behavior using real-time data, prior knowledge, and simulation models. DS roles are defined as follows:

- Represent the physical system as an abstract model.
- Track the physical system state in real-time.
- Provide a behavioral explanation of the physical system state.
- Refine our knowledge of the system by comparing the abstract model to the physical system's behavior.
- Visualize the state of the physical system.

DS framework is designed based on event-driven architecture. The framework components communicate through a shared Pub/Sub communication channel. Microservices interface enables DS components to exchange information using a loosely-coupled approach. Figure 6.1 shows the DS framework architecture design. The DS framework is composed of: A) DS coordinator, B) Pub/Sub communication channel, C) Source gateway, D) Digital model manager (DMM), and E) Digital model (DM). The details of each DS component are presented in the following subsections.

6.5.1 Digital Shadow Coordinator

The DS framework is designed for deployment on distributed systems such as cloud platforms. DS coordinator provides an administrator interface to control the DS framework components. Additionally, it deploys and manages other DS components and their configurations on a K8s cluster.

Kubernetes (K8s) is an open-source orchestration system used in automating container deployment, scaling, and managing workload operations [118]. It was designed by Google engineers and released in 2015. K8s software is installed on each computer in the distributed systems, which are linked through K8s control plane. DS coordinator sends deployment commands and configurations to K8s control plane. K8s control plane is composed of multiple components such as scheduler and controller manager. K8s scheduler selects K8s node to deploy a container according to computational resources availability. While K8s controller manager keeps the K8s cluster at the configured state by DS coordinator [119]. This means

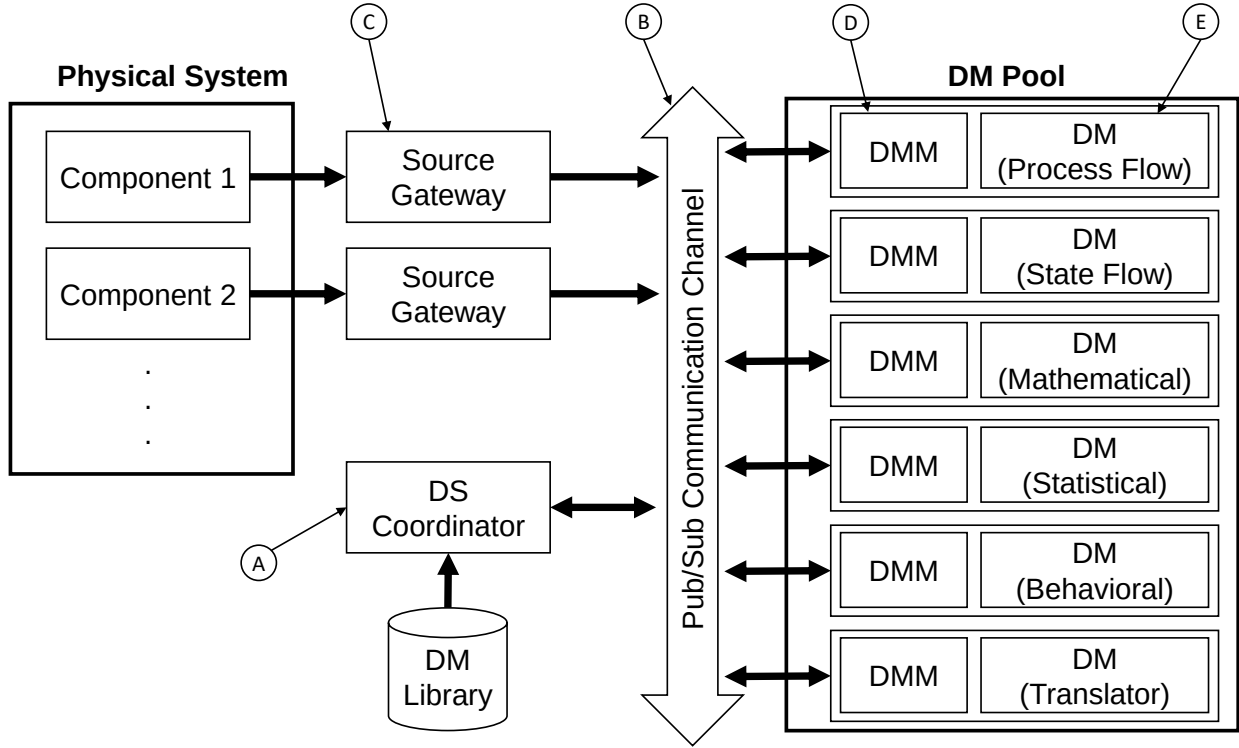


Figure 6.1 Digital Shadow System Architecture

if a DM container stopped due to runtime error, K8s controller manager will notify the scheduler to restart that DM container.

DS coordinator setups the K8s cluster that hosts the DS framework. K8s cluster is formed by a set of computational and storage resources, named K8s nodes. DS coordinator deploys DS components starting by the Pub/Sub communication channel. A shared storage library contains the K8s deployment scripts, which are used by DS coordinator. These scripts define Docker images of each DS component, services provided, and configuration. Figure 6.2 shows a sequence diagram for DM creation and deployment by DS coordinator.

After DS components are deployed on the K8s cluster, DS coordinator provides access to system administrators in order to manage DMs in the DM Pool. DS coordinator starts and stops the simulation of a DM and retrieves DM status. It also enables the deployment of new DMs on an existing DS framework or updating a DM with a newer version.

6.5.2 Pub/Sub Communication channel

The Pub/Sub communication channel is considered the backbone of event-based architectures. Apache Kafka® is used as the Pub/Sub communication channel. Kafka was developed

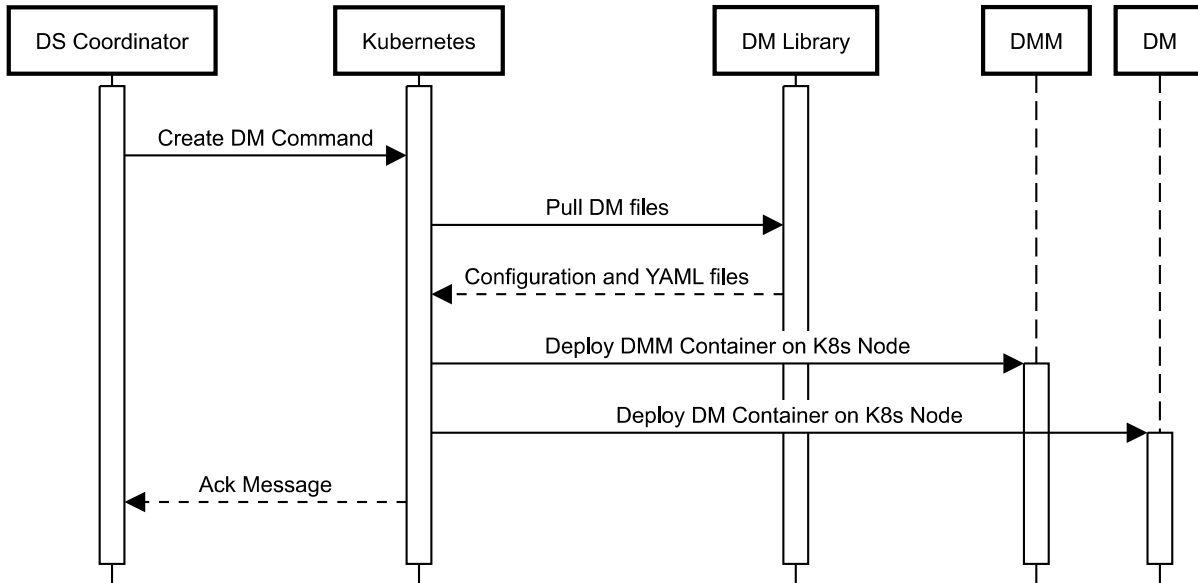


Figure 6.2 UML Sequence Diagram of DM Creation

by a group of researchers at LinkedIn as an open-source project in 2011. Kafka then became managed by Apache® Software Foundation. Apache Kafka® is used by a large number of companies in multiple domains.

Kafka provides a high-throughput low-latency messaging system for stream-processing platforms [116]. Kafka stores messages received from publishers in a key-value message format. A message's key is called a topic, which groups messages with the same context. Subscribers receive Kafka messages according to the subscribed topic names. Kafka runs on a cluster of multiple servers called Kafka brokers. Messages are distributed and replicated on multiple Kafka brokers according to the topic value. Distributing replicated messages on multiple Kafka brokers increases the availability of Kafka and its tolerance to system faults.

Kafka cluster is deployed on K8s using a dockerized version of Kafka v2.7.0. Each Kafka broker is deployed on a separate K8s node, where each node has a set of K8s headless services to communicate with the rest of the Kafka brokers. This approach allows scaling the Kafka cluster depending on the message traffic. Kafka cluster is scaled up by increasing the number of K8s pod replicas. A shared volume is used to write the Kafka message log. The shared volume allows any Kafka broker to restart from its current state if failure happens. This ensures the fault tolerance of the Kafka message system. Figure 6.3 shows an example of Kafka brokers and their communication with the DS framework components. In this example, 4 DS components communicate through 3 Kafka brokers. DS components send and receive

messages over 5 Kafka topics.

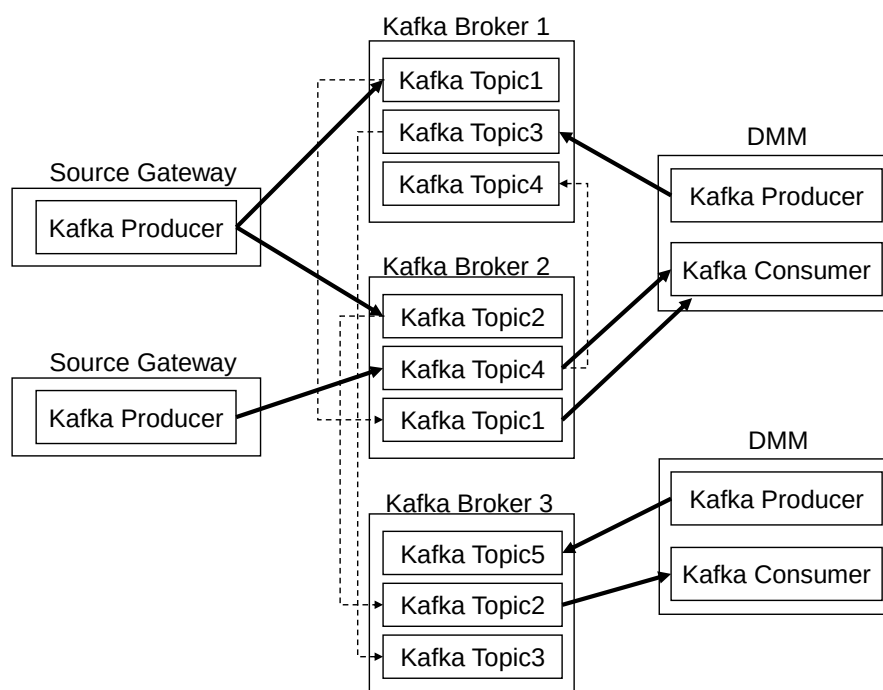


Figure 6.3 An example of Pub/Sub Communication Channel using Apache Kafka®

Kafka was originally developed to distribute data processing for fast stream processing. However, Kafka allows the same message to be sent to multiple subscribers by assigning each subscriber to a different consumer group. Kafka ensures that each message is read by only one subscriber from each consumer group. Therefore, a unique consumer group id is created for each DM.

Kafka messages are labeled by topic names. A publisher sends to Kafka brokers all topic names that will be published. While each subscriber subscribes to the needed topics. DM messages are designed to hold one physical attribute only. Kafka topic names take the form of *%PublisherName%_%CreationTime%_%AttributeName%*, where *%PublisherName%* is the name of the publisher, *%CreationTime%* is the creation time of the publisher K8s pod, and *%AttributeName%* is the physical attribute being published onto the Pub/Sub channel. For example, if a publisher sends temperature and vibration measurements of a machine, the publisher will send a message for temperature and another message for vibration. This message per attribute design enables subscribers to subscribe only to the needed information, which will reduce the message payload ratio, but also reduce the number of subscribers per topic.

A separate Kafka admin module is deployed as a K8s pod to provide service discovery for

other components in DS framework. The Kafka admin module manages the Kafka cluster and its properties. This module keeps track of Kafka brokers running on the K8s cluster and their IP addresses. When a component needs to connect to the Pub/Sub communication channel for the first time, it requests the IP addresses of the available Kafka brokers from the Kafka admin.

6.5.3 Source Gateway

Data acquisition from the physical system is done by the source gateway service. Source gateway type depends on access authorization and its proximity to the physical system. Source gateway can be either tightly-coupled or loosely-coupled.

A tightly-coupled source gateway is an edge-located service. The physical system information is collected and aggregated on edge node servers. Source gateway publishes the aggregated data directly to the Pub/Sub communication channel. The tightly-coupled approach reduces communication overhead as physical data is published once it is collected. However, the source gateway needs to be deployed at the factory side, which requires access authorization.

A loosely-coupled source gateway is a cloud-located service. Source gateway pulls the physical system data according to a user-defined rate. The physical data needs to be accessible through a secure network connection. The loosely-coupled approach requires lower access authorization, as data can be pulled from a shared database or a web server. However, higher communication overhead will occur due to the high data pulling rate needed to capture real-time changes in the physical system.

In both approaches, the DS framework does not terminate due to hardware faults, data acquisition faults, or network faults. When a hardware fault occurs in the physical system, the source gateway will not receive any new information to publish. While data acquisition faults are translated into missing data. Finally, the Pub/Sub communication channel depends on acknowledgment messages to confirm message reception. Therefore, lost messages will be resubmitted for publishing after network faults were resolved.

System designers define a state map translation for each source gateway as a JSON file. The use of JSON files as component definition enables the automation of system design using ontology-based knowledge graphs. The state map JSON file is then added to the K8s configuration script of the source gateway. This state map defines Kafka topics that are used to partition and publish the physical data, as described before in Sect. 6.5.2. Each physical information is labeled by combining the following names separated by underscore symbols:

- *%PublisherName%* is the source gateway name

- *%CreationTime%* is the creation time of the source gateway instance
- *%AttributeName%* is the physical attribute name

6.5.4 Digital Model Manager

Digital model manager (DMM) controls and integrates DM into DS framework. Each DM is connected to a single DMM and vice versa. DMM applies control commands received from DS coordinator. These control commands are used to start, stop, or request information from DM.

DMM connects DM models to the Pub/Sub communication channel. DMM contains a Kafka client interface that publishes and subscribes information to the Kafka cluster. DMM subscribes to all needed Kafka topics at startup. When the start signal is received from DS coordinator, DMM pulls the message already published at the subscribed topic. Then DMM listens to any new messages on the Pub/Sub communication channel. DMM also publishes DM information when ready. Moreover, DMM keeps the latest model status locally stored if requested by a user. Figure 6.4 shows an UML sequence diagram for DMM initialization operation.

As stated in Sect. 6.5.2 and 6.5.3, information is published to the Pub/Sub communication channel using Kafka topics. These topics are defined by system designers as a state map that translates the attribute name to its corresponding Kafka topic name. DMM contains an inverse mapping for the subscribed Kafka topics. Inverse map defines which Kafka topics need to be subscribed by DMM. The inverse map also defines variable names used by DM to represent these Kafka topics. DMM translates variable names in any message received from the Pub/Sub communication channel. Then DMM passes the translated messages to DM. Similar to the source gateway, DMM inverse mapping is defined as a JSON file in the K8s configuration script.

An additional state map is defined for DMM for publishing DM information. This state map translates DM state into Kafka topics. DM information is then published by DMM to other DMs through the Pub/Sub communication channel. DMM uses the same Kafka topic naming convention; however, *%PublisherName%* contains DM model name. DMM publishing state map is defined in the same JSON file inside K8s configuration script.

6.5.5 Digital Model

Digital models (DMs) are core computational components of DS framework, which simulate the physical system. DM is a Docker container that executes a single simulation model. Dif-

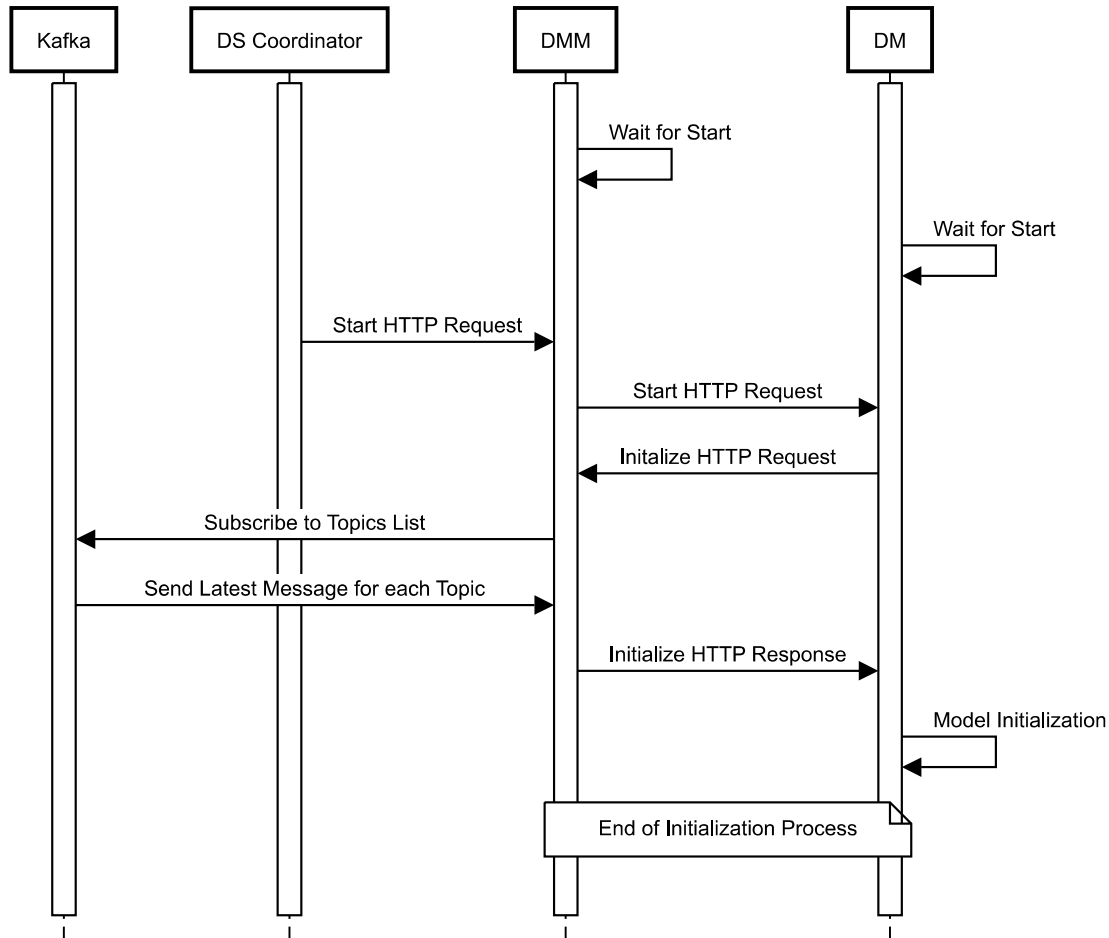


Figure 6.4 UML Sequence Diagram of DMM-DM Initialization

ferent simulation modeling techniques can be implemented in a DM, such as; mathematical, statistical, behavioral, process-flow, or state-flow. DM can also host a translation model, which transforms information from one physical domain to another. For example, a translator DM can be used to translate robot position and orientation into a task progress state, which can be then used in a process-flow simulation to estimate task processing time.

DM shadows the physical system. DM must be initialized by the latest physical system state before starting the simulation. Moreover, DM needs to pause between time steps in order to receive any changes in the physical system. This implies the need to control DM model flow programmatically. Therefore, a model controller component is needed for each DM to initialize, pause, and resume the simulation. The model controller uses API provided by the modeling tool in order to control the model's flow. Controlling user-programmed models is done through direct programming. However, commercial modeling tools will require an API

interface to be used in a DM.

Since commercial modeling tools require licensing, DM Docker file must contain the needed steps to activate a license. This can be done by either starting the license server application or adding the needed environment variables loaded by the modeling tool. License files and credentials are encrypted as secrets in the K8s cluster. This ensures the execution of the dockerized model without exposing the license to any unauthorized third party. Another approach is to deploy these models as executable applications after exporting them from commercial tools. These applications can be easily dockerized after adding all necessary dependencies.

Since DM is deployed as a Docker container on the K8s cluster, K8s deployment script is developed for each DM. The K8s deployment script defines the Docker image version and configuration. K8s configuration script defines how DM model should behave using the following settings:

- *Cont_Time_Indicator* defines the model time flow as continuous-time or discrete-time.
- *Time_Resolution* defines the time-step used to advance model time if the model type is continuous-time.
- *Online_Indicator* defines if the model works in online or offline mode.

In order for DM to communicate with DMM and receive changes in the physical system, HTTP servlet is integrated into DM to enable the microservices interface. HTTP servlet can serve as a RESTful API for stateless DMs, such as translators and time-independent models. While it serves as a normal web API for stateful DMs.

DM provides two main services through HTTP servlet. Firstly, DM can be asked to update its state of the physical world. DM receives one or more physical attributes at each update state request. Secondly, DM can be asked to advance the model in time. Separating the two services is crucial as DM can receive information from multiple source gateways and DMs. Therefore, DMM passes any new Kafka message from the Pub/Sub communication channel as soon as it is received. While message sources are predetermined as subscribed Kafka topics. In addition to the previous services, DM provides a set of managerial services to DMM in order to control DM. DMM can request to start or stop the DM simulation. Moreover, DMM can request the latest DM state when needed. Figure 6.5 shows a sequence diagram for the HTTP requests passed between DMM and DM for a discrete-time DM. While Figure 6.6 shows the sequence diagram for a continuous-time DM.

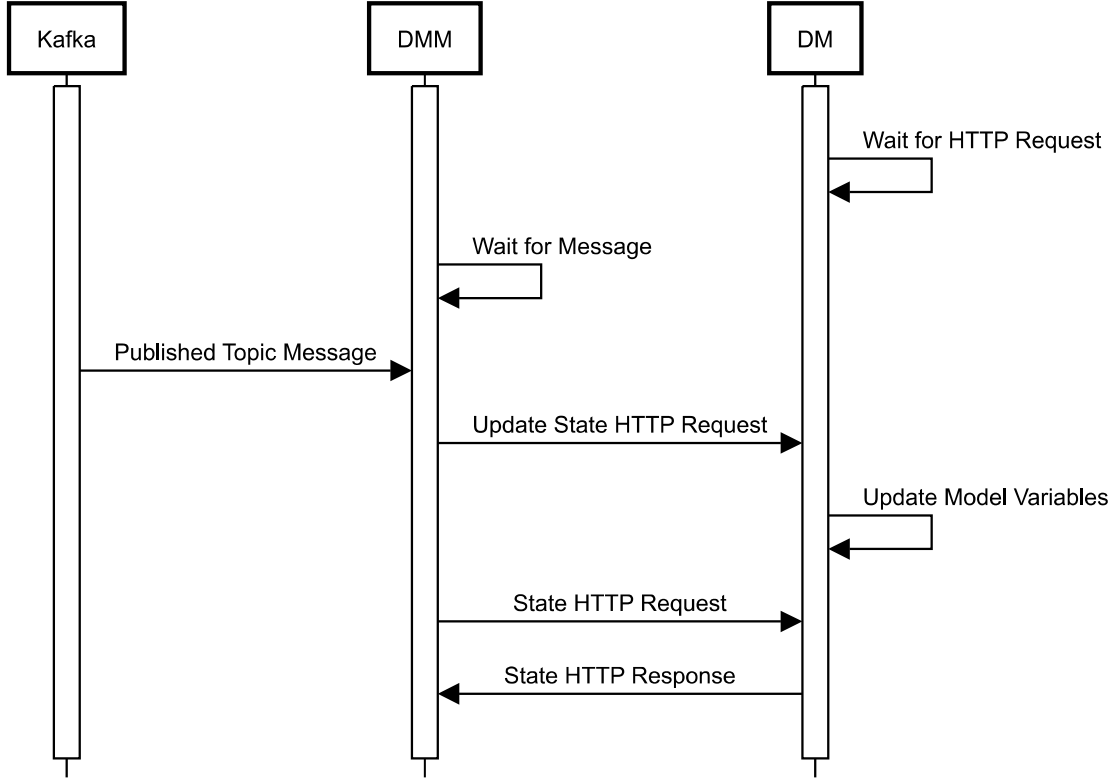


Figure 6.5 UML Sequence Diagram of DMM-DM Communication for Discrete-Time DM

Integrating HTTP servlet into DM depends on the modeling tool's flexibility. For example, ML models can easily integrate HTTP servlets by adding a few lines of code, since they have high programmatic flexibility. Other modeling tools might need an external interface to communicate and control a model. A DM external tool can connect to the model using either a TCP/IP communication or Inter-process communication (IPC). Figure 6.7 shows the different approaches to connect a DM with its DMM.

The proposed DS framework allows DM to run locally instead of running it on the K8s cluster. This could be helpful for running models that can not be dockerized, or for testing models before deployment. DM still needs to integrate an HTTP servlet to communicate with DMM. However, the K8s configuration script defines how DMM can connect to the local DM using the following settings:

- *Model_Ref* defines the IP address of the server hosting the local model.

Deployment of DM locally would limit the scalability of DM as only one instance of the model can run. Moreover, local deployment might require some manual operations, if local deployment is not automated.

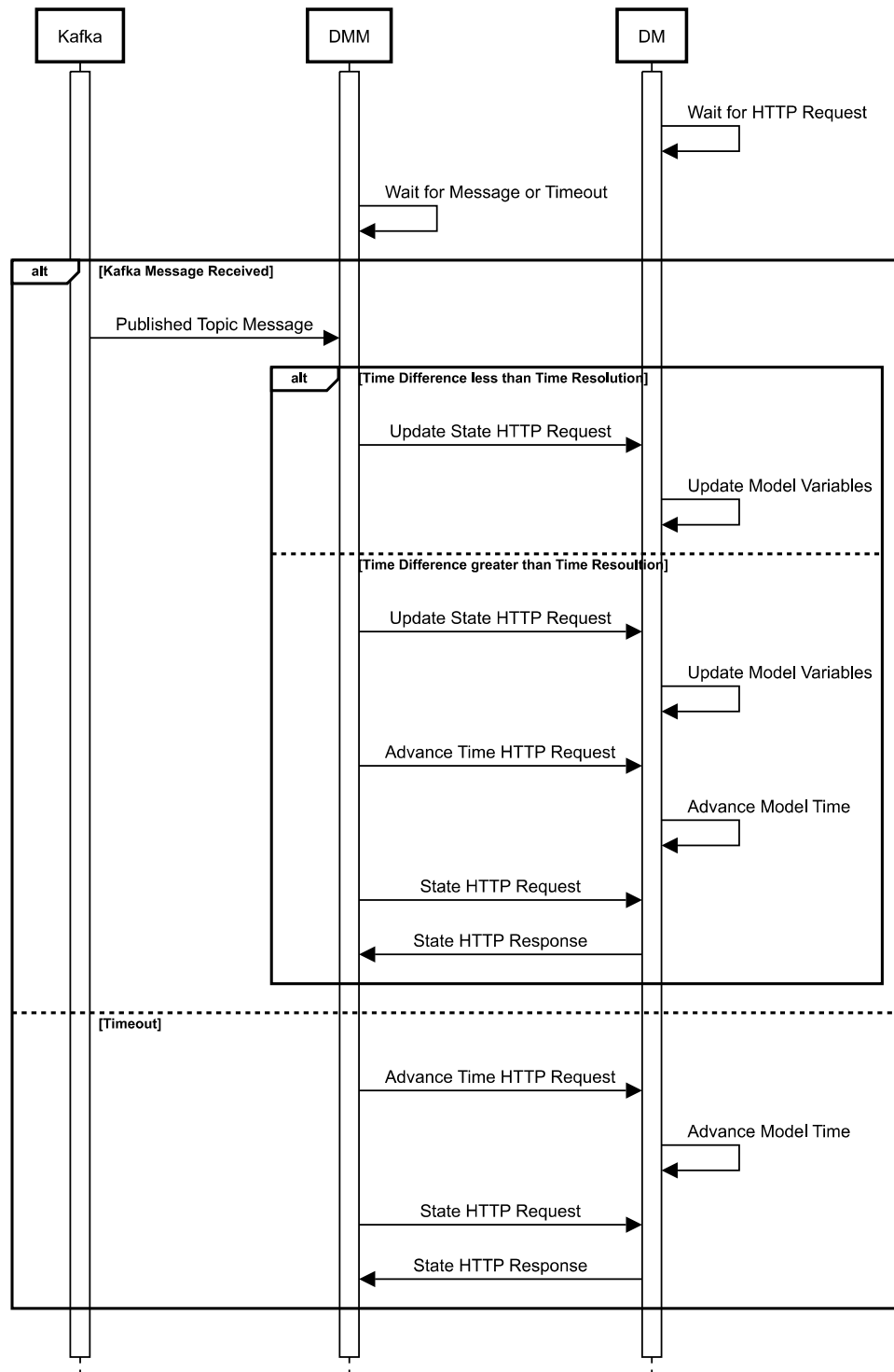


Figure 6.6 UML Sequence Diagram of DMM-DM Communication for Continuous-Time DM

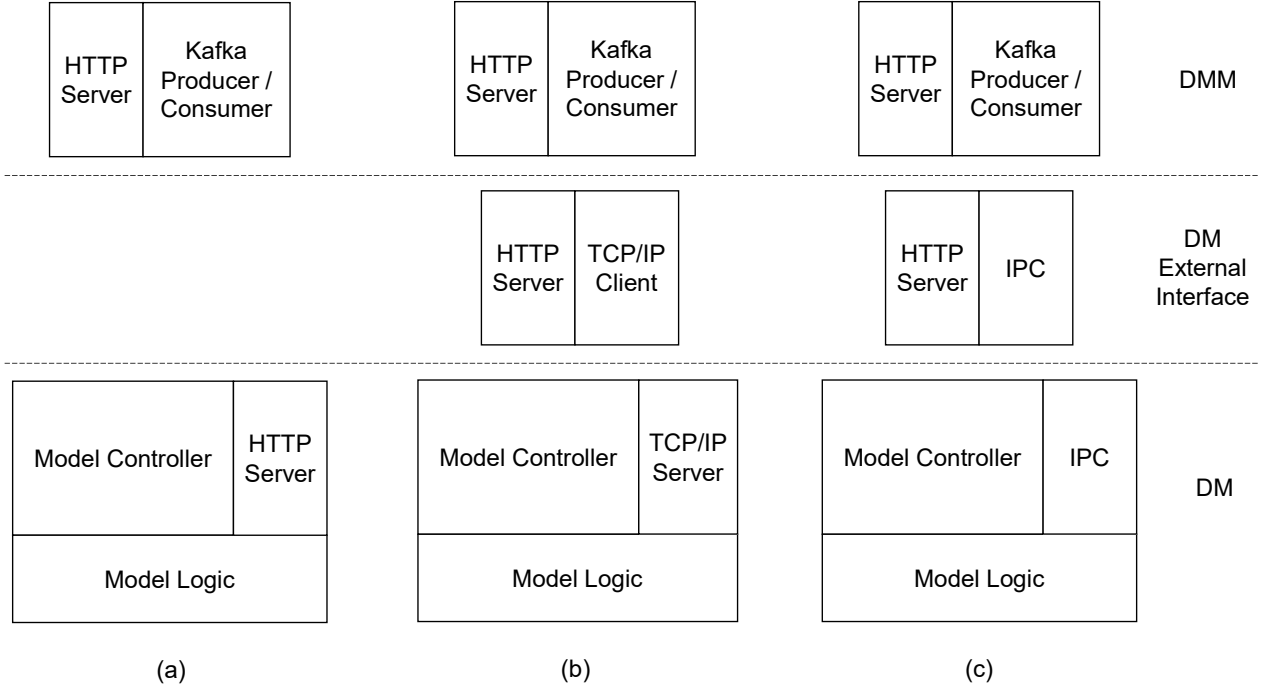


Figure 6.7 Digital Model Interface with the Digital Model Manager

6.6 Case Study

The proposed DS framework is applied to a natural gas compressor. The gas compressor is a part of an offshore gas transportation pipeline accessed through Open Industrial Data (OID) project [1]. OID is a collaboration project between Aker BP and Cognite to provide a stream of industrial data publicly for researchers. Aker BP publishes sensors measurements captured at the first stage of a four stages gas compressor on the Valhall PH platform, CDF provides access to the gas compressor's sensors measurements through their web servers.

The gas compressor stage is composed of multiple components as shown in Figure 6.8. The gas is cooled in the suction cooler *23-HA-9103*. Liquid droplets are removed in the suction scrubber *23-VG-9101*, while the cooled gas is compressed by the fixed-speed centrifugal compressor *23-KA-9101*. The compressed gas is cooled in two discharge coolers *23-HA-9114* and *23-HA-9115*. A set of valves controls the gas flow to and from the compressor stage. An anti-surge valve is used for surge control. The anti-surge valve passes the pressurized gas back to the suction scrubber during the surge to prevent reverse flow within the compressor.

The proposed DS framework is deployed on Google cloud platform (GCP) using Google Kubernetes engine (GKE). DS coordinator creates a K8s cluster composed of 8 standard E2 nodes. Each node has 2 shared virtual CPUs with 4GB RAM memory and 100GB storage

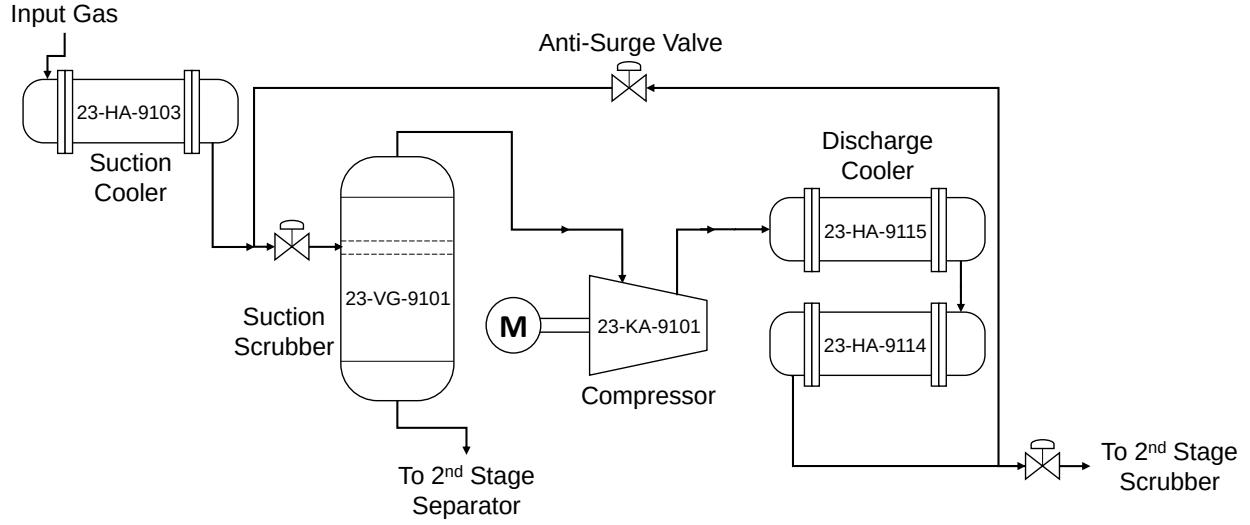


Figure 6.8 First Stage Natural Gas Compressor System [1]

memory. Three Kafka brokers are deployed as a Kafka cluster for Pub/Sub communication. Figure 6.9 shows different server locations that interacts within the DS framework. Five DMs are used to describe different aspects of the physical system as shown in Figure 6.10.

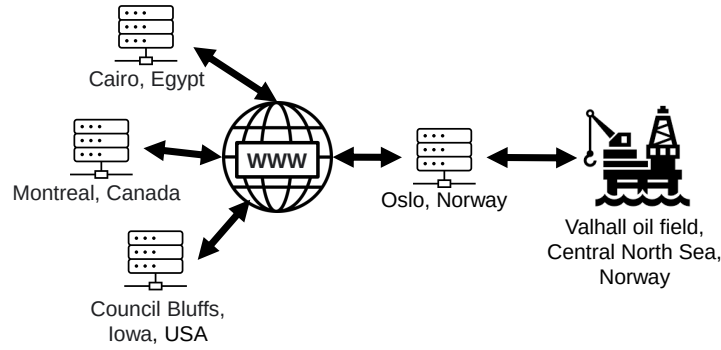


Figure 6.9 Distributed System Servers

6.6.1 Source Gateway

The gas compressor information is provided only through the CDF platform. CDF provides a RESTful API interface for data access. Each component on-site is defined by a tag name in the CDF platform. HTTP requests are used to pull live and historical data. Figure 6.11 shows pressure and temperature sensors' measurements for the in-flow and out-flow of the compressor. In this study, online sensor measurements are collected over the span of 33

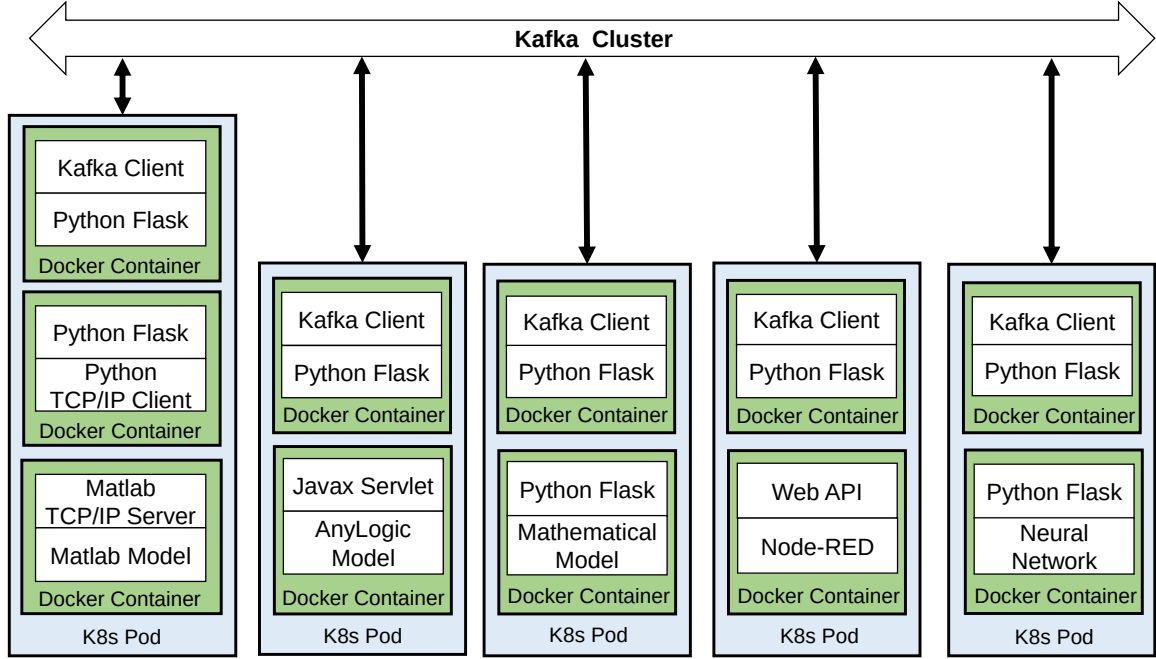


Figure 6.10 Digital Models Used to Represent the System

hours using 1-minute granularity. Missing data is pulled as empty messages from the CDF platform.

Loosely-coupled source gateway is developed and deployed on the DS framework. The source gateway connects to CDF using a secure HTTP connection. Source gateway pulls sensor measurements according to pre-defined tag names for the available sensors. Table 6.2 describes the list of tag names pulled by the source gateway.

6.6.2 Gas Pressure Model

Estimating compressed gas pressure is important to assess prior knowledge of the system and predict pressure surges. ML provides a statistical approach to predict system behavior based on historical data. ANN models have been used in literature for estimating gas compressors' health index and detecting anomalies [120]. ANN provides high-accuracy estimates in a short evaluation time; however, a large amount of processed data is needed for training ANN.

A Pytorch ANN model is developed to estimate the out-flow gas pressure after compression. The ANN model was trained on historical data collected over the span of 1 month. Data points with missing information were removed from the training dataset to clean the training data for ANN. The ANN model is designed as a small 2 hidden layered network, where the

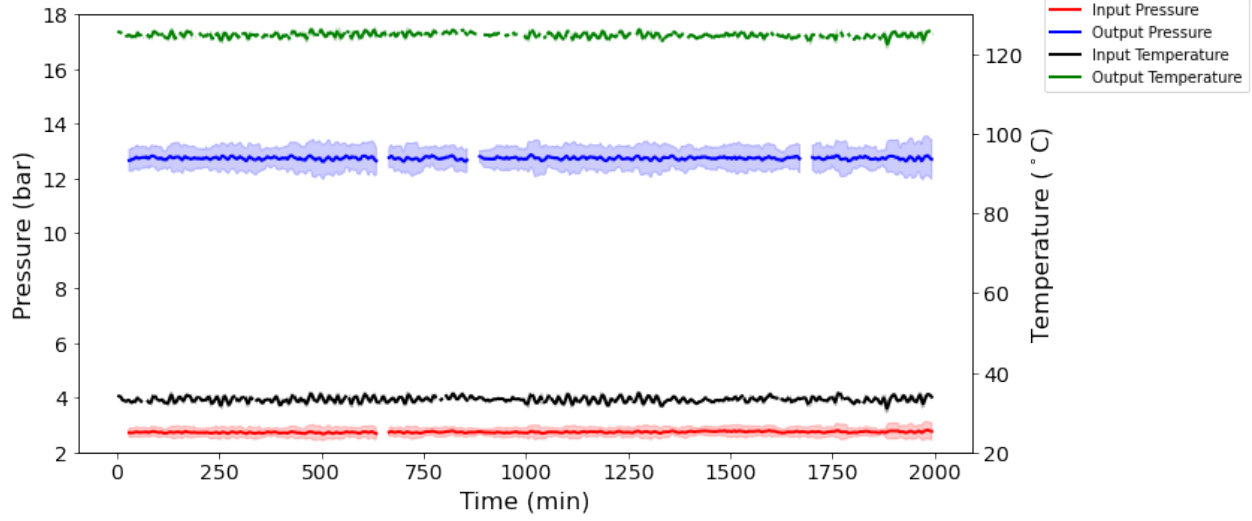


Figure 6.11 Compressor's Pressure and Temperature Sensors Measurements

Table 6.2 Description of Tags Pulled from CDF

CDF Tag	Description
<i>VAL_23-TT-92533</i>	Temperature of in-flow gas
<i>VAL_23-PT-92532</i>	Pressure of in-flow gas
<i>VAL_23-PDT-92534</i>	Differential Pressure of in-flow gas
<i>VAL_23-TT-92539</i>	Temperature of out-flow gas
<i>VAL_23-PT-92539</i>	Pressure of out-flow gas
<i>VAL_23-FT-92537-01</i>	Mass Flow of out-flow gas
<i>VAL_23-LT-96182-01</i>	Lube oil reservoir level indicator 1
<i>VAL_23-LT-96182-02</i>	Lube oil reservoir level indicator 2
<i>VAL_23-LT-96182-03</i>	Lube oil reservoir level indicator 3

first layer has 32 nodes and the second layer has 8 nodes.

The trained ANN model is exported to a DM that is deployed on the DS framework. DM is developed to load the ANN model and estimate the gas pressure using the latest physical system state. The developed DM is configured as a continuous-time model with a 1-minute time-step. DMM sends an *update state* request to DM when it receives a change in a sensor value. While an *advance time* request is sent to DM every 1 minute, where the ANN model generates a new estimate for the out-flow gas pressure. Figure 6.12 shows the ANN predicted gas pressure and the measured gas pressure during the DS framework deployment test. The ANN predicts the out-flow gas pressure with an R-squared (R^2) of 0.74 and mean absolute error (MAE) of 0.053. This means that the error difference between ANN estimated pressure

and measured pressure is relatively low. The ANN also generates a pressure prediction with missing information, since DMM keeps the latest received measurements from the source gateway.

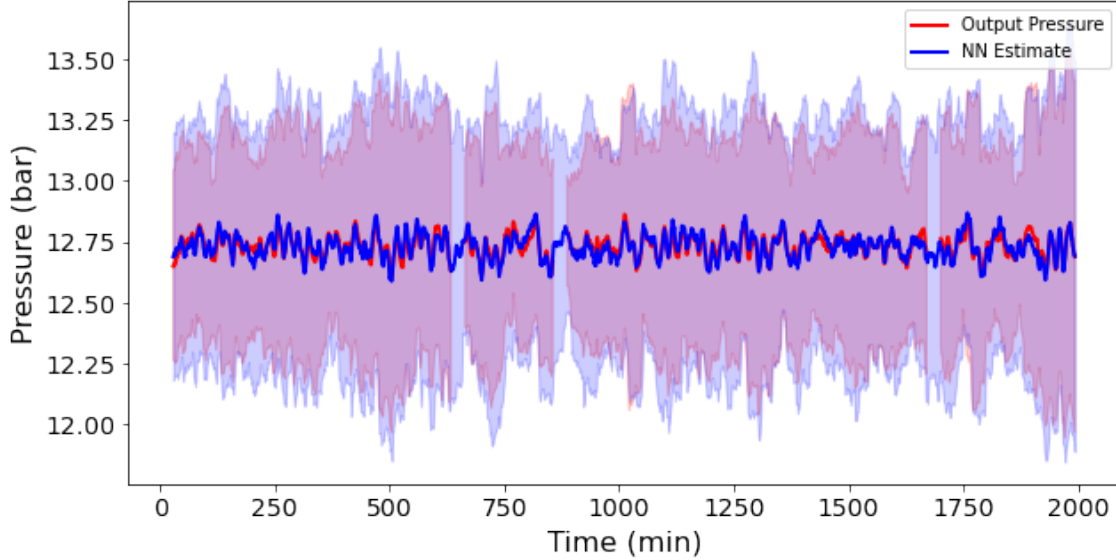


Figure 6.12 ANN Model Estimate of the Compressed Gas Pressure During DS Operation

A Flask web API is used to communicate between DM and DMM. DMM subscribes to the following Kafka topics, where *%Time%* is the creation time of the source gateway.

- *CogniteGateway_%Time%_VAL_23-TT-92533*
- *CogniteGateway_%Time%_VAL_23-PT-92532*
- *CogniteGateway_%Time%_VAL_23-PDT-92534*
- *CogniteGateway_%Time%_VAL_23-TT-92539*
- *CogniteGateway_%Time%_VAL_23-FT-92537-01*

DMM publishes the estimated pressure on the following Kafka topic:

- *GasPressureModel_%Time%_OutPressure*

6.6.3 Gas Compressor Model

Another approach is to estimate out-flow gas pressure using gas laws. The gas compressor is modeled as a polytropic system [121]. The pressure-temperature relationship equation is

implemented in Matlab. The compressor's efficiency map η is estimated based on the compressor's specifications and historical data. The pressure-temperature relationship equation is applied as shown in Eq. 6.1.

$$T_{out}/T_{in} = (P_{out}/P_{in})^{f(\eta)} \quad (6.1)$$

The Matlab model is exported to a standalone application that is packaged into a docker image. The Matlab docker image is deployed as a DM on the DS framework. Similarly to the ML-based DM, the Matlab-based DM is configured as a continuous-time model with 1 minute time-step. Figure 6.13 shows the Pressure-Temperature estimated gas pressure and the measured gas pressure during the DS framework deployment test. The wider confidence interval of the Pressure-Temperature model indicates that the model abstraction does not include all the principal factors of the system. This means other factors as the mass flow of the out-flow gas affects the pressure estimates.

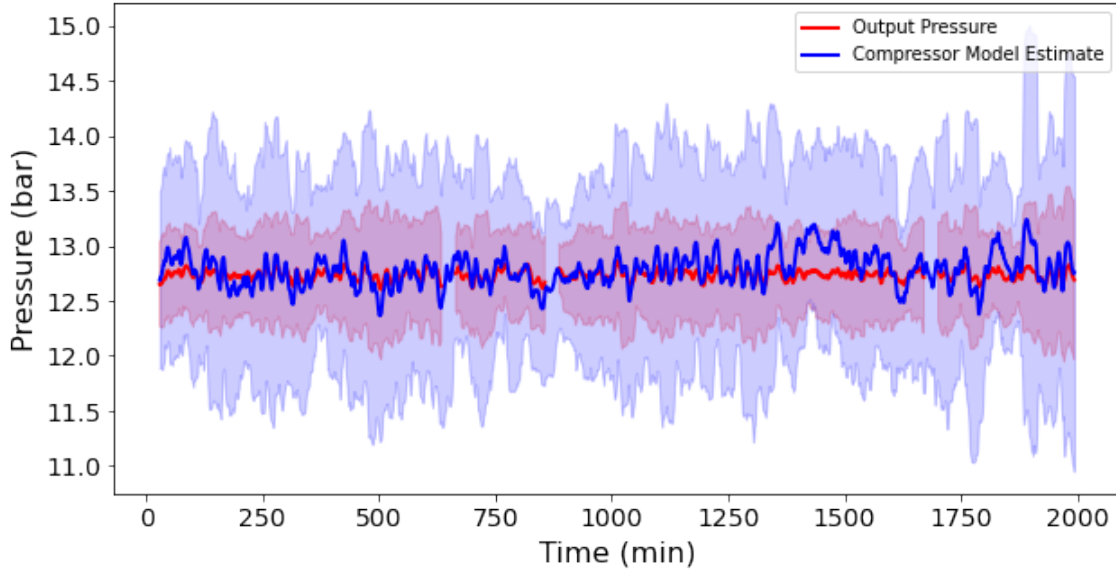


Figure 6.13 Pressure-Temperature Model Estimate of the Compressed Gas Pressure During DS Operation

Python Flask web API and Matlab TCP/IP are used to communicate between DM and DMM as shown in Figure 6.10. DMM subscribes to the following Kafka topics:

- *CogniteGateway_%Time%_VAL_23-TT-92533*
- *CogniteGateway_%Time%_VAL_23-PT-92532*

- *CogniteGateway_%Time%_VAL_23-TT-92539*

While it publishes the estimated out-flow gas pressure on the following Kafka topic:

- *CompressorModel_%Time%_OutPressure*

6.6.4 Gas Compressibility Model

The gas compressibility factor describes the deviation of the gas from ideal gas properties [122]. Gas compressibility estimate helps in correcting the estimation of gas volume. A Python-based mathematical model is developed to estimate gas compressibility coefficient z . Compressibility coefficient correlation is implemented based on the model developed by Kareem et al. [122] as shown in Eq. 6.2.

$$z = \frac{1 + A^2}{C} - \frac{A^2 * B}{C^3} \quad (6.2)$$

where $A = a_1 * t * e^{a_2 * (1-t)^2} * P_{pr}$,

$B = a_3 * t + a_4 * t^2 + a_5 * t^6 * P_{pr}^6$,

$A = a_9 + a_8 * t * P_{pr} + a_7 * t^2 * P_{pr}^2 + a_6 * t^3 * P_{pr}^3$,

$t = \frac{1}{T_{pr}}$,

$a_i = \text{constant } \forall i \in \{1 \rightarrow 9\}$.

The Python model is deployed on the DS framework as a continuous-time model with a 1-minute time-step. A Flask web API is used to communicate between DM and DMM. DMM subscribes to the following Kafka topics:

- *CogniteGateway_%Time%_VAL_23-TT-92539*
- *CogniteGateway_%Time%_VAL_23-PT-92539*

While it publishes the estimated gas compressibility coefficient on the following Kafka topic:

- *CompressibilityModel_%Time%_zCoefficient*

6.6.5 Lube Oil Reservoir Model

Lube oil is pumped into the compressor system to lubricate moving parts, before returning back to the reservoir. Lube oil level indicates if a failure occurred in the compressor cooling cycle. An AnyLogic® behavioral simulation model is developed to simulate lube oil level in a

reservoir. DM uses AnyLogic® fluid library to simulate the lube oil reservoir as a fluid tank. The model estimates the flow of the oil to keep the oil level according to the measurements of 3 level indicators. Figure 6.14 shows the main components in the AnyLogic® model.

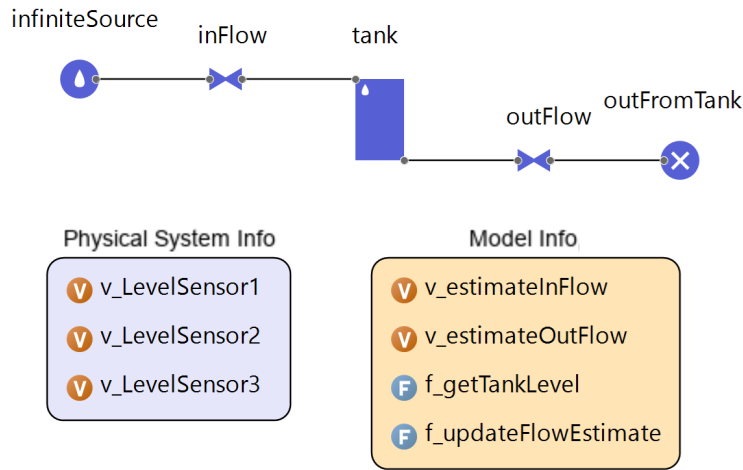


Figure 6.14 Lube Oil Reservoir AnyLogic® Model

The AnyLogic® model is exported to a standalone Java application, which is deployed on the DS framework. A docker image is developed for running the exported Java application. A Java servlet API is used to communicate between DM and DMM. DMM subscribes to the following Kafka topics:

- *CogniteGateway_%Time%_VAL_23-LT-96182-01*
- *CogniteGateway_%Time%_VAL_23-LT-96182-02*
- *CogniteGateway_%Time%_VAL_23-LT-96182-03*

While it publishes the estimated in/out flow on the following Kafka topics:

- *OilTankModel_%Time%_InFlow*
- *OilTankModel_%Time%_OutFlow*
- *OilTankModel_%Time%_Level*

6.6.6 Visualization

Visualizing the current system state and model estimations is important for decision-makers. BI dashboards are interactive web applications that present statistical information graphically. A Node-RED flow is designed to visualize the state of the real system and different

DMs. Node-RED listens to information passed from DMM using an HTTP web server. Then Node-RED updates multiple graphs and gauges used for visualization purposes. DMM subscribes to the following Kafka topics:

- *CogniteGateway_%Time%_VAL_23-PT-92539*
- *GasPressureModel_%Time%_OutPressure*
- *CompressorModel_%Time%_OutPressure*
- *CompressibilityModel_%Time%_zCoefficient*
- *OilTankModel_%Time%_Level*

6.7 Discussion

The main focus of the use-case experiment is to prove the generality and practicality of the proposed framework. Therefore, simulation model accuracy is not discussed in detail, since the goal is to integrate different model types into the DS framework. The five deployed DMs show the proposed framework's ability to incorporate different modeling approaches such as statistical, mathematical, and behavioral modeling. While the DMs are developed using different modeling tools such as ANN, Matlab, and AnyLogic.

In order to identify the benefits of applying distributed system concepts, three aspects of the proposed DS framework are discussed. Throughput is measured to prove the real-time capabilities of the DS framework. DS scalability and fault-tolerance are also discussed since framework generality requires the DS ability to simulate new models and handle faults during runtime. Other aspects such as cloud computing cost and power consumption depend on the use case. These aspects are not considered in this discussion since they vary based on the number of DMs deployed and their container image size.

6.7.1 DS Throughput

Concurrent utilization of resources increases system throughput. However, the communication network affects the overall system performance. Prometheus and Grafana are used to monitor the Kafka cluster performance and Kafka messages throughput, while computation time is recorded at each DM. Table 6.3 shows average time taken at each microservice in DS framework. DMs take milliseconds to update the model state and advance the model in time, while the added DMM communication overhead due to HTTP servlets is still in

milliseconds. Table 6.4 shows that the Kafka cluster provides a low-latency high-throughput communication system. Kafka message throughput is relatively low due to low network traffic, therefore the Kafka cluster can handle more messages without reducing its performance. The timing analysis proves DM ability to shadow the physical system in real-time.

Table 6.3 DS Components' Performance

DS Component	Service	Avg Time
Source Gateway	Publish Physical State	185 ms
DMM	Update Model	154 ms
	Publish Model State	183 ms
Gas Pressure DM	Update State	0.08 ms
	Advance Model Time	0.1 ms
Gas Compressor DM	Update State	0.12 ms
	Advance Model Time	0.17 ms
Gas Compressibility DM	Update State	0.07 ms
	Advance Model Time	0.09 ms
Lube Oil Reservoir DM	Update State	0.09 ms
	Advance Model Time	0.12 ms

Table 6.4 Kafka Cluster Performance

Performance Indicator	Value
Message Throughput	34.3 msg/s
Produce Time Latency	4.75 ms
Consume Time Latency	13.5 ms
Fetch Request per Broker	250 req/s
Produce Request per Broker	6.7 req/s

6.7.2 Scalability

K8s scales up any service by replicating K8s pods of the deployed service [118, 119]. This property of K8s is used to run multiple DMs of the same model. The results of these independent runs are then used for randomness tests. Each of these independent DMs connects to the Pub/Sub communication channel to receive physical system information. DMs also publish their status to the Pub/Sub channel by adding their replication number to the model name.

K8s provides a rolling update policy, which enables DS framework to replace a DM with a newer version without interrupting the framework's operations. DS coordinator deploys new

source gateways or DMs on the current K8s cluster without terminating the framework. K8s redistributes the K8s pods if the number of K8s nodes changed due to scaling up or down.

6.7.3 Fault Tolerance

One of the main benefits of distributed systems is keeping the service available even during runtime faults. Apache Kafka® has a set of policies to support fault tolerance [116]. These policies ensure a single write/read for information on the Kafka cluster. Moreover, distributed Kafka brokers increases data availability by keeping multiple replicas of a Kafka topic on different brokers. While K8s deployment script deploys each Kafka broker on a different K8s node to ensure service availability during hardware faults.

Containerization of DM enables DS framework to run each DM in complete isolation. Therefore, a failure in one DM does not affect the rest of the framework. Moreover, the K8s control plane tracks the number of deployed K8s pods to schedule any new deployment or restart any failed service [119]. This means DM can be designed to adapt a fail-safe policy. Whenever a runtime exception occurs in the model, it terminates the Docker container. The K8s control plane then restarts the terminated container. A new DM container is then initialized by the latest physical system state and resumes its operation.

6.8 Conclusion

DT is a trending research topic in the Industry 4.0 era. However, the difference in definition and roles between DT and DS is still blurry. In this paper, the differentiation between the two concepts based on functional roles is proposed. DS provides a real-time virtual replica of the physical system. While DT provides intelligence and decision-support services. This proposed definition considers DS as a core component of DT.

Despite the potential benefits of applying distributed systems concepts in the industrial domain, the number of publications that applies these concepts are few. In this paper, a DS framework is proposed based on distributed systems concepts. DS framework is built based on an event-driven architecture using Apache Kafka® as its communication backbone. DMs are encapsulated in Docker containers and deployed on K8s cluster. Microservice APIs are used to design a loosely-coupled communication system between different DS components.

The proposed DS framework is applied in a natural gas compressor case study. Access to the gas compressor sensors' readings is provided through the OID project. A loosely-coupled gateway pulls the physical system status from the CDF platform. Four different DMs were developed to model different parts of the system with different abstraction levels. These DMs

were developed using different modeling tools and programming languages. Additionally, a Node-RED interface is developed as a DM to visualize the main simulated attributes of the physical system. Experimental results prove the DS framework’s ability to accommodate different modeling tools. Pub/Sub communication channel and microservices API enable DS to shadow the physical system in real-time.

6.9 Future Work

For future work, the proposed DS framework will be used as a core component to create a realistic general-purpose DT framework. The developed DT will be used to provide system-level applications and intelligence for a semiconductor manufacturing system. DT framework will be implemented using the same distributed systems concepts and technologies. Additionally, a systematic interface for information sharing between DS and DT will be developed. This approach will enable DT applications to collect needed information from multiple DMs and even other DT applications. While distributed systems technologies such as Kafka, Dockers, and K8s will ensure DT scalability and fault-tolerance.

**CHAPTER 7 ARTICLE 4: GENERAL PURPOSE DIGITAL TWIN
FRAMEWORK USING DIGITAL SHADOW AND DISTRIBUTED SYSTEM
CONCEPTS**

Ayman AboElHassan, Ahmed Sakr, Soumaya Yacout

Submitted to:

Computers & Industrial Engineering

Date of submission: 10-Nov 2022

7.1 Abstract

Digital twin (DT) is an emerging concept in the industry 4.0 era. It integrates intelligence into industrial processes. The broadness of DT's concept allows for multiple definitions and designs to be labeled as DTs. However, general-purpose DTs are still out of reach due to the lack of consensus, tool dependency, and the focus on system integration. In this paper, a general-purpose DT framework is proposed by utilizing distributed system concepts. Data acquisition, system modeling, and intelligence applications are modularized in order to enable framework generality. Digital shadow (DS) is implemented as a core component of DT, which provides a real-time virtual replica of its physical counterpart. DT applications utilize DS models for recommending intelligent decisions. A systematic modeling approach is designed by differentiating between DS models and DT applications. The proposed approach defines the model flow based on its roles and objectives. The proposed DT framework is applied to a semiconductor production use case. For this system, the proposed DT provides insights into the production system's behavior and estimates product completion time. Operators dispatch work-in-progress products using DT recommendations based on the equipment setup and the production system's status.

Keywords

Digital Twin, Digital Shadow, Distributed Systems, Modeling, Simulation, Production System

7.2 Introduction

DT is an emerging concept that focuses on embedding simulation, AI and digitalization in the industrial domain. NASA researchers first defined DT as an integrated multi-scale multi-physics simulation of the physical system [17]. The goal of DT is to provide insights for decision-makers in order to improve the system's performance. DT definition expanded to become a real-time digital replica of a product, equipment, system, or system of systems [14,15]. The expanded DT definition incorporates real-time monitoring, simulation, ontology-based knowledge, optimization, and AI.

7.2.1 Motivation

The broadness of DT concept created multiple definitions and roles of DT [47]. Some studies focused on the real-time monitoring property of DT [48, 49], where DT collects real-time

data from multiple sources for data analysis and visualization. Other studies focused on the simulation capabilities of DT [15, 18, 27]. Finally, some studies focused on the dataflow and connectivity between multiple DTs components [16, 25, 50, 51]. The common aspect between the previous studies is their lack of consensus on DT definition. Additionally, these studies develop DT systems for specific applications. The development of a general-purpose DT is still infeasible in literature.

The lack of conceptual standardization and generalization in DT frameworks is addressed in this paper. In a previous publication, DT is defined as "*real-time decision-support sandbox*" [103]. DT sandbox is a virtual environment for simulation models and decision-support applications to operate. This definition specifies DT role in the decision-making process, rather than technologies and tools utilized. DT generates insights using artificial and rule-based intelligence. This expands the set of tools used in DT applications from behavioral flowcharts to ML algorithms. General-purpose DT is enabled through the separation between data capturing, behavior modeling and simulation, and intelligence applications.

In this paper, DS concept is used in designing a general-purpose DT. DS is an artifact concept of DT [29]. The separation between DS and DT enables the generalization of the DT framework [2, 25, 71]. In a previous publication, DS is defined as "*digital replica of the physical system that matches its physical counterpart's state in real-time*" [103]. DS applies different modeling techniques and prior knowledge in creating a virtual replica of the system. The virtual replica forms a knowledge graph used by DT applications for decision support.

Different modeling approaches are used in developing DT models. Qi et al. [27] 5-dimension DT defined DMs as a dimension. DT models are 3D CAD, physical models, behavioral models, or rule-based models. Since DT and DS have different roles, simulation models are designed differently for both. DS requires real-time simulation of system behavior. While DT predicts future system states to generate insights. A systematic modeling approach is proposed for designing DS models and DT applications.

7.2.2 Contribution

This paper's contributions answer the following research questions;

RQ1 How to design a general-purpose DT framework?

A general-purpose DT framework provides real-time intelligence regardless of the application scope and system properties. General DT generates insights using multiple different models and tools created by different vendors. In this paper, distributed systems concepts are used in designing a general-purpose DT framework. Distributed

systems enable DT generalization through modularization of DT components and integration of these different components using Pub/Sub communication and microservices. The originality of the proposed DT framework is in the modularization and isolation of DS models and DT applications. DT forms a sandbox for different applications to run concurrently. DS and DT exchange information through a Pub/Sub communication channel.

RQ2 How to design DS models and DT applications for deployment on the DT framework? Simulation models are used in both DS and DT; however, the same system can be modeled in different ways. In order to systemize the process of model design, multiple modeling aspects are considered. DT applications are developed differently from DS models since the model's objective determines the used modeling approach.

7.2.3 Organization

The rest of the paper is organized as follows; Discussion of the related work is presented in Sect. 7.3. The proposed DT framework is presented in Sect. 7.4. Then a systematic approach for designing DT and DS models is described in Sect. 7.5. The proposed DT framework is applied in a semiconductor production system use case, which is presented in Sect. 7.6. Finally, the paper's conclusion and future work are presented in Sect. 7.7 and 7.8.

7.3 Related Work

7.3.1 Digital Twin

In 2010, NASA defined DT as a real-time multi-scale multi-physics simulation used for real-time prediction of faults in the real vehicle [17,18]. The DT concept did not gain momentum till the start of the Industry 4.0 era. DT opportunities in the industrial domain were discussed by [20]. Since then the number of publications discussing DT concept, frameworks, and applications has increased exponentially [13,48]. DT systems have been categorized according to their components and the corresponding physical system [14,15,27]. However, some studies categorized DT based on their functionality and proximity to the physical system [28,34].

Multiple DT definitions were proposed in the literature other than NASA's definition. DT is widely defined as a digital replica of a product, equipment, system, or system of systems [14,15,123]. Rosen et al. [20] DT is the combination of physical and virtual systems where a change in one is reflected in the other. Grieves and Vickers [16] DT is the collection of information that provides a virtual representation of the product throughout its lifetime. Tao

et al. [69] compared between DT and CPS, where CPS is more scientifically driven. Brauner et al. [51] DT is a digital representation of objects and processes of the real world.

DT frameworks utilize existing standards for communication, data security, modeling, and data storage. However, standardization of DT is still an ongoing process. ISO 23247 [59] is a generic guideline for developing DT systems. ISO 23247 provides DT implementation approaches for case-specific systems. However, ISO 23247 focuses on the definition of OME and how to connect OME to DT system [61]. Other studies proposed the implementation of DT based on RAMI 4.0 as a part of AAS [55–58]. IEC 62832 [60] provides a high-level digital factory framework for maintaining a digital representation of a production system. IEC 62832 focuses on information sharing between different components of a digital factory.

7.3.2 Digital Shadow

Digital shadow (DS) concept is a by-product of DT, which was first defined in 2016 by WGP [30,31]. WGP defined DS as a general image of the physical system formed using real-time information collected from multiple sources. WGP’s DS definition is interpreted by researchers in 2 ways. The first approach defines DS as a one-directional DT [8,14,29,32,33]. Stark and Damerau [15] DS is the collection of real-time information. Konstantinov et al. [76] DS keeps track of the physical system state. The second approach defines DS as a more abstract version of DT that represents a part of the system [8,30,31]. Becker et al. [72] DS is a compact purpose-driven model representing part of the physical system. Ladj et al. [73] DS is ontology-based data-models capturing the physical system state. Kassen et al. [77] DS is a data-driven simulation model designed to accommodate different types of production systems Brauner et al. [51] DS is a task-driven domain-specific simulation model describing part of the physical system in real-time.

DS and DT definition and roles overlap due to the lack of separation between the 2 concepts. Stark and Damerau [15] proposed an intersected functionality between the two terms, where a DT is the digital model and its shadow. Ladj et al. [73] limited DS’s role to real-time data acquisition similar to WGP. Other studies differentiate between DS and DT based on model fidelity [30,72], or the dataflow direction [76,77].

AboElHassan et al. [103] separates between DS and DT according to functionality as illustrated in Figure 7.1. DS collects real-time information from the physical system and creates a detailed digital representation of the system entities. DT supports decision-makers through BI applications, decision recommendation, and simulation models. Functionality-based separation between DS and DT enables the generalization of the DT framework [2,25,71].

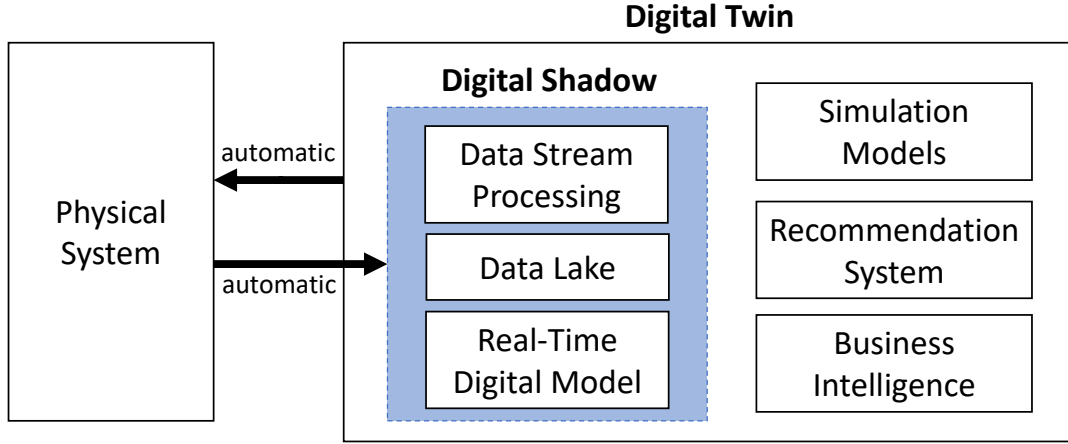


Figure 7.1 Functionality-based Separation between Digital Twin vs Digital Shadow

7.3.3 DT Models Development

The process of developing DT models has been discussed in the literature. Some studies focused on the modeling process. Lu and Xu [86] proposed a DT modeling process similar to the simulation model designing technique. Reed et al. [124], Moyne et al. [125] and Qamsane et al. [126] proposed a model development cycles for designing DT simulation model. Other studies focused on the DT system development lifecycle [34, 58]. Figure 7.2 summarizes the DT development process.

A standard description of DT models has been discussed to automate DT model integration and provide interoperability. Moreno et al. [127] proposed a Business-to-business (B2B) industrial marketplace, where machine vendors provide commercial DT models and data-driven services for their machines. Becker et al. [72] proposed the use of UML to provide a conceptual design of the DT, its components, and the information shared between them. Göppert et al. [58] proposed an ontology-based process flow for designing DT, where the ontology graph provides a descriptive semantic of system components and their connections. Wilking et al. [128] proposed the use of SysML diagrams for model generation. Co-simulation and FMI have been proposed for deploying simulation models in DT systems [46, 129].

7.4 Digital Twin Framework

In continuation to the work in AboElHassan et al. [2, 103], Digital twin (DT) is defined as a DS-based real-time decision-support sandbox. DT uses DS for creating a real-time virtual image of the physical system. DT framework deploys multiple intelligent applications, gener-

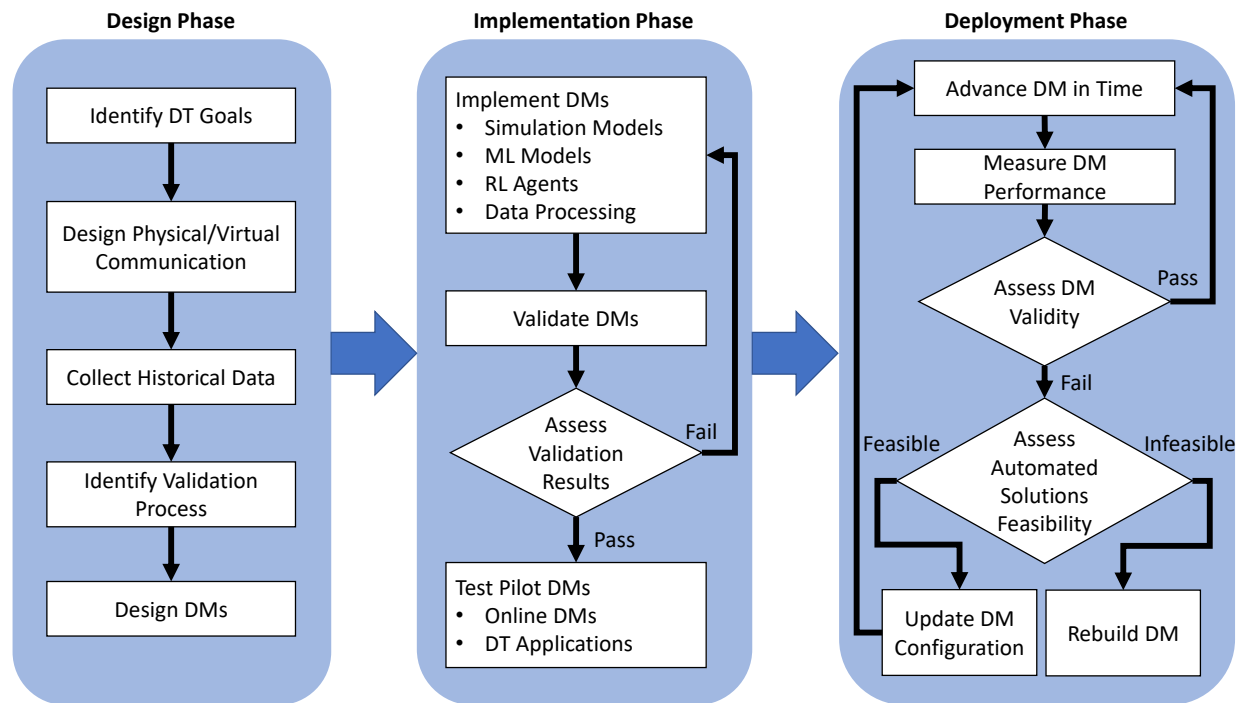


Figure 7.2 Digital Twin Development Process

ating insights for decision-makers. DT applications utilize different tools such as simulation, scheduling, optimization, ML algorithms, diagnosis, prognosis, and performance evaluation. DT roles are defined, but not limited to, the following activities:

1. Analyze the physical/virtual system's performance.
2. Evaluate production plan.
3. Optimize shopfloor operations.
4. Schedule maintenance operations.
5. Estimate cost and delivery date.
6. Perform virtual commissioning of equipment and products.

Distributed system concepts are used in designing the DT framework, as in AboElHassan and Yacout [2] DS. A shared Pub/Sub communication channel provides a data-sharing platform for different DT components. While microservices are used to interact with DT components using a loosely-coupled approach. Figure 7.3 shows the DT framework architecture design, which improved upon the previously proposed architecture in AboElHassan et al. [103]. The

novelty of the proposed DT framework is the use of distributed system concepts to separate between DT applications and DS models. The rest of this section describes the different components of the DT framework.

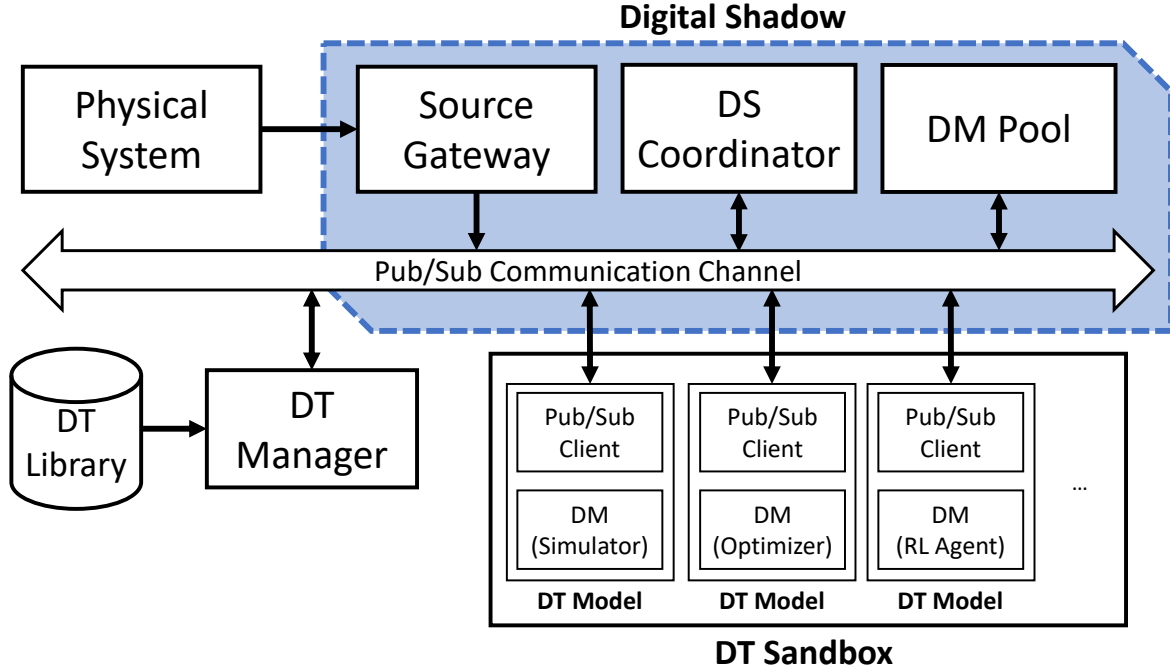


Figure 7.3 Digital Twin Framework Architecture

7.4.1 DT Manager

DT manager provides administrative control over the DT components. DT applications are deployed inside the DT sandbox by the DT manager using scripts and configuration files stored in the DT task library. In this sense, DT manager takes over some roles from the DS coordinator described in AboElHassan and Yacout [2].

The DT framework is designed for deployment on cloud services through Docker and K8s. Kubernetes (K8s) is an open-source orchestration system designed by Google engineers. K8s automates container deployment, scaling, and managing workload operations. As shown in Figure 7.4, K8s scheduler deploys containers inside a K8s node according to its computational load. K8s is used in deploying DT components as Docker containers and orchestrating computational load on the available resources. K8s cluster information is stored in a K8s persistent memory *etcd*. DT manager defines the state of each Docker container in the DT framework. K8s controller manager keeps the K8s cluster state as configured by the DT manager.

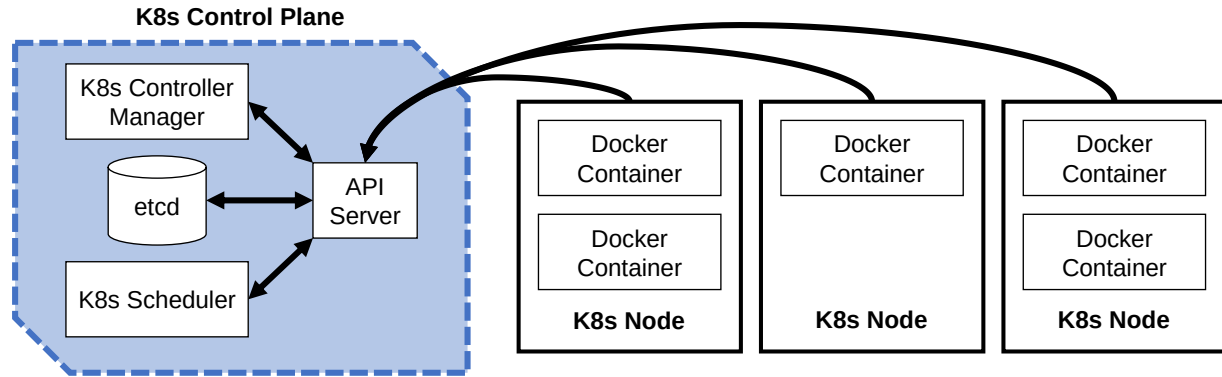


Figure 7.4 Example of a K8s Cluster

DT manager controls the K8s cluster that hosts the DT framework using K8s commands, as shown in Figure 7.5. A K8s deployment script defines how a module is deployed and connected to the rest of the system. While a K8s configuration script defines the parameters' values of the deployed module. A set of K8s scripts is developed for each DT component and stored in the DT task library. DT manager uses the K8s scripts in creating the DT components starting by Pub/Sub communication channel. After the deployment of DT components, DT manager provides administrative access to these components. DT manager starts, stops, and queries DT applications. Additionally, DT manager deploys new DT applications and updates existing ones with new versions.

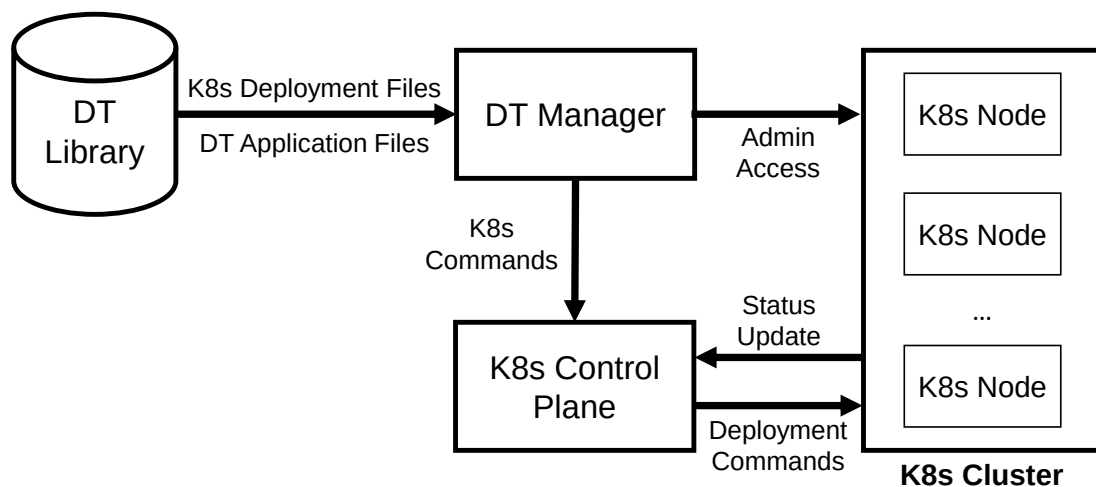


Figure 7.5 DT Manager connection to K8s cluster

DT library is a shared volume that holds a DT application's Docker image and its K8s scripts. DT manager creates a container registry containing all Docker images defined in the

DT library. The container registry holds all versions of DT applications. Additionally, the DT manager updates an existing DT application with a newer version when it is pushed into the container registry. K8s rolling update policy is used to keep old version DT applications available until the new version is ready. Operators can specify the DT application version in the K8s scripts. This approach helps operators to build and test new DT applications before deploying them on the DT framework.

7.4.2 Pub/Sub Communication Channel

The Pub/Sub communication channel is a shared component between both DS and DT. In coordination with the work published in AboElHassan and Yacout [2], Apache Kafka® is used as the backbone Pub/Sub communication channel. Kafka is a Pub/Sub messaging system that provides high-throughput, low-latency, and fault-tolerance capabilities for datastream processing. Kafka distributes the messages published over multiple servers called brokers. A replica of each message is stored at each Kafka broker through a leader-follower cooperation approach. Kafka’s capabilities are enabled by data replication, the single-writer principle, and distributed storage.

Dockerized version of Apache Kafka® is used for deploying Kafka on K8s. Each Kafka broker is deployed on a separate K8s node with a set of K8s headless services for communication. Kafka’s deployment on K8s enables the scalability of Kafka in runtime. The number of Kafka brokers can be scaled up by increasing the number of K8s replicas. A shared volume is used to write the Kafka message log. The shared volume allows any Kafka broker to restart from its current state.

Each Kafka message consists of a topic name and the message content. A Kafka message can be received by any DT component that subscribes to the corresponding topic. Kafka topic names take the form of *PublisherName_AttributeName*, where *PublisherName* is the name of the DT component publishing the message, and *AttributeName* is the physical/virtual attribute being published onto the Pub/Sub communication channel. Creation time was used in Kafka topic name in AboElHassan and Yacout [2]; however, automating the subscription step is easier using unique publisher names. Each published message holds a single attribute. If a DT component’s state contains multiple attributes, then a message is published for each attribute using its corresponding Kafka topic. The message per attribute design enables subscribers to subscribe only to the needed information.

Source gateway publishes the physical system information onto the Pub/Sub communication channel. The physical data are then processed by the subscribing DT components. DS models and DT applications share information through the Pub/Sub communication channel. Each

DS model publishes its state with a Kafka topic name, which contains the DS model name and the published variable. DT applications then subscribe to the needed Kafka topics, which are published by the source gateway or another DS model.

7.4.3 Digital Shadow

AboElHassan and Yacout [2] DS is used as a component in the DT framework. DS collects the physical system's state through source gateways, publishing the state on the Pub/Sub communication channel. DS models, deployed in DS pool, use the physical system's state for creating a virtual replica. DS sub-components details are described in a previous publication [2]. Figure 7.6 shows the architectural design of DS and its components.

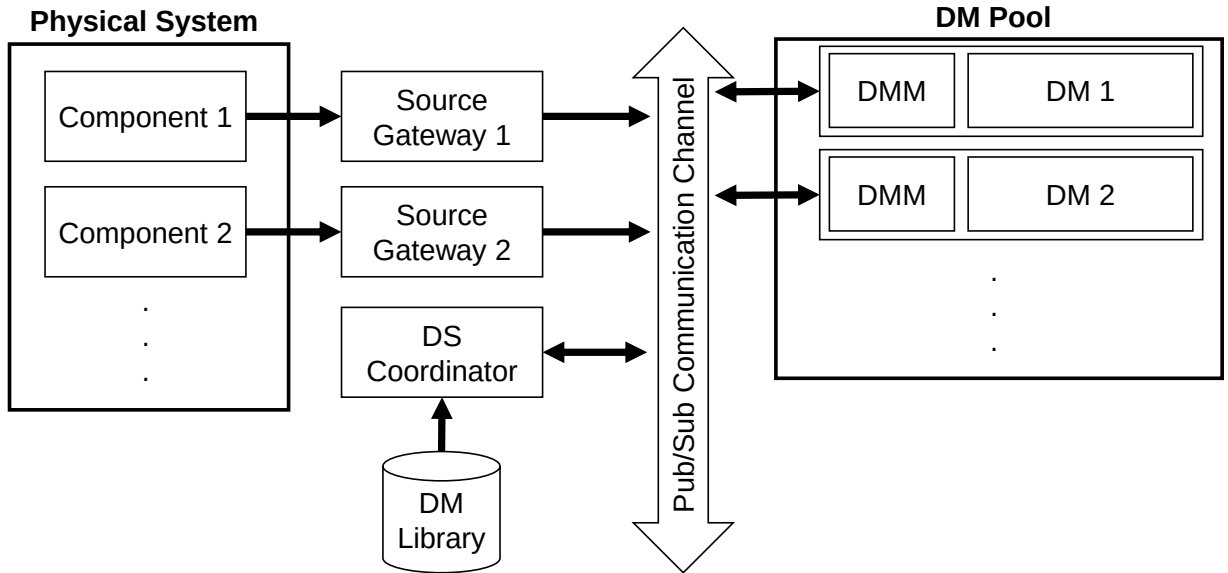


Figure 7.6 Digital Shadow Architecture [2]

The physical system's information is acquired by the source gateway. source gateway collects the physical system state according to the source gateway's proximity to the system. Source gateway could be deployed on edge devices inside the physical system as shown in Figure 7.7. A data aggregator API collects the information from physical system modules, then sends it to the DT system through a Kafka client connector. This enables the source gateway to listen directly to the physical system controller, reducing the communication overhead. However, this approach requires access authorization and knowledge of the physical system communication. Another approach is deploying the source gateway at a business-level server as shown in Figure 7.8. The source gateway collects the system's information by pulling the system's state through a secure network connection. This approach requires lower access

authorization since data is pulled from a web server. However, real-time data acquisition needs a high pulling rate, resulting in increased communication overhead.

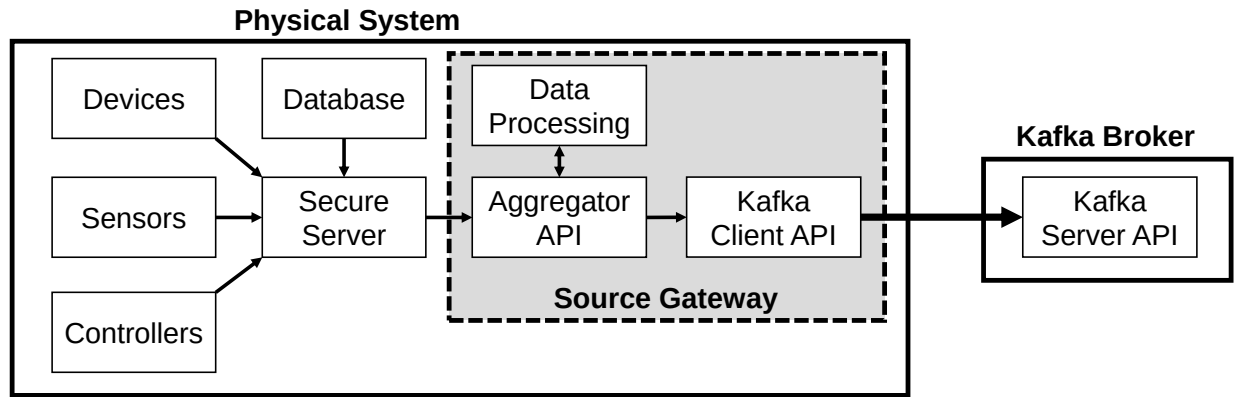


Figure 7.7 Tightly-Coupled Source Gateway

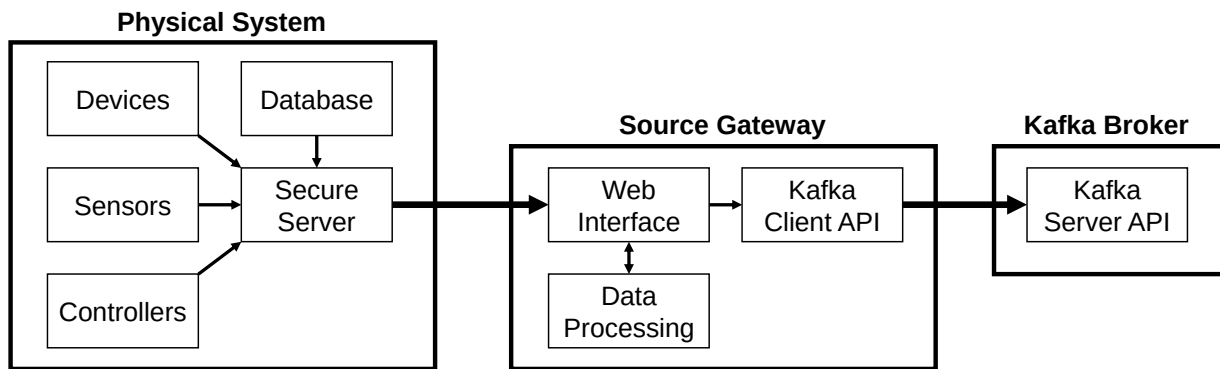


Figure 7.8 Loosely-Coupled Source Gateway

DS pool is a virtual namespace that contains the DS models. DM is a Docker container running a simulation model. DM can use different simulation techniques such as mathematical, statistical, behavioral, process-flow, state-flow, or data mapping. The latest physical system state is pulled from the Pub/Sub communication channel during DM Initialization. Then, DM shadows the physical system changes by listening to any update published by the source gateway. HTTP API is integrated into DMs to receive control messages and send status updates.

DT manager controls the DS sub-components through the DS coordinator. DS coordinator deploys new DMs to the DM pool and provides access to them. It also controls the start and stop of DM simulation, and pulls DM state upon request. DS coordinator interacts with any DM using its HTTP API.

7.4.4 DT Application

DT sandbox is a virtual workspace for DT applications. DT applications provide intelligent recommendations and insights for decision-makers. Any artificial or rule-based tool could be used as a DT application for real-time decision support. DT applications use different data analysis and prediction techniques such as simulation, optimization, and ML algorithms. An Operator sends a query request to a DT application. The DT application executes the application's logic using the current system's state. DT application results are then sent back to the operator.

DT application is composed of 2 components; Pub/Sub client and DM. Pub/Sub client connects the DT application with the rest of the DT through the Pub/Sub communication channel. DM contains the application logic which simulates the system behavior, interprets the current state, predicts future states, or recommends the next actions. DM is implemented by modularizing the DM logic into a Docker container with an HTTP API. The Docker image contains the DM logic and its dependencies. For example, an AnyLogic DM image would contain the AnyLogic model, Java runtime environment (JRE), and any other libraries needed to run the AnyLogic model. The HTTP API provides an interface to interact with the DM logic. Operators use the HTTP API to query the DM and retrieve its recommendations.

DT application are configured using K8s configuration script. This script defines the DT application name, the Docker image of the DM, the Pub/Sub subscription list, and the DM configuration. DT application name is a unique name assigned to the DT application. This name is used as the *PublisherName* if the DT application published its results on the Pub/Sub communication channel. The Pub/Sub subscription list is used by the Pub/Sub client to subscribe to the needed Kafka topics. Finally, the DM configuration contains any parameters defined for the DT application. These parameters define the DM logic execution mode. Additionally, they could be used to set the Pub/Sub pulling mode of the DT application, which is either *latest* or *earliest*.

Pub/Sub client provides access to the information published on the Pub/Sub communication channel. The Pub/Sub client implements a Kafka subscriber client script that pulls the needed Kafka topics according to the DT application's Pub/Sub subscription list and pulling mode. *latest* Pub/Sub pulling mode pulls the latest published messages as the most recent state of the system. While *earliest* Pub/Sub pulling mode pulls all published messages to track system changes. The Pub/Sub client passes the received information to the DM logic using HTTP requests.

In contrary to DS models in AboElHassan and Yacout [2], the Pub/Sub client does not

control the model flow of the DT application. DS models wait for events to occur in the real world in order to apply them in the virtual replica. While DT applications take the current system's state from the DS, then look ahead of the physical system in time. Therefore, a simple Kafka subscriber script is enough to provide the DT application with the needed information.

DT application interacts with the Pub/Sub client through an HTTP API. DT application listens for any operator request over a dedicated port. An operator's request includes performance inquiry, product status, or simulation request. Upon receiving an operator query, the DT application sends a pull request to the Pub/Sub client. The DT application then initializes the DM logic using the Pub/Sub client's response data. The DT application executes the DM and presents the DT results to the operator.

7.5 Designing Models for DT vs DS

A Model is an abstract description of the real system. Model designers develop models based on their understanding of the system and their expertise. Hence, the same system can be modeled in different approaches. Additionally, model designs depend on their objective. For example, a store can be modeled in order to optimize the employees' schedules. While the same store can be modeled to analyze and enforce social distancing between customers. In this section, multiple aspects that differentiate between DS models and DT applications are discussed. These aspects include the model objectives, time flow in the model, and handling stochastic events.

7.5.1 Digital Shadow Model

DS focuses on virtually replicating the real system behavior. DS models process real-time information to keep the virtual state of the system synchronized with the physical state. This requires DS models to keep a consistent description of the system a priority over generating insights.

Model Flow

DS models collect and process their physical counterparts' information over the physical system lifetime. A DS model needs to be initialized by the current physical system state before starting the simulation process. Afterward, the DS model updates its status belief of the physical system using sensor measurements received from the real world. This process of continuous update is the core concept in designing a DS model. Figure 7.9 shows the DS

model execution flow inside the DT framework. DMM controls the model by passing the sensor measurements and sending advance time commands to the DS model. The DS model waits until it receives a control command from the DMM then it updates the model state.

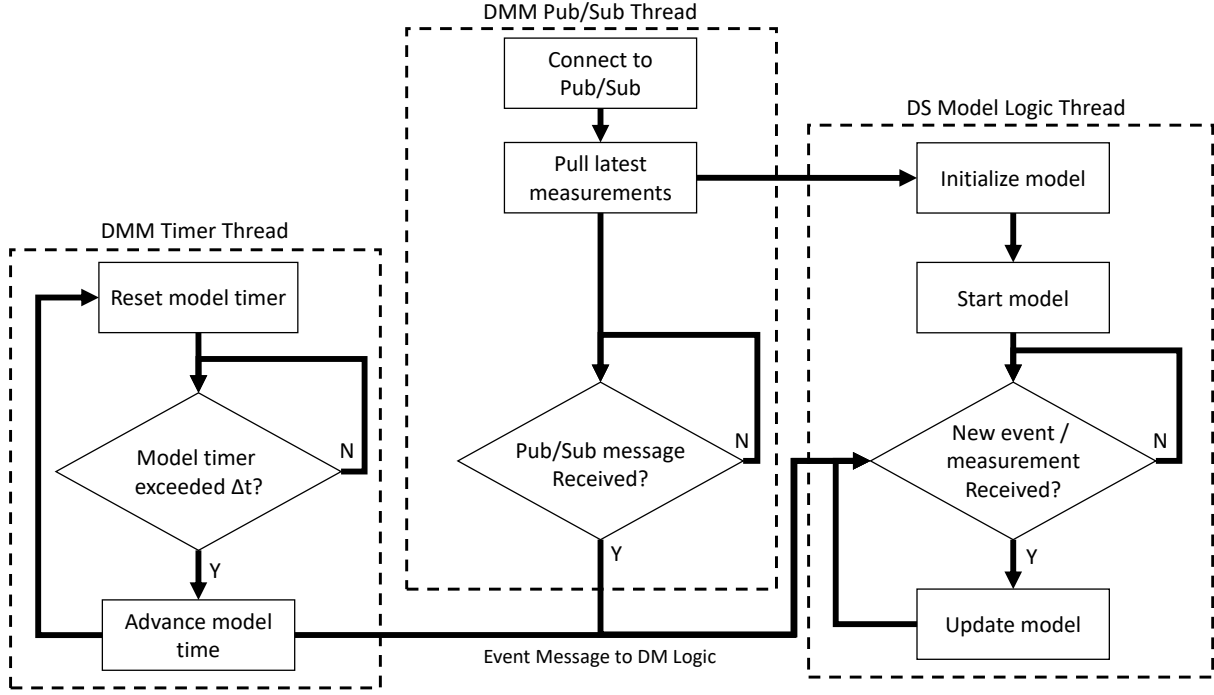


Figure 7.9 DS Model Execution Flow Chart

Due to the real-time requirement of DS, a DS model must have a short cycle time. DS model flow must not be blocked by delays or waiting for external inputs. However, DS fast response is subjective depending on the application criticality and domain. For example, a DS model of a drilling machine should have a cycle time in milliseconds, while a DS model of a water reservoir could have a cycle time in minutes.

Dynamic Behavior

Another core aspect of DS models is tracking the physical world at every step. This requires the DS model to be dynamic and adaptive in order to mimic changes in system behavior during runtime. For example, a collaborative robot DS model should reflect the change in robot speed and torque if the human operator entered a restricted zone. The DS model incorporates both normal and restricted modes of operations. Safety sensors trigger the restricted mode in the real robot to reduce the robot's speed and force. The triggering event must be modeled in the collaborative robot DS model to capture the behavior change.

Underlying physics must be considered for designing dynamic DS models. For example, the kinematics and dynamics of a robotic arm motion are not enough to simulate an emergency stop behavior. This means gravity and free motion need to be incorporated into a robotic arm DS model. Momentum and mechanical brakes are also modeled to accurately estimate the final position and orientation of the robotic arm. External factors and unmodeled scenarios should also be considered when designing a DS model. The DS model should adjust the model operation if the model's estimate differs from the real sensor measurements. For example, a DS model of an exoskeleton suit should be flexible enough to consider the operator's activities and the external load factor.

Modeling Stochasticity

Finally, stochastic properties of the real world must be handled within the DS model. Estimating probability distributions and periodic updates of the system status are needed to keep the DS model close to the real physical state. For example, a conveyor DS model should take into consideration the randomness due to friction and heat, alongside the change in load weight and its effect on the conveyor's speed. In such a scenario, it is better to periodically update the conveyor speed in the DS model by sensor readings. Another example is the quality of a production process, where unmodeled factors affect the quality of products. The DS model should estimate the quality level of the production process. However, the actual product's quality must be captured from the real quality test.

7.5.2 Digital Twin Application

DT focuses on generating data insights and providing decision support. Simulation models are used in DT applications for simulating future events in order to predict the effect of each decision made. Historical events are also analyzed and visualized to provide insights to decision-makers. Therefore, business requirements shape the design aspects of the DT applications and simulation models.

Model Flow

Similar to DS models, DT applications need to be initialized by the current system state before execution. However, the DT applications outpace the real world in order to predict future events. Figure 7.10 shows the DT simulation model execution flow inside the DT framework. DT application starts from the current state of the system, then DT continues the simulation run till it reaches a terminating condition. The simulation stops after a predefined

duration or when an event occurs. Stopping the DT simulation model after a specific time duration generates data insights about the system behavior during that interval. While event-based termination predicts the time and condition of such an event. For example, simulating a production line for a week estimates the throughput and the quality yield percentage. While simulating a production line, till a specific product is produced, estimates the delivery date of that product.

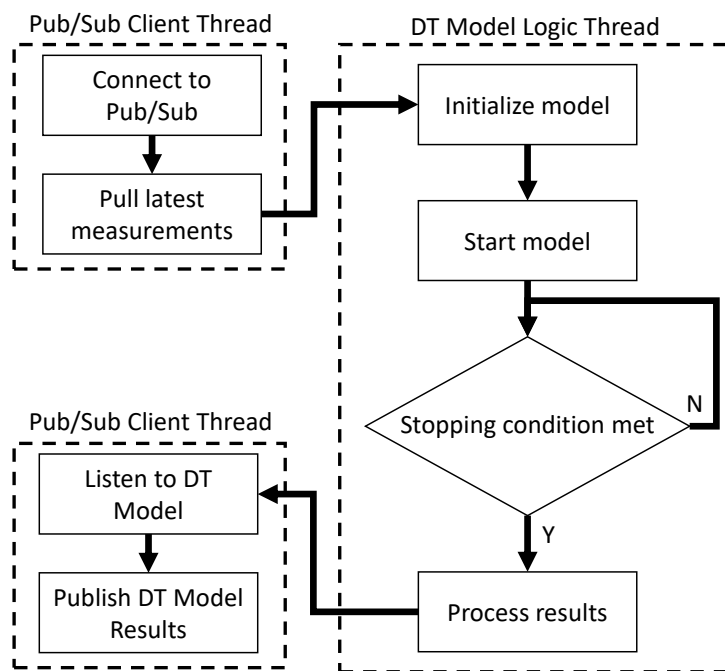


Figure 7.10 DT Simulation Model Execution Flow Chart

Modeling Stochasticity

Contrary to DS models, stochastic aspects of the system can be modeled in DT simulation models using random variables. The DT application still needs to consider stochastic behavior in the real system. Probability distributions can be used in representing random events during forecasting. For example, the downtime of a machine in a production line can be sampled from a probability distribution based on historical data. Randomly generated downtimes will result in a more realistic prediction of the system behavior.

7.6 Semiconductor Production System Use Case

We applied the proposed DT framework in Sect. 7.4 to a semiconductor production system use case. Integrated circuits (ICs) are manufactured in batches, where a single semiconductor wafer contains multiple ICs. Wafers are processed through a process of repeated steps called recipes. A fixed number of wafers of the same size and product type is called a lot. Wafers in a lot are processed and transferred together from one process step to another.

In addition to the process flow's repetitive steps, a lot or some wafers in a lot may repeat the same process steps due to reworking. Rework is performed due to production yield and timed recipes called timers. A timer is a set of recipes that must be completed within a specific time duration. A lot is scrapped or reworked if its timer exceeded the time limit.

IC fabrication areas are divided into several workstations, where each workstation contains equipment of the same type. Some equipment process multiple wafers of different product types simultaneously. While other equipment process only one wafer at a time. Equipment is reconfigured before performing a different recipe. Managing the production system is a difficult task due to the complex fabrication areas' structure and batch-based manufacturing [130].

7.6.1 Source Gateway

The production system use case utilizes identification labels to recognize lots at each workstation. Lot ID is scanned at loading and unloading operations. The scanning timestamp is recorded in a database to identify the start and end of a recipe step. These timestamps are called *track-in* and *track-out* records. Additionally, an *order-release* information is recorded in the database when a job is dispatched.

A loosely-coupled source gateway is designed to pull the production system events periodically from the database. *Track-in*, *track-out*, and *order-release* events pulled and pre-processed before being pushed into the DT system. When a new record is pulled from the database, the record information is transformed into a Kafka message based on the event type. Kafka messages consist of message content and a key identifier called Kafka topic as presented in Sect. 7.4.2. DT components subscribe to the needed information using their Kafka topics. Kafka topic name contains the event type, which is either *track-in*, *track-out*, or *order-release*. Kafka topics take the following naming convention:

- *PublisherName* is the source gateway name
- *AttributeName* is the physical attribute name

Each Kafka message contains the event log in JSON format. *Track-in* and *track-out* messages contain lot ID, product type, equipment ID, and event timestamp. While *order-release* messages contain lot ID, product type, number of lots per order, event timestamp, and deadline date. These messages are used to track the status of each product in the production system.

7.6.2 Digital Shadow

DS models aggregate information received from the source gateway to generate a virtual image of the system. These models are used to provide historical and real-time states to DT applications. Two DMs were developed to represent the production system and the products inside. The first DM traces the product through its lifetime in production. The second DM simulates recipe production times and behavior in the production system.

Product DM

Tracking the product status over its lifetime is important to analyze the production system's performance. Ontology provides a conceptual description of the system, formalizing prior knowledge about the system's entities and their relationships. Ontology graphs queries provide knowledge inference and data consistency. An ontology graph is designed from scratch to represent the authors' understanding of the semiconductor production system and the relationship between its entities. Product DM collects products' information from the source gateway. Products' information is then used to update the products' status and their relation to equipment and recipe steps.

Figure 7.11 shows the ontology semantic graph of the production system and the product's meta-data. The production system parts are defined as entities, such as product type, equipment, process, and recipe. Product type entities are unique identifiers for the different products manufactured in the fabrication facility. While equipment entities define the manufacturing resources available in the different fabrication areas. Process entities define the process flow of the manufacturing process through a set of self-ordering relationships. Finally, recipe entities describe the process step details, including the recipe time duration, equipment used, and if activates/deactivates a timer.

Products are defined as lot entities. Lots are part of a job entity, where a job includes one or more lots. Job and lot entities are created dynamically inside the DM when an *order-release* event message is received. The job entity contains a job ID for identification and has a *type* relationship with one of the pre-defined product type entities. Each lot entity has its unique

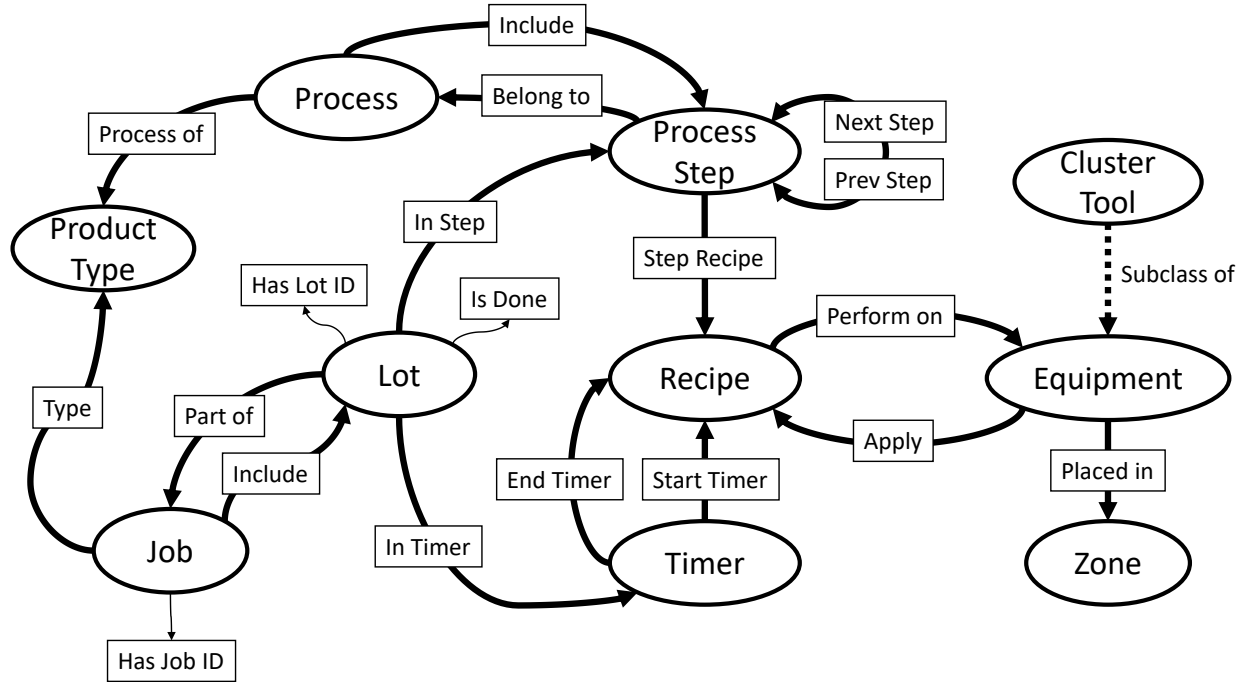


Figure 7.11 Product DM Ontology Graph

lot ID and has a *part of* relationship with its job entity. When a *track-in* event message is received, the corresponding lot entity is updated to start the process step operation. The event time is captured as the starting time of the process step, which is used to check the timer status if the lot is in a timer. When a *track-out* event message is received, the corresponding lot entity is updated to move the lot from the current process step's location to the next process step's location.

Web Ontology Language (OWL) is used in implementing the product DM. *OWLready2* is a Python library that generates and updates OWL graphs. The Python-based DM receives all messages generated from the source gateway. It updates the status of lot entities according to the event log received. The ontology graph is exported and downloaded as an RDF/XML file. Additionally, the product DM published the information of the lot entity upon any change in its status.

Production System DM

The product DM describes the manufacturing process from the product's perspective. Data-driven and Agent-based model (ABM) techniques are used in designing the production system's DES model on AnyLogic™. Data-driven modeling technique enables the design of

reconfigurable models. Process flow and entities in data-driven models are created according to the information stored in a database or a file system. This eases the process of reconfiguration since changing the process flow details in the database will change the model behavior in consequence. ABM technique enables encapsulation and reusability. In ABM, each entity type in the system is modeled as an agent, where each agent has a process flow or a state chart behavior. Agents are used in encapsulating the entity's logic. Additionally, an agent could be used as a module inside other agents for reusability.

The production system DM focuses on the fabrication facility resources and their allocation. Equipment is defined as an agent as shown in Figure 7.12. Equipment agent models the equipment's operation as a single server process. Lots wait until the equipment is ready to process a new lot. Then a lot is selected from the waiting area and moved to the processing area, where the lot is held until processing is done. Upon process completion, the lot checks if the current process step is the last step or not. The lot is removed from the fabrication area at the end of its process flow.

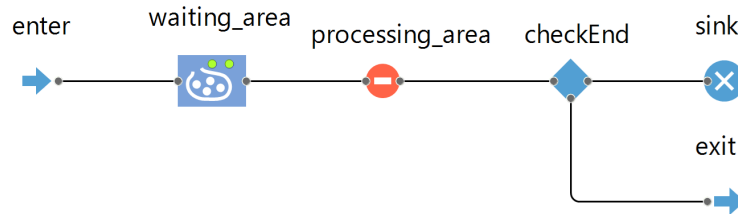


Figure 7.12 Equipment Agent Process in Production System DM

The rest of the production system is described as data agents, where the information about the agent status is encapsulated. A Lot is defined as an agent that holds all the needed information regarding its type and current progress, as shown in Figure 7.13 and Table 7.1. Each lot agent has a timer agent as a part of it. A timer agent captures the start and end of a recipe timer. If a timer exceeds the time limit, an event is triggered that highlight the corresponding lot for scrap or rework, as shown in Figure 7.14 and Table 7.2. Finally, a recipe agent describes the recipe in terms of the product type and its order in the product's process flow. the recipe agent also indicates which equipment is performing this recipe, as shown in Figure 7.15 and Table 7.3.

The production system DM subscribes to *track-in* and *track-out* event messages from the source gateway. When a *track-in* message is received, the corresponding lot is moved from the waiting queue to the equipment's processing area. While the lot is removed from the operation area when a *track-out* message indicates the end of recipe processing. After ending a process step, the lot is moved to the equipment performing the next process step. Lots

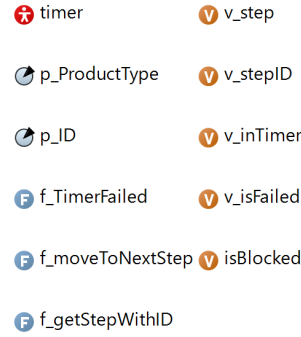


Figure 7.13 Lot Agent Description in Production System DM

Table 7.1 Lot Agent Elements Description

Agent Element	Description
timer	An instance of the timer agent
p_ProductType	The product type of wafers in the lot
p_ID	The lot identification number
v_step	The current recipe step
v_stepID	The current recipe step number
v_inTimer	A boolean indicates if the lot is within a timer
v_isFailed	A boolean indicates if the lot had failed its timer
isBlocked	A boolean indicates if the lot is blocked at the current step
f_TimerFailed	A function updating the lots info if the lot failed its timer
f_moveToNextStep	A function moving the lot to the next step
f_getStepWithID	A function returning the recipe step by its ID

move from one workstation to another based on their product type and current production progress. Figure 7.16 shows an example screenshot of WIP lots waiting at a workstation.

As previously discussed in Sect. 7.5, the production system DM is initialized at startup with the current system's state. This includes the current equipment recipe setup, equipment availability, and WIP lots inside the system. Equipment information is pulled from the fabrication facility database. However, a copy of the equipment information is stored in a local database in this study. WIP lots' information is pulled from the source gateway through the Pub/Sub communication channel.

7.6.3 Digital Twin

DT models use the information processed by DS to generate insights for decision-makers. Three DT applications are developed while adapting the human-in-the-loop paradigm. The



Figure 7.14 Timer Agent Description in Production System DM

Table 7.2 Timer Agent Elements Description

Agent Element	Description
p_Lot	The lot which this timer is associated
p_Duration	The timer duration limit
f_startTimer	A function starting the timer counter
f_stopTimer	A function stopping the timer counter
ExceedTimer	An event triggered if the timer was not stopped before its duration

first DT application keeps operators notified of the system status. The second DT application utilizes RL to recommend the next lot to process for each equipment. Finally, the third DT application utilizes DES models to estimate the production time of a job.

Human-DT Interface

Interpreting the DT recommendations is important in involving decision-makers. A human-DT interface provides an interactive medium for operators to interact with the DT system. Web-based human-DT interface is developed to visualize the virtual replica of the production

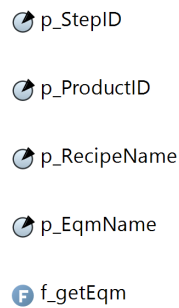


Figure 7.15 Recipe Agent Description in Production System DM

Table 7.3 Recipe Agent Elements Description

Agent Element	Description
p_StepID	Recipe step identification
p_ProductID	Product type applying the recipe step
p_RecipeName	Recipe step name
p_EqmName	Name of equipment performing the recipe step
f_getEqm	A function returning the equipment performing the recipe step

system and notify operators of any modifications to the system.

A node-red dashboard is developed to visualize the production system information as an interactive Business intelligence (BI) dashboard. The BI dashboard presents recipes' details, equipment setup, and different product types available in the system. This information could be pulled from the fabrication facility database. However, a copy of the equipment information is stored in a local database in this study similar to the production system DM. Lot progress is also visualized by the BI dashboard. WIP lot information and state are accessed by selecting the lot in question from the list of WIP lots. Figure 7.17 shows WIP lot info displayed on the BI dashboard. The operator accesses the lot's current state such as the in-progress recipe, the workstation applying this recipe, and the dispatching time to that workstation.

Dispatching Recommendation DM

In the semiconductor industry, different heuristic rules are combined when dispatching a lot to equipment [130]. However, It is hard to optimize heuristic rules for complex and dynamic systems such as semiconductor production systems. RL is a branch of ML domain that studies the use of intelligent agents to take actions that maximizes the returned reward. RL agents learn policies to select the best action based on the current state of the environment. The use of RL in the use case production systems has been studied in a previous publication [130].

An RL agent is created for each workstation in the production system. The RL agents are trained to cooperate in order to improve the overall system throughput and avoid bottlenecks. Deep-Q network (DQN) algorithm is implemented in Python using TensorFlow. The DQN parameters and state-action-reward definition is described in details in Sakr et al. [130]. The trained RL agents are exported into TensorFlow models, which are deployed as a part of the dispatching DT application. Figure 7.18 shows the DT logic flowchart when an operator runs the dispatching recommendation DT. The RL agent corresponding to the workstation is called to rank the WIP lots using the current state of the production system. The RL

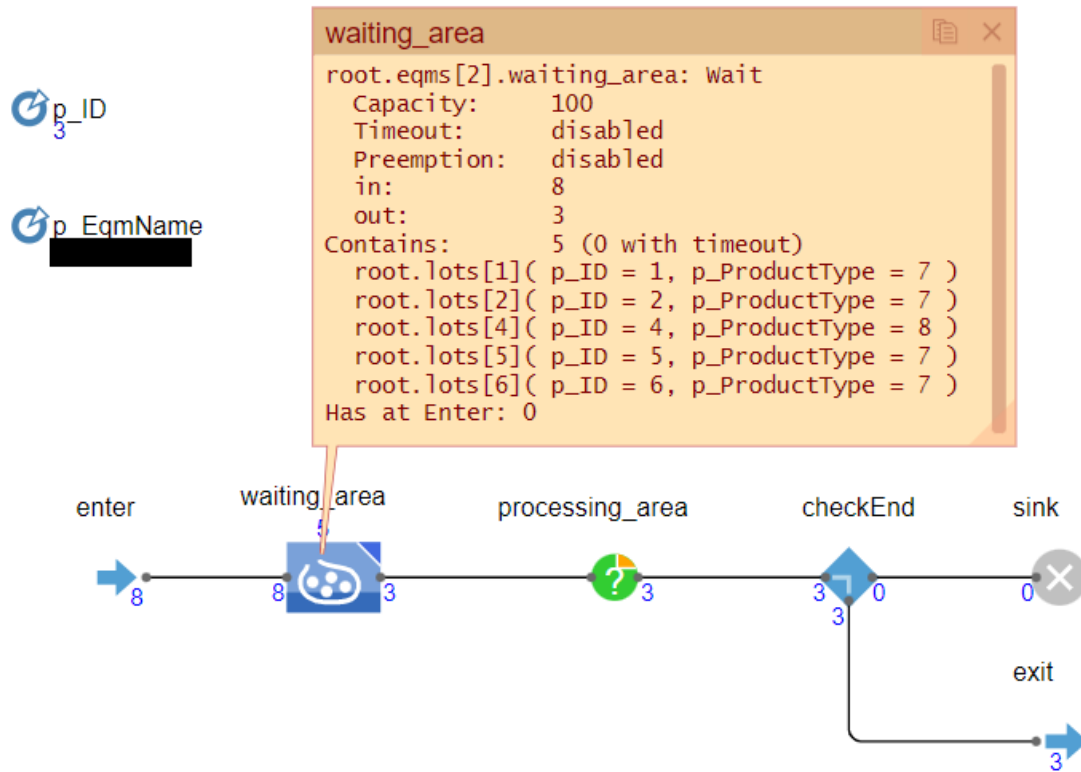


Figure 7.16 Screenshot of an Equipment State in Production System DM

agent takes into account the availability of the human operator and the possibility of their interruption during the loading/unloading operation.

The dispatching DM collects the needed lots' information from product DM. While the equipment states are collected from the production system DM. Both DS models form a comprehensive description of the system state. The dispatching DM then gets the operator's inquiry through a web-based interface. Operators ask for the RL recommendation using computer terminals located at each workstation. Figure 7.19 shows an example screenshot of the web interface used by the operator. The dispatching DM ranks the waiting lots according to the estimated reward. The ranked lots are displayed back to the operator with their slack ratio and timer status. Displayed information helps the operator in interpreting the RL recommendation. The operator selects the RL recommended lot or a different lot based on their verdict. This approach enables the supervision of human operators over the DT decisions.

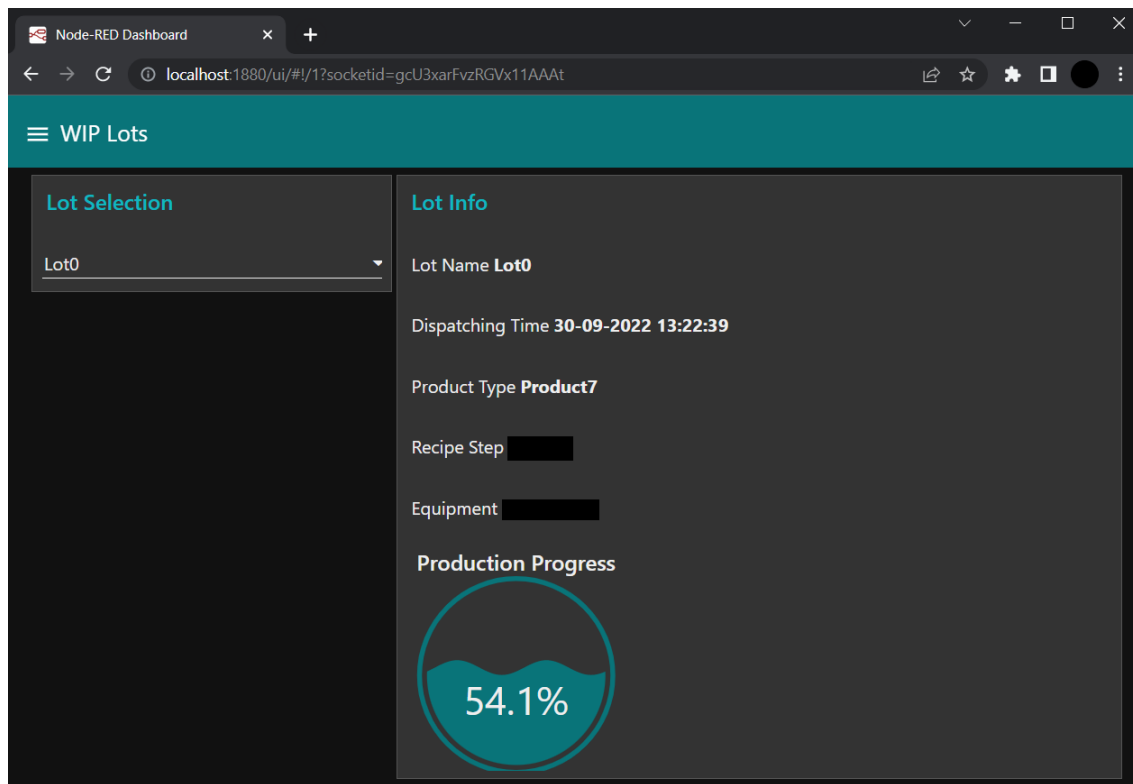


Figure 7.17 Example of an WIP Lot Information Displayed using Node-Red BI dashboard

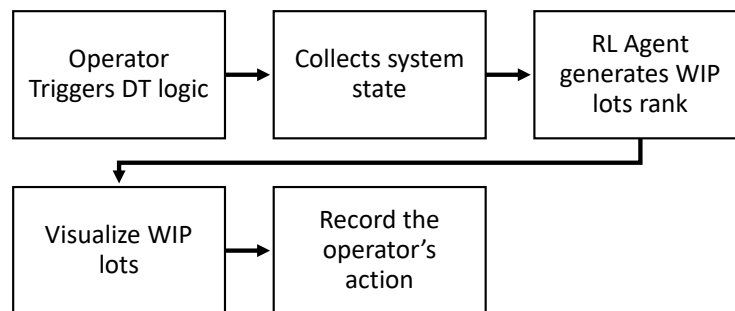


Figure 7.18 Dispatching Recommendation DT Logic Flowchart

Lot Dispatching Recommender

Select Station:

Run Recommender

Rank	Job ID	Lot ID	Product Type	Slack Ratio	Timer Status	
1	35	70	Product7	42.8%	In Timer	<button>Select Lot</button>
2	30	62	Product7	62.3%	Not in Timer	<button>Select Lot</button>
3	36	73	Product8	20.1%	Not in Timer	<button>Select Lot</button>

Figure 7.19 A Screenshot for Dispatching Recommendation DM Web-Based Interface

Deadline Estimation DM

Deadline estimation is important for managing clients' expectations. Statistical approaches are usually used in deadline estimation by analyzing historical data. However, statistical approaches do not consider the production system state. DT provides a realistic deadline estimation using VC. VC uses digital models as a virtual environment for commissioning and testing control systems before deployment [131]. In this paper, the VC concept is used with a broader view, where VC uses a simulation environment to commission system changes such as production plans, process-flow changes, machine configuration, and new products. The deadline DM simulates the production system from its current state. The DT simulation considers the system behavior, stochastic parameters, and the effect of the initial state.

As discussed previously in Sect. 7.5, a DS model and a DT model could be describing the same system with the same behavior. However, both models are implemented differently due to objective differences intended for each model. The deadline DM models the production system using DES approach on AnyLogic™, similar to the production system DM. The production system DM runs at real-time speed and waits for source gateway event messages. While the deadline DM runs at virtual speed to simulate future events. Additionally, a single deadline DM run is repeated multiple times to capture the effect of system stochasticity on the simulation results.

Data agents implemented for the production system DM are reused in the deadline DM; however, the equipment agent is implemented differently. Since DT models do not pause between time steps, the equipment agent uses a *delay* module instead of a *hold* module as shown in Figure 7.20. Processing time for each recipe is modeled using random distribution extracted from historical data. The equipment agent delays the in-process lot according to the equipment setup and the lot's product type. Another key difference between the production system DM and deadline DM is the modeling of lot dispatching. In the production system DM, lot dispatching is captured by the messages received from the physical system. While the deadline DM uses RL agents in lot dispatching. The equipment agent calls the corresponding RL agent for selecting the next lot to process. The RL agent returns the ranked lots, where the equipment agent selects the highest rank lot.

The deadline DM subscribes to *track-in* and *track-out* event messages from the source gateway. At the start of the simulation, the deadline DM pulls the current WIP lots' information during initialization. Afterward, the deadline DM resets the Pub/Sub messages subscription. This enables the deadline DM to pull the latest system information for each VC request.

Finally, the deadline DM results are visualized through the AnyLogic™ simulation view. The

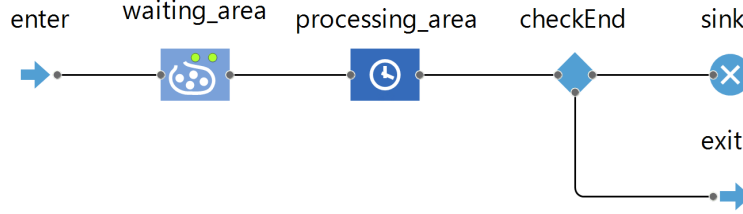


Figure 7.20 Equipment Agent Process in Deadline Estimation DM

operator selects the product type to be tested as shown in Figure 7.21. The AnyLogic™ simulation view allows the operator to inspect the model operation as in Figure 7.22. The operator inspects the VC test results through multiple graphs in the AnyLogic™ simulation view. Histogram plots are used to visualize the estimated processing time of the VC lot. Additional information is presented to the decision-maker such as the average utilization of equipment and the average queue length at each workstation.

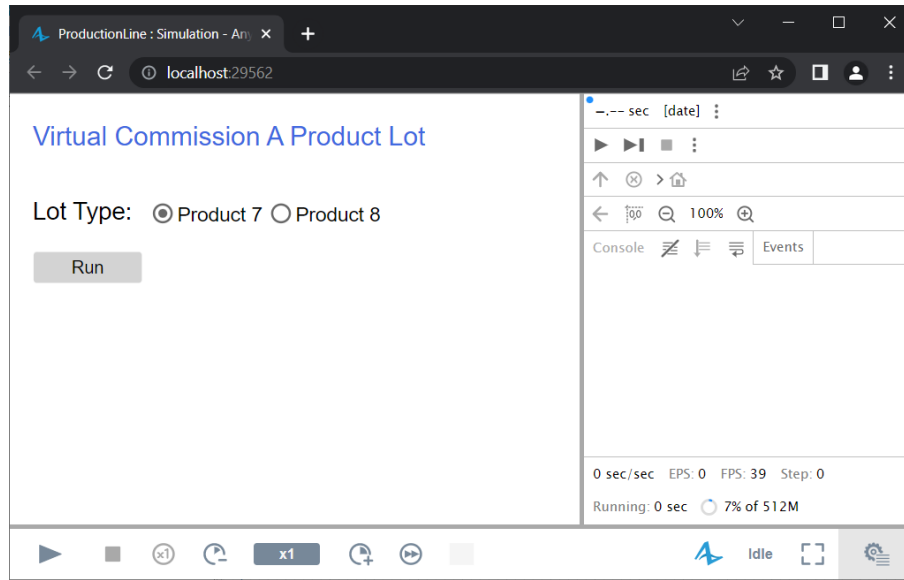


Figure 7.21 Screenshot of the start screen of Deadline Estimation DM

7.7 Conclusion and Outlook

In this paper, a general-purpose DT framework is proposed. The DT framework architecture is implemented using distributed system concepts such as modularization, event-driven architecture, and microservices. DS is used as a core component of DT. DS collects the physical system's information using a source gateway module. Then DS creates a virtual replica of

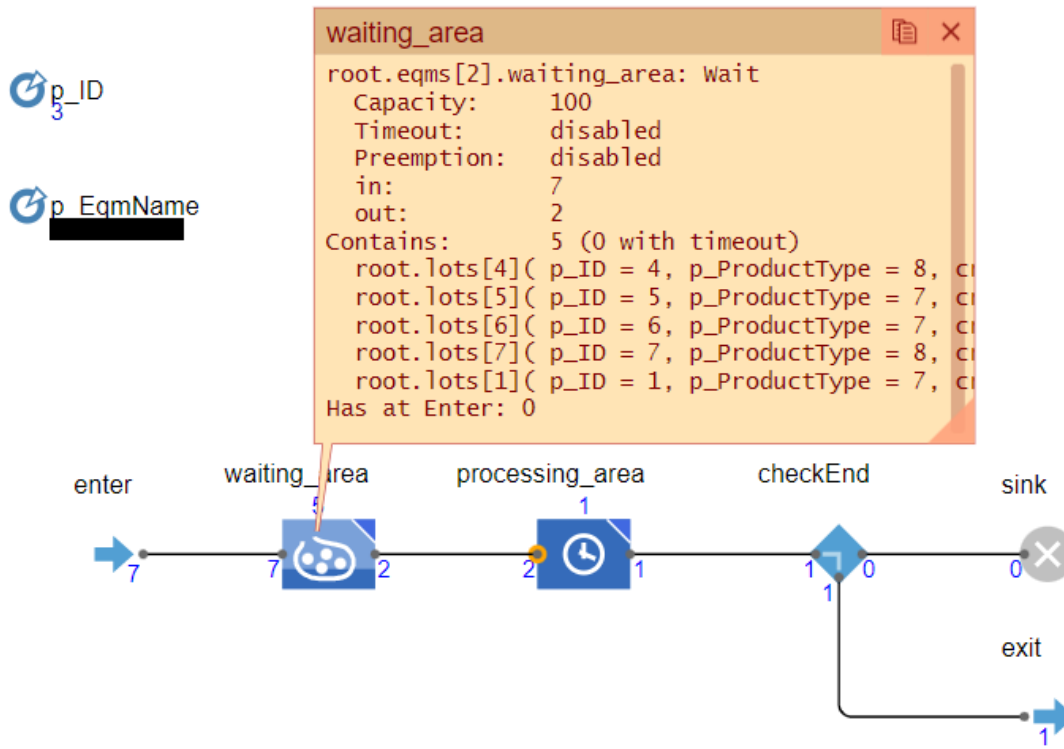


Figure 7.22 Screenshot of an Equipment State in Deadline Estimation DM

the physical system using multiple DS models. These models simulate the system behavior from different perspectives on different business levels. DT applications pull the needed information from DS for data analysis and insight generation. DT provides the analysis results and the recommended actions to human operators for decision-making.

Distributed system concepts enable the generality of the proposed DT framework and answer the first research question *RQ1*. Modularization of DT components enables the DT framework to deploy different types of DS models and DT applications. Docker containers are used to containerize DT components, while K8s is used to deploy and orchestrate the DT components. Kafka Pub/Sub communication channel is used as an information sharing platform between all DT components. Finally, microservices provide a flexible interface to integrate DS models and DT applications into the DT framework.

The DT framework generality is enabled through the separation between modeling the system and generating insights. DS provides a real-time digital image of the physical system using multiple models forming a multi-scale multi-physics description. DS models shadow the system's state and track any changes in its behavior. This requires the DS model to run in real-time and listen for any messages sent from the source gateway. On the other hand,

DT generates data insights and recommends decisions. DT applications look ahead into the future to predict the best course of action. This requires the DT application to run faster than the real-time rate. The use of objective-driven differentiation between DS and DT answers the second research question *RQ2*.

The proposed DT framework is applied in a semiconductor use case. An event-driven source gateway is implemented to capture the change in the system state. Two DS models are developed to model the production system and the product's flow. An ontology-based model is used to track the products' state through their production lifetime. While a DES model describes the production system's behavior and the WIP products moving between workstations. Three DT applications are developed to provide operators with recommendations. A BI dashboard is designed as a human-DT interface. While an RL-based application ranks products for dispatching based on the current state of the workstations. Finally, a product completion estimator is developed using DES simulation.

7.8 Future Work

This paper builds upon previous work in order to design a flexible general-purpose DT. Further study of DT components connectivity and adapting industrial standards is planned. Additionally, DT designs will be improved by analyzing DT's performance during anomalies such as network failures and missing data. Standardizing DT system design enables researchers to focus on DT applications regardless of the hosting framework. While applying DT in erroneous environments will improve current DT designs.

7.9 Author's Contribution

A. AboElHassan conducted the conceptual design of the DT framework and the distinction between DS models and DT applications, developed the DT framework and its components, deployed and tested the DT framework, and wrote the manuscript. A. Sakr developed the API access to the semiconductor system, contributed to the simulation models development, and trained RL agents used in the dispatching recommendation DM. S. Yacout provided guidance for the conceptual design and methodology. All authors reviewed the final manuscript.

**CHAPTER 8 ARTICLE 5: A DIGITAL TWIN DEVELOPMENT
METHODOLOGY FOR MANUFACTURING SYSTEMS**

Ayman AboElHassan, Ahmed Sakr, Soumaya Yacout

Submitted to:

Production Engineering

Date of submission: 02-Mar 2023

8.1 Abstract

Digital twin (DT) is used as a key enabling technology for manufacturing systems to capture the fast-paced changes in production systems. Disruptions in supply chains and the shift toward customer-centric production add pressure on manufacturers to adopt fast-response decision-making systems. DT systems utilize simulation models, artificial intelligence, information and communication technologies to provide real-time decision support. The process of designing and developing DTs is yet to be standardized. In this paper, a methodology is proposed to provide an iterative development process for DTs. The proposed methodology describes the steps of identifying the DT scope and goals, building a conceptual design of DT models, and implementing and validating the DT system. The proposed methodology takes into consideration the communication infrastructure required to transform a physical system into a cyber-physical system. The DT development process includes developing digital models, converting digital models into digital shadows, and adding decision-support intelligence. A warehouse testbed is used to validate the proposed methodology. A research laboratory is built as a cyber-physical system for warehouse operations. Virtual commissioning and operations management are used as objectives for the DT system. The proposed methodology is applied to develop the DT system for scheduling, dispatching, and quality inspection. The testbed experiment proves the applicability of the proposed methodology to develop DTs for warehouse systems.

Keywords

Manufacturing, Warehouse, Digital Twin, Design Methodology, Virtual Commissioning, Scheduling

8.2 Introduction

Customer-centric manufacturing and short design-to-production time push current manufacturing systems to their limits [132–134]. Supply-chain disruptions and market uncertainties increase the need for fast adaptive systems [132, 135]. Digital twin (DT) provides a solution for quick and smart response to changes in a manufacturing system [136–138]. Potential benefits of DT in the industrial domain encourage researchers to explore its usage in different applications. The number of publications in this field is increasing exponentially [13, 48].

DT provides a dynamic platform, creating a virtual image of a physical system [14, 15]. NASA first defined DT as "*a real-time multi-disciplinary multi-scale simulation replica of a*

real system" [17]. Real-time sensory data is collected from the physical system. Simulation and mathematical models then create a virtual replica of the physical system, capturing the system's behavior. This enables DT to support decision-makers by analyzing collected real-time data, recognizing behavioral patterns, and simulating decision consequences. DT is applied in multiple industrial domains. Its applications vary from virtual commissioning (VC) and deadline estimation to inventory management and production plan scheduling [8,14,139].

A CPS infrastructure is required for machine connectivity in a DT [24,69,140]. Figure 8.1 shows the different components of a DT and the CPS communication infrastructure required to connect these components. Physical machines are connected using industrial network protocols. CPS integrates IT and OT to pass information between the physical and virtual domains. DMs are the core components of a DT [2,8,14,15]. DM is a digital representation of the physical system. A DT may have multiple DMs running concurrently and sharing information. Contrary to a traditional simulation approach, DT simulates the physical system behavior taking into consideration real-time OT data collected.

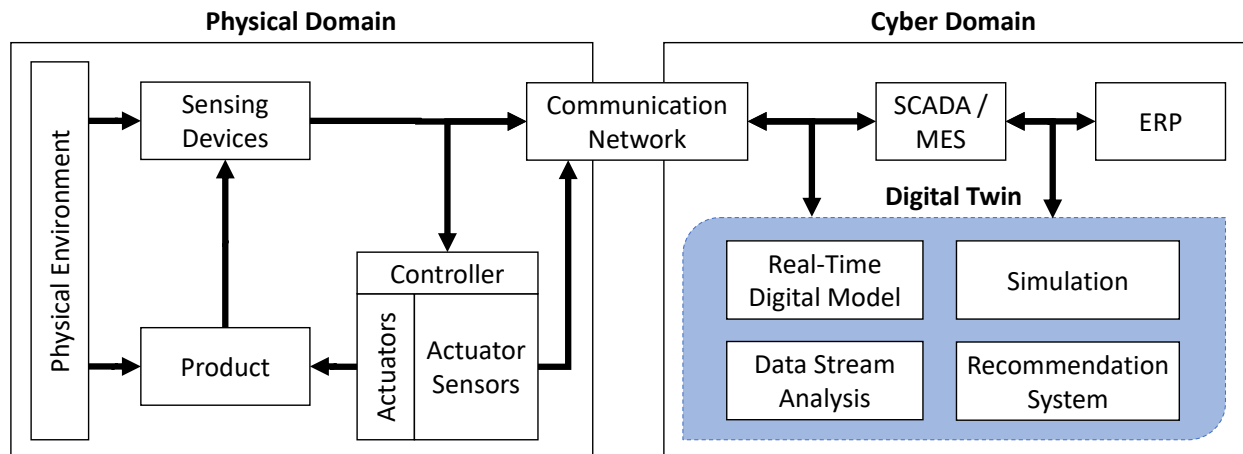


Figure 8.1 Digital twin as a part of cyber-physical system

DTs are often categorized according to their communication dataflow [8,14]. Traditional simulation approach utilizes offline DM. Physical system historical data is collected for simulating the system behavior. DT has an automated bidirectional communication channel with the physical system. It receives real-time data from the physical system so that decisions can be directly applied by on-site controllers. DS is a by-product concept of DT [2]. It receives the physical system data automatically but does not send feedback automatically. As such, DS does not affect the physical system directly. Figure 8.2 shows the communication dataflow directions between the physical system and its digital representation in the case of DM, DS, and DT.

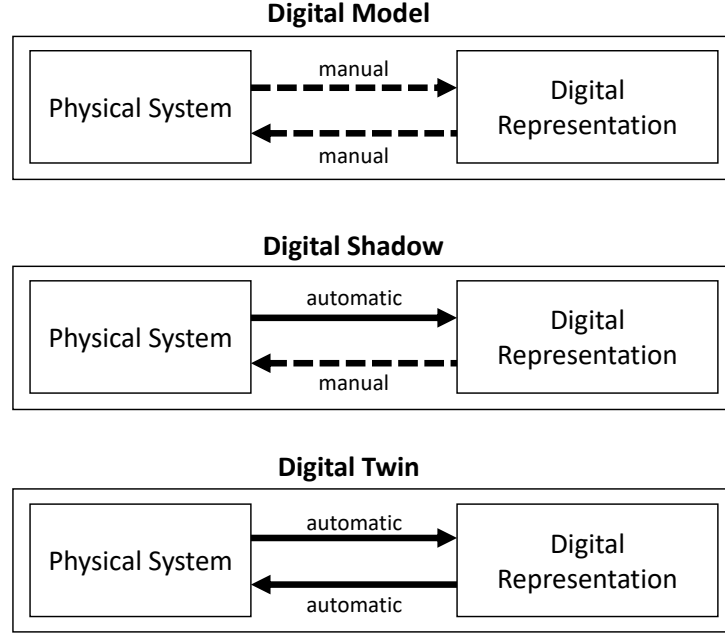


Figure 8.2 Dataflow in DM, DS, and DT

Real-time capabilities of DT emphasize its benefits in smart manufacturing systems [71]. DT's ability to simulate a system using live data generates real-time predictions of the physical system's behavior. Decision makers use DT for testing different production plans before deploying them on the real system [141]. DT enables fast response to sudden and unexpected system behaviors or changes. Data stream analysis continuously identifies behavioral inconsistencies and anomalies [8]. DT's access to the physical system's controllers enables fast response to production changes [8, 142].

8.2.1 Designing digital twin systems

The process of developing DTs has been discussed in the literature. Lu and Xu [86] adapted simulation model designing methodology in designing a DT development process. Phua et al. [34] proposed a stepwise process, where DT models are designed first, then connected to the physical system, and finally ML-based intelligence is integrated into DT. Reed et al. [124] proposed a DT simulation model development process, where the DT model is optimized according to the validity of the simulation results. Barbieri et al. [141] emphasized the importance of the planning phase in DT development, where the DT designers decide the technologies used, modeling tool, DT solution, and CPS interface. Moyne et al. [125] discussed DT requirements according to the DT definition and applications. Göppert et al. [58] proposed the use of ontology meta-models to describe the physical system components

and their corresponding virtual models. DT applications are then deployed using these ontology-based data models. Moyne et al. [125] and Qamsane et al. [126] proposed the use of a DT quality inspection to assess the DT's performance, which is then used in tuning the DT simulation parameters or notifying the DT designers. A summarized description of the mentioned approaches is shown in Figure 8.3.

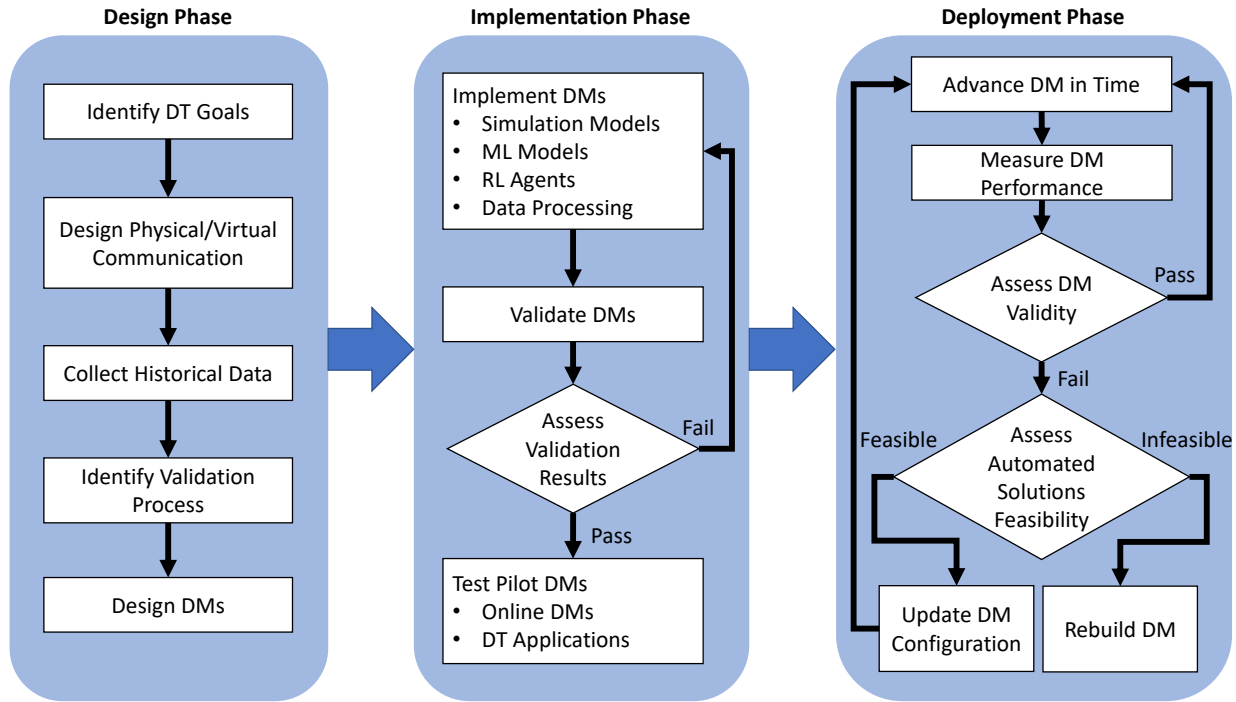


Figure 8.3 A summary of a collection of digital twin design methodologies in literature

8.2.2 Virtual commissioning and operation management using digital twins

A simulation model of the production system provides a virtual testbed to optimize the system's performance and explore different business strategies. virtual commissioning (VC) uses simulation models to test these strategies before deploying them in the real world [131, 141, 143]. As such, DT simulation models work as VC applications to predict future events based on DT live data. DT improves operation management for stochastic processes through the real-time update of the system's status. Barbieri et al. [141] adopted VC-based methodology for designing a DT for shopfloor scheduling. Wang et al. [135] used a shopfloor simulation model for real-time job rescheduling in a DT. Chakroun et al. [144] developed a CPS-based simulation model to optimize the production plan of an assembly workshop. Lai et al. [133] and Kumbhar et al. [137] proposed the use of DT for bottleneck detection. Machines' status and products' waiting time are collected by the DT from the physical sys-

tem. Then data-driven simulation models are used to study bottlenecks in different common production-line layouts. Kuhn et al. [143] proposed the use of error modeling in VC models to reduce the gap between reality and simulation in DTs. In this paper, the VC concept is used with a broader view, where VC uses a simulation environment to commission system changes such as production plans, process-flow changes, machine configuration, and new products.

8.2.3 Research objectives and contribution

The standardization of DT concept is still an ongoing process. DTs enable decision-makers to generate and test their systems virtually before deploying actions in the physical world. Developing the DT architecture during the planning phase of the CPS facilitates the assessment of possible problems. In this paper, a methodology is proposed to develop and deploy DTs in production systems. DS is used as a stepping stone in the process of designing and implementing a DT. The required CPS infrastructure is also addressed. A warehouse testbed system is used to validate the proposed methodology. The proposed methodology is used to build a DT for VC and operations management applications. A DT architecture is developed to schedule the warehouse operations and assign physical resources. The DT updates the operations schedule according to real-time product quality inspection. The paper's contributions are summarized as follows:

- Propose an iterative methodology for DT development based on agile development approach;
- Provide an example of CPS communication infrastructure that is required to enable DT integration;
- Build a CPS warehouse testbed system;
- Validate the proposed DT development methodology using the CPS warehouse testbed.

The rest of the paper is organized as follows; The methodology used to develop DT is presented in Sect.8.3. The warehouse testbed system and the DT development are described in Sect.8.4. The developed DT's characteristics are discussed in Sect.8.5. Finally, the paper's conclusion and future work are presented in Sect.8.6 and Sect.8.7.

8.3 Methodology

Linear DT development approaches were studied in the literature, as shown in Figure 8.3. These approaches define DT development as a model design process. However, linear model

development is not practical for large and complex production systems. Additionally, holistic collaborative models are hard to build using linear approaches. In this paper, the agile development approach is adopted to devise an iterative DT development methodology. The iterative approach enables designers to continuously improve models' performance. Its fast delivery approach keeps DT models' objectives concise and provides stakeholders' feedback to DT designers. The proposed methodology eases the development of holistic models through the definition of models' objectives in the planning phase, then enables concurrent development and deployment of digital models.

The DT development stages shown in Figure 8.3 do not take into consideration the DT enabling technologies during the planning phase. Data collection and aggregation are planned; however, the communication system's requirements to pass information from the physical system to the DT are not elaborated. In this paper, the CPS communication network is defined as a key enabling technology for DT integration. Industrial Ethernet (IE) is used to connect the physical and cyber domains in the CPS system. DT requires low-latency communication infrastructure, which is provided by some IE technologies. Migrating physical systems into CPS systems is discussed. The CPS system's network is assessed to ensure real-time communication for DT integration.

The proposed methodology divides the DT development process into 5 phases, as shown in Figure 8.4. The methodology first defines the DT role and transforms the physical system into a CPS. Then a DS architecture implementation of the DT is deployed and validated. Finally, the DS system is transformed into a DT by integrating DT applications. Utilizing DS as the core building block enables DT designers to generalize the DT and its purposes. Deploying the DS system validates the real-time simulation models before applying the DT decision-making process.

8.3.1 Planning Phase

The first step in the DT development process is to identify the physical system properties and plan the DT integration into the physical system. Production systems are identified by their stakeholders, processes, components, and quality specifications. These factors categorize the physical system to be twinned and determine the needed tools for simulation and operations management.

Defining stakeholders is important for understanding the operators, their experiences, and their tasks. For instance, DT applications used by assembly-line workers are operational-level applications, which are designed for assisting them through assembly operations. While DT applications for production managers are business-level applications such as inventory

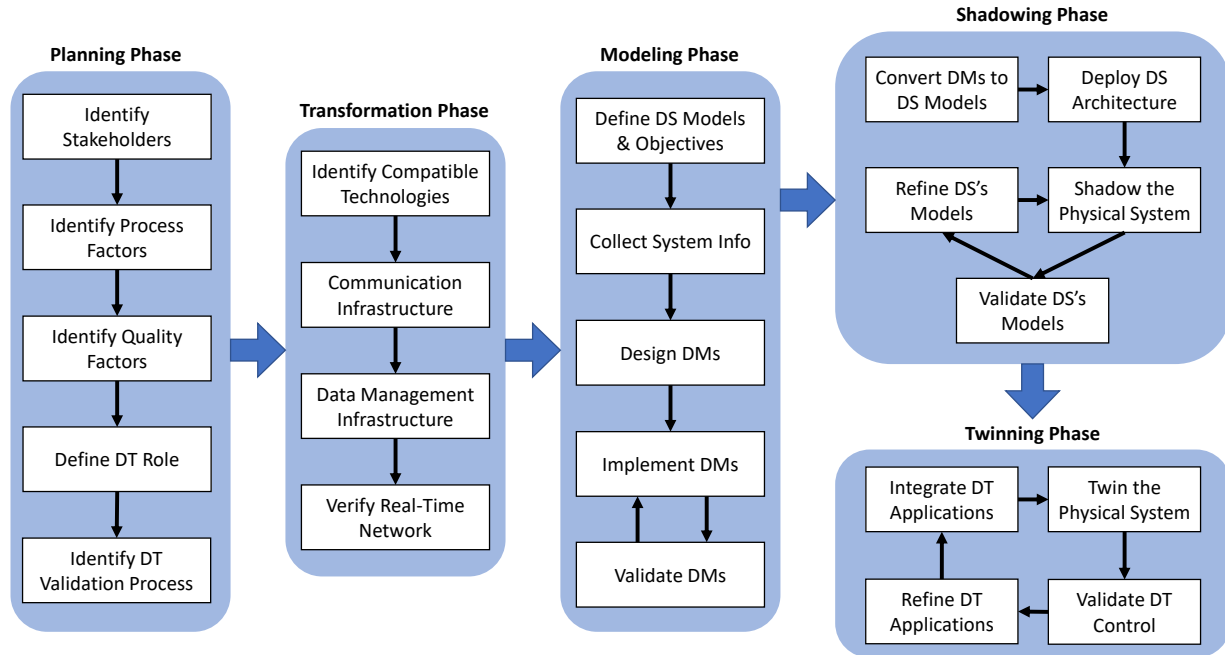


Figure 8.4 The proposed digital twin design methodology

management, internal logistics management, and supply-chain management.

Production processes are identified by product properties and production factors. Product properties describe the product's Bill of materials (BOM), geometry, and movement through the system. While production factors describe the input resources and operations that are needed to produce the product. For example, the production factors of a car assembly line are defined by the body parts, assembly steps, and machines used in each step.

The production system's components are identified along with the manufacturing process. DT designers need to understand the machines' operations and behavior. For example, a conveyor transporting products between multiple machines needs to be described in detail before modeling that conveyor in a DT. The conveyor type, belt or rollers, and accumulation behavior affect the flow of the products in the system. While existing sensors on the conveyor determine the level of visibility.

Identifying the product's quality specifications is also done in the planning phase. The goal is to define the quality inspection process, the product's specifications targeted, and KPIs used. Product specifications are defined early in the planning phase to highlight the factors affecting the product's quality and capture their measurements from the physical system.

The definition of DT's granularity level, scope, and roles is crucial. Multiple DTs work

concurrently to represent different parts of the same system. Defining the DT scope and roles enable DT designers to specify the CPS requirements for the DT, and establish the DT's level of detail. Similarly, the DT validation process is defined in the planning phase. The DT validation process is used for validating the DT simulation models and testing the DTs' ability to perform their intended roles. The validation process usually consists of behavioral tests, real-time latency tests, statistical tests, and deviation tests.

8.3.2 Transformation Phase

In the transformation phase, the physical system is transformed into a CPS system. CPS systems provide the required communication infrastructure to connect DTs to both the physical and cyber domains. Existing industrial communication systems are utilized to access the machine's information, while IoT edge devices are used to connect offline components to the CPS system.

DT designers identify existing communication systems and their compatible technologies. Minimal changes are needed for Industrial Ethernet (IE) network systems, while Fieldbus communication networks are connected to IE networks using IE-link devices. Offline and analog machines require additional interfacing devices to integrate into the CPS system. For instance, cameras are placed at sensor gauges to capture the gauge value. While IoT devices are attached to offline electric machines to passively read their status.

The communication infrastructure is designed and implemented after the physical system components are studied. The infrastructure design determines the network topology and switching nodes to connect different parts of the system. Wireless networks are used to connect moving components such as mobile robots and transporters, while wired networks connect stationary components.

Once the communication infrastructure is set up, the production system's machines are connected to the newly deployed IE network. The supervisory system needs to extend the data management system to enable data-stream processing. The ICT such as big data analytics and cloud computing are used to process the incoming stream of structured and unstructured data [2].

Finally, CPS real-time network requirements are verified. The goal is to ensure the CPS network's ability to pass real-time information between the physical and cyber domains. IEC 61784-2 standard defines the network KPIs used in ensuring the real-time performance of IE networks [145, 146]. These network KPIs include but are not limited to message latency, throughput, and fault recovery time.

8.3.3 Modeling Phase

In the modeling phase, Digital models (DMs) are designed, implemented, and verified. DMs are the computational core of DS and DT [2, 138]. A physical system is represented using multiple DMs. Each DM describes a part of the physical system and shares its status with other DMs. DMs are implemented using several modeling techniques such as semantic ontology, discrete-event simulation, system dynamics simulation, statistical modeling, and mathematical modeling.

The modeling phase adopts traditional simulation model development's steps [147]. Firstly, DM's objectives are determined in order to define the scope and the modeling technique to be used. Secondly, the physical system and machines' specifications are collected along with available historical data. The collected information is used in building and validating each in-development DM. Then, a conceptual design of the DM is created. The conceptual design defines the physical system behavior and the level of abstraction used in describing the system. Functional flow diagrams, Flowcharts, and UML diagrams are used to represent the modeled system and the interactions between its components. The conceptual DM design is implemented using programming languages and/or simulation tools. Finally, confirming the DM's ability to mimic the real system needs to be done before converting the DM into a DS. The DM is validated by running the DM's simulation using multiple pre-defined scenarios. The DM's implementation is refined according to the validation results and the stakeholders' feedback.

8.3.4 Shadowing Phase

In the shadowing phase, DMs are transformed into DS models and integrated into the CPS system. The objective of this phase is to ensure the DS models' validity and stability over a prolonged operational period. Since DMs are developed as offline simulation models, modifications to DMs are performed to convert them into online DS's models. Automatic dataflow from the CPS is included in the DM through the integration of message broker connectors into the DM. Additionally, synchronization between the physical system and the DM is needed. DMM provides an API to communicate with a DM and control its flow [2]. Message brokers pass the physical system messages to the DMM. Then the DMM updates the DM according to the status messages received and synchronizes the DM with the physical system.

DS models shadow the physical system once they are integrated into the CPS system. The DS models are initialized with live system status. The DMMs listen to any status update messages published to the message broker. The DMMs track the status variables received

from both the physical system and DS models.

The DS's models are validated by running for long operational periods. During validation, the state variables' deviation range and response time are calculated. The DS model deviation must not exceed a predetermined threshold to keep the model's abstraction error acceptable. Similarly, the response time of a DS model affects the correctness of any decision built on its value. Therefore, DS models require a response time faster than the physical system's critical time. For instance, a DS model of a robotic arm requires response time in milliseconds. While a DS model of a truck fleet requires response time in seconds or minutes. Log messages generated during the DS model's validation step are used to identify deviation sources. The DS model's logic is then improved over multiple iterations of deployment, validation, and refinement cycle.

8.3.5 Twinning Phase

In the twinning phase, the DT decision-support applications are integrated. DT provides intelligence to the CPS system through the use of data analysis, simulation models, Machine Learning (ML) models, and recommendation systems. DT applications are deployed to the CPS in a similar way to DS's models. DT applications are initialized using the real-time system status. DT executes simulation models and predicts the future states of the CPS. It then recommends the best course of action. The DT's decisions are sent back to the CPS according to the predefined DT's role. The DT's decisions could take the form of machine settings, control signals, or operator notifications.

DT decisions are validated by comparing them with the defined operational settings value ranges. Additionally, system experts supervise the DT applications by analyzing the recommendations and actions generated by these DT applications. Based on the stakeholders' feedback, the DT application is refined to reduce the simulation-to-reality gap in the DT's decisions. Moreover, DT applications can be added to the DT and improved iteratively.

8.4 Experimental Testbed

An intelligent CPS (ICPS) is being built as a test to design and assess customer-centric industry 4.0 systems. The laboratory is used to develop and improve industry 4.0 concepts in order to reach Technology Readiness Level (TRL) level 4. The laboratory is being built as a CPS quality inspection and warehousing system. The CPS warehouse operations include dispatching, inspection, labeling, and storage. A DT is developed to provide operation scheduling, dispatching, and virtual commissioning (VC). The CPS warehouse testbed is

used to validate the proposed DT methodology. Developing the DT in parallel with the CPS warehouse enables system designers to test different scenarios and accommodate potential problems during the design phase. Additionally, it eases the process of connecting the DT to the physical system.

8.4.1 Planning Phase

DT stakeholders are identified in the planning phase. In this testbed, two stakeholders are identified; warehouse operators and managers. Warehouse operators use the DT for real-time monitoring and operation management. While managers use DT simulation models as a sandbox to assess different layout configurations and equipment settings.

In this study, the ICPS laboratory is used as a testbed for a CPS of a warehouse. The laboratory includes multiple robotic arms to manipulate and move products. It also contains Autonomous mobile robots (AMRs) to move the products to different stations. Figure 8.5 shows the warehouse layout configuration that is used. The CPS warehouse system consists of 9 machines that are distributed over 7 stations and 2 AMRs moving between them.

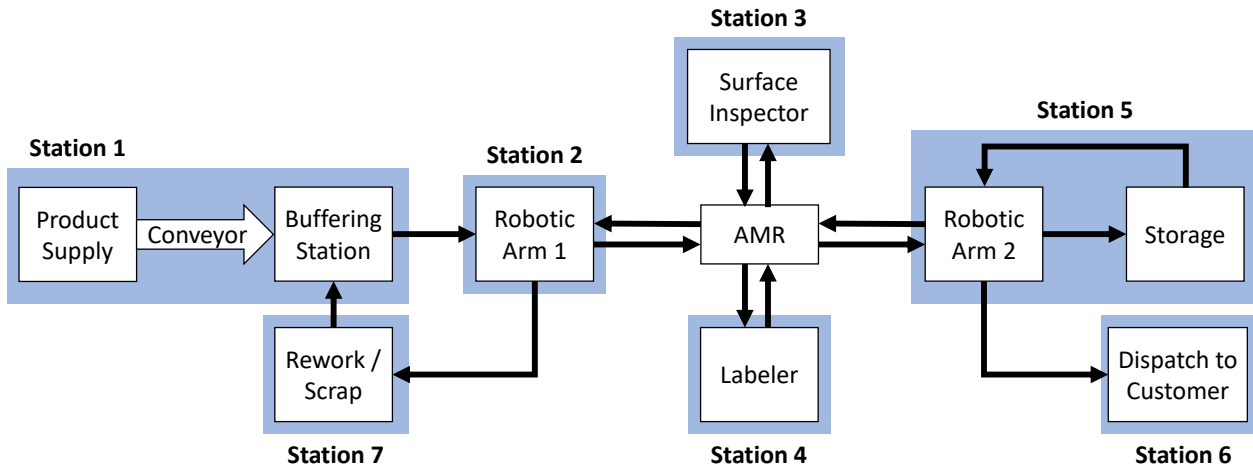


Figure 8.5 Schematic diagram of the warehouse layout

The products are imported into the warehouse at station 1. Products are moved by the *conveyor* to the *buffering* zone. *Robotic arm 1* loads products on AMRs at station 2. AMRs move the loaded products to the *surface inspector* for surface roughness quality inspection. *Robotic arm 1* and *2* are used to rotate products. Each product type has a different inspection sequence according to its geometry. Products failing the surface roughness specification are either reworked or scrapped at the *rework/scrap* zone in station 7. Products passing the quality check are stored in station 5. When a customer's order is ready for dispatching,

robotic arm 2 unloads the order's products from the *storage*. Figure 8.6 shows the warehouse operations and the different equipment that are used in the process.

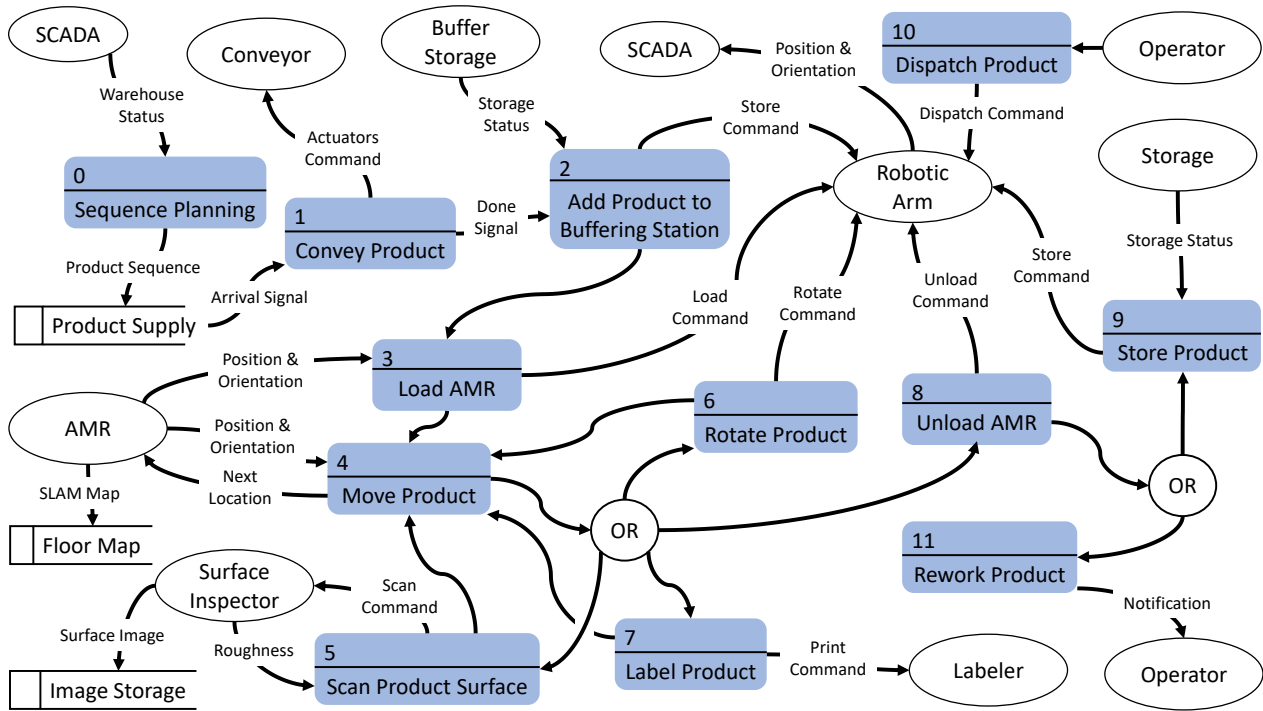


Figure 8.6 Functional flow block diagram of the CPS of the warehouse

The CPS of the warehouse is controlled using a SCADA system deployed on an industrial controller. Operators and managers control the warehouse operations using the SCADA. Equipment settings, surface roughness inspection images, and dispatching commands are sent through a wireless network. Figure 8.7 shows the CPS control system and the data flow between its different levels. *Surface inspector* measures products' surface roughness using depth camera images. These images are processed and stored on the industrial controller. Multiple sensors are used to track the products' progress and location. RF read/write modules identify products at the different stations. While infrared sensors track the state of storage cells. AMRs send their location periodically. Sensor readings are aggregated at the industrial controller through the wireless network.

In this study, the DT is developed to track and manage the CPS warehouse operations and to ensure that all products are dispatched to the customers on time. The DT provides real-time monitoring of products and warehouse equipment. Product progress, surface images, and order information are accessible in real-time through the DT. The results of the quality inspection check inform the operators if a product is not conforming to specifications. Customer orders could be delayed due to multiple reworks and product scrapping. The DT

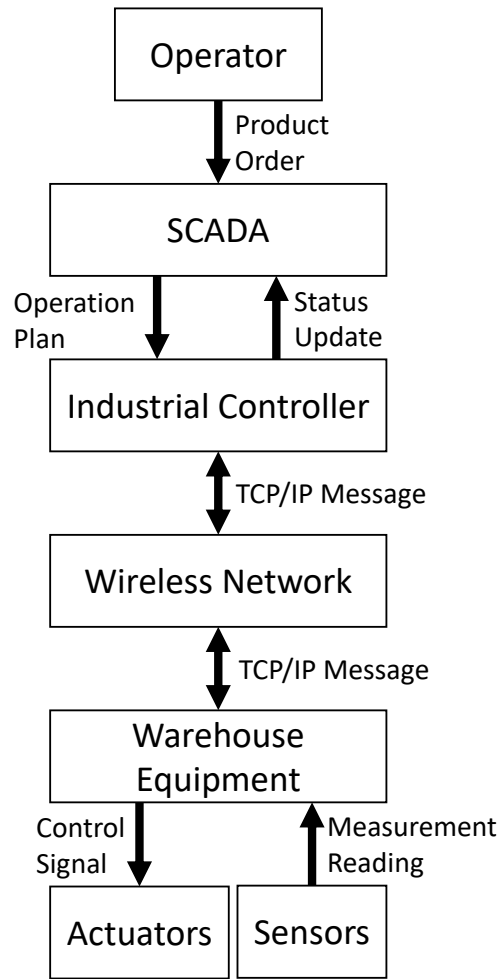


Figure 8.7 The CPS control system architecture diagram

adjusts products' status according to online quality inspection results. Warehouse operations are rescheduled if a product failed the quality inspection test. The dispatching and delivery dates are adjusted accordingly.

The DT is validated over two steps. Firstly, The DT's real-time monitoring is validated by running the DS in normal operation. The DS latency is measured to assess its real-time constraint. Secondly, the DT's products dispatching sequence adjustments are validated by the warehouse operator. The DT dispatching sequence is checked for invalid and inefficient actions.

8.4.2 Transformation Phase

The warehouse is designed as a CPS system with an Industrial Ethernet (IE) wireless network for connectivity. Hence, the warehouse has the needed communication infrastructure to integrate the DT. Figure 8.8 shows the network topology for the warehouse equipment. A single switching node is used to connect all the warehouse equipment since the number of equipment is low. The current topology complies with the IEC 61784-2 standard for real-time IE networks.

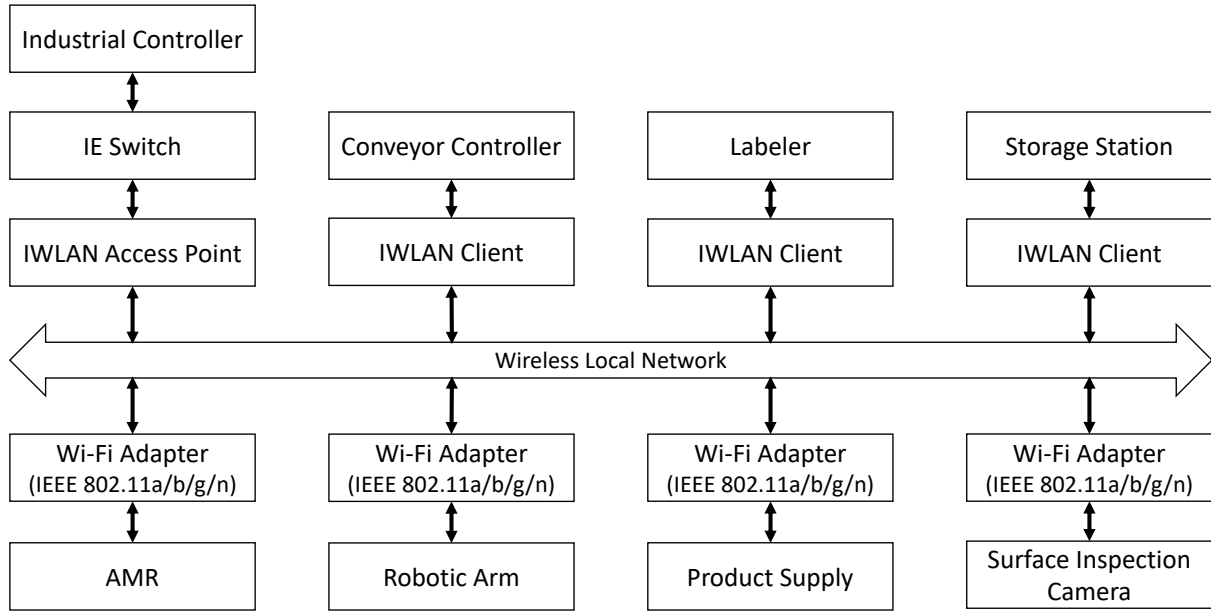


Figure 8.8 The CPS communication network topology

ICT technologies such as cloud computing and Pub/Sub communication systems are used in building the DT [138]. Simulation tools and programming languages are used in developing the DS models and DT applications. AnyLogic® simulation tool is used to model the warehouse operations. *OWLready2* library is used to build a semantic ontology model representing the system components and their relations. Python and PyTorch are used to implement Reinforcement Learning (RL) dispatching agents. Python and Java programming languages are used to implement the DT components and their DMM interfaces. Apache Kafka® is used as Pub/Sub communication channel to connect DT components for information sharing. The DT is deployed on GCP using GKE. A Kubernetes cluster composed of 8 standard E2 nodes is used to host the DT. Each node has 2 shared virtual CPUs with 4GB RAM and 100GB storage memory.

The DT interacts with the CPS warehouse equipment through the SCADA and the wireless

network. Figure 8.9 shows the DT components and their connectivity with the CPS warehouse control system. The Kafka message broker passes the physical system information to the DS models. The DS models update their status and share them with other DT components. The DT applications send the decisions back to the CPS warehouse through the SCADA and web-based interfaces.

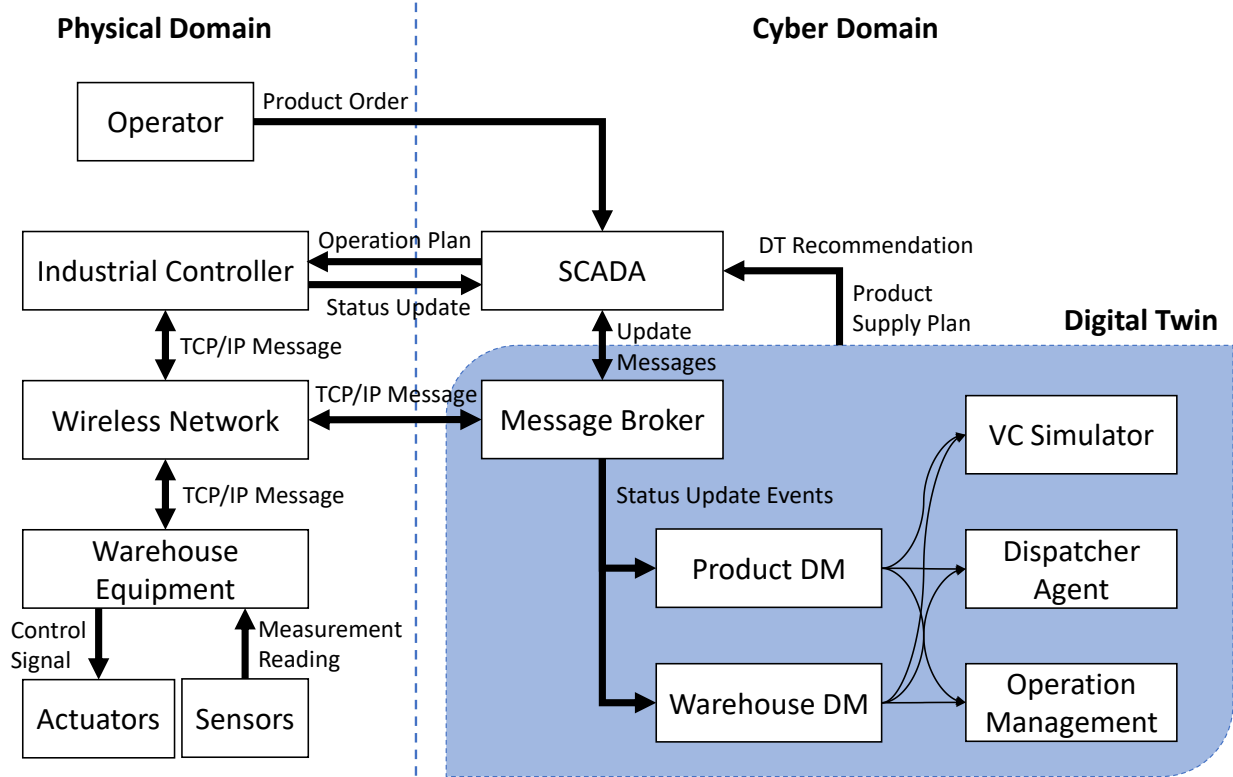


Figure 8.9 Digital twin system integrated into the CPS control system

The CPS communication infrastructure's real-time capability is verified. The IE wireless network provides real-time communication between the warehouse equipment and the industrial controller. The Kafka Pub/Sub communication system ensures a high-throughput low-latency messaging system as indicated in [2]. GCP provides high-computational power for DT components.

8.4.3 Modeling Phase

Two DM are developed to represent the CPS warehouse. In this phase, DMs are not yet connected to the physical system. The warehouse equipment's settings are passed to the DMs using Excel sheets. These Excel sheets contain the warehouse layout, equipment specifications, quality specifications, and operations performed.

Firstly, the semantic ontology model, *Product DM*, is developed to describe the warehouse entities and their relationships. Warehouse equipment are modeled as entities placed at a station performing one or more processes. Products move between stations. Each product belongs to a unique product type, which is used for identifying its quality specifications and workflow. Figure 8.10 shows the ontology diagram of the warehouse system. *OWLready2* is a OWL Python library used to generate and update ontology graphs during runtime. New products are added to the ontology graph programmatically. While existing products and equipment update their properties according to the CPS warehouse status. *OWLready2* enables operators to export the ontology graph as an RDF/XML file.

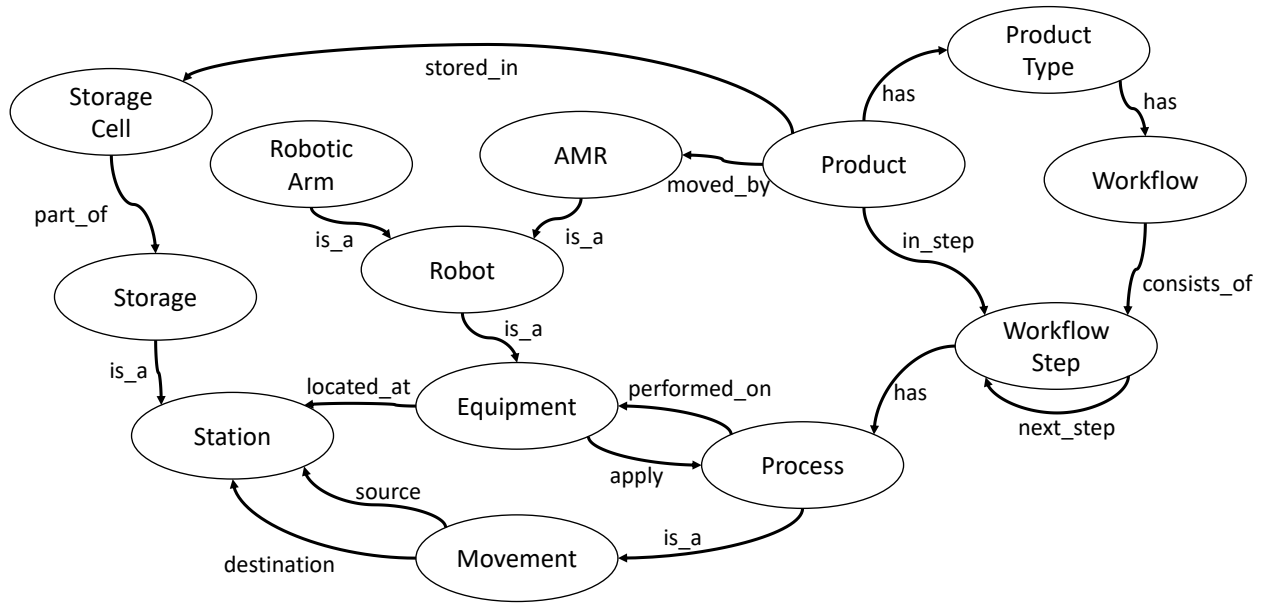


Figure 8.10 *Product DM* ontology diagram

Secondly, a Discrete event simulation (DES) model, called *Warehouse DM*, is developed to simulate the warehouse operations. AnyLogic® is used to build the DES model according to Agent-based model (ABM) approach. AnyLogic® is a Java-based simulation tool that provides a wide range of libraries for model designers. The data-driven modeling approach is adapted to build the CPS warehouse model. Figure 8.11 shows a screenshot of the warehouse simulation model during execution. Warehouse equipment are modeled as agents that process products according to equipment settings. AMRs are modeled as transportation agents that move products between stations. Products' workflow steps are also modeled as agents, where each workflow step defines the equipment used, the duration of the operation, and the next workflow step according to the product's status. New product types and operation sequences are introduced by adding them to the Excel sheets' configuration. The warehouse DES model

is validated by the stakeholders.

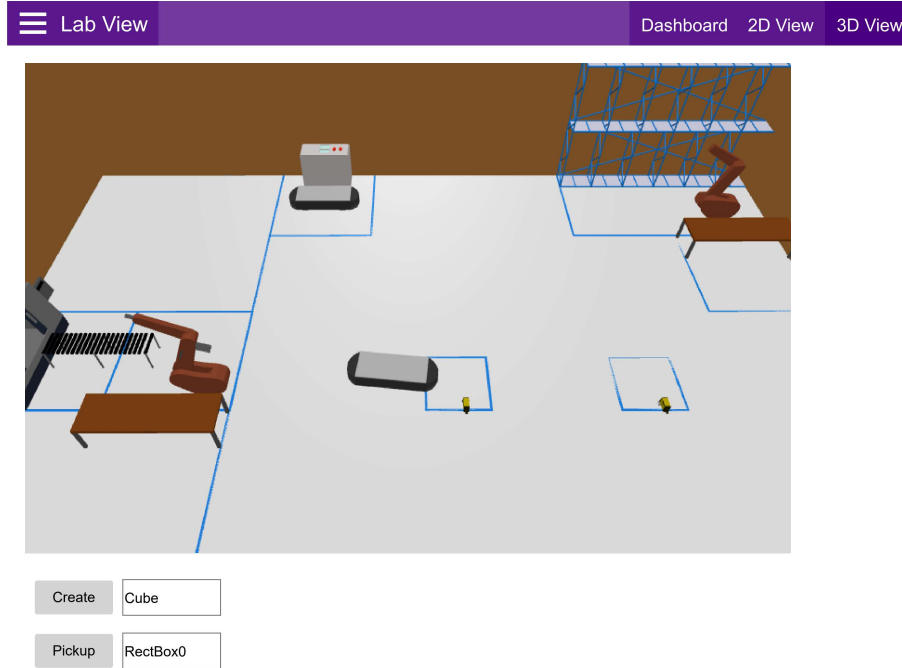


Figure 8.11 *Warehouse DM* screenshot of the model's 3D view

8.4.4 Shadowing Phase

In this phase, DMs are converted into DS models. Digital model manager (DMM) is integrated into the DMs. Then, the SCADA system is programmed to pass the CPS warehouse information to the Kafka message broker. Since the goal is to validate the methodology process, an instance of the *warehouse DM* is used instead of the physical warehouse during development and validation. The use of simulation models for validation was used in literature as in [141].

A Python-based DMM is implemented to integrate *product DM* into the DS. A Confluent Kafka® Python-based client is used to connect the OWL model to the Kafka message broker. DMM listens to the messages published on the Kafka Pub/Sub system. DMM then processes and passes the changes in the warehouse's status to *product DM*.

A Java-based DMM is implemented to integrate *warehouse DM* into the DS. An Apache Kafka® Java-based client is used to connect the AnyLogic model to the Kafka message broker. Products and equipment states are received by the DMM through the Kafka Pub/Sub system. The AnyLogic model is modified to keep the model synchronized with the physical system [138]. Delay modules are replaced by conditional hold modules to control the operation flow

according to Kafka Pub/Sub messages.

8.4.5 Twinning Phase

In the final phase, DT applications are implemented and added to the system. Three DT applications are implemented and connected to the DT. The DT applications provide decision support for the stakeholders; warehouse operators and managers. DT applications are connected to the Kafka Pub/Sub system to receive the CPS warehouse and DS models status.

A modified version of *Warehouse DM* is transformed into a DT application, called *Warehouse VC*, in order to provide simulation capabilities for warehouse managers. *Warehouse VC* initializes the simulation model using real-time warehouse status pulled from the Kafka Pub/Sub system. The simulation model then runs the warehouse configuration selected by the manager. The manager assesses the warehouse configuration using dashboards inside the simulation model. *Warehouse VC* is also used to simulate new production plans.

A dispatcher DT application, called *dispatcher agent*, is implemented to select which product to inspect next by assigning a priority value for each product in the queue. A pre-trained Python-based RL agent is implemented to set a priority value for products in-progress. The SCADA reorders the products according to their priority value. Figure 8.12 shows a UML sequence diagram for the products' sequence updates. The SCADA sends a notification of the new products order to the DT. The DT initiates the product's *dispatcher agent* to reorder products in-progress. Warehouse operators can switch between the RL dispatching policy and a simple First-In-First-Out (FIFO) policy.

Another pre-trained Python-based RL agent is used to select the next AMR destination, where the RL agent decides which robotic arm to use. The AMR *dispatcher agent* is called before the product rotation step. The SCADA then sends the next destination coordinates to the AMR. Figure 8.13 shows a UML sequence diagram for the workflow step processing and AMR next destination decision.

Finally, an *operation management* application is implemented. *Operation management* tracks products and orders status. Once a product fails a quality inspection check, *Operation management* notifies the warehouse manager and adjusts the product supply plan.

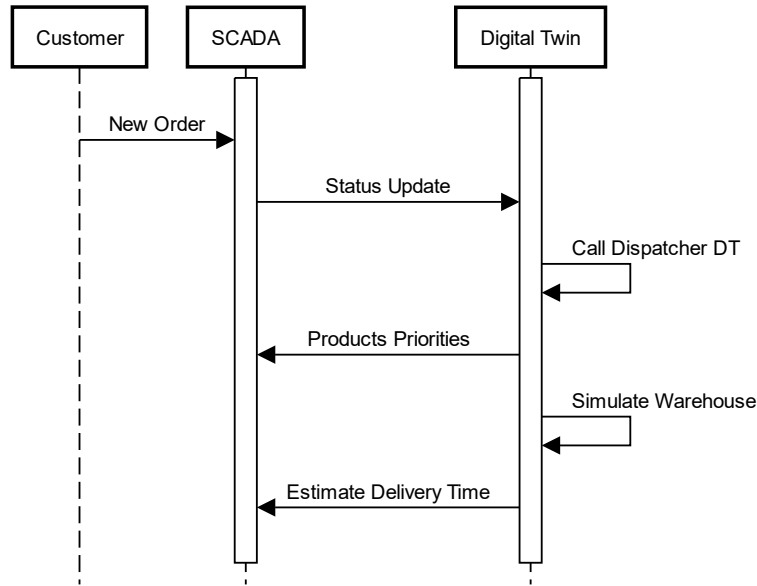


Figure 8.12 UML sequence diagram for products reordering

8.5 Discussion

CPS connectivity is crucial for DT development. Wireless networks create a shared data-bus to exchange real-time information. The SCADA serves as a communication hub between the physical and virtual systems. Communication between the physical components and DT is possible through the IE wireless network and the SCADA.

Providing a realistic VC simulation tool for stakeholders was the main objective that directed the DT design. *Warehouse DM* was built with a high level of granularity in order to be reused for multiple different purposes. *Warehouse DM* provides real-time monitoring of warehouse operations. In-model dashboards visualize the warehouse status and performance to provide insights for stakeholders. For instance, an overview dashboard visualizes the CPS warehouse performance as shown in Figure 8.14. While dynamic dashboards present online products' and AMRs' state as shown in Figure 8.15 and 8.16.

Expanding the CPS warehouse operations by adding new equipment is applicable using the CPS control system. Newly added equipment are connected to the IE wireless network using Wi-Fi adapters or IWLAN client modules. The DT can be expanded with new DS models and DT applications due to adopting a generalized DT framework as in [138].

The proposed methodology enables new DT applications to be integrated into the DT during runtime. New DT applications are first planned according to stakeholders and DT objectives. Additional DMs are implemented if needed, then converted into DS models and DT

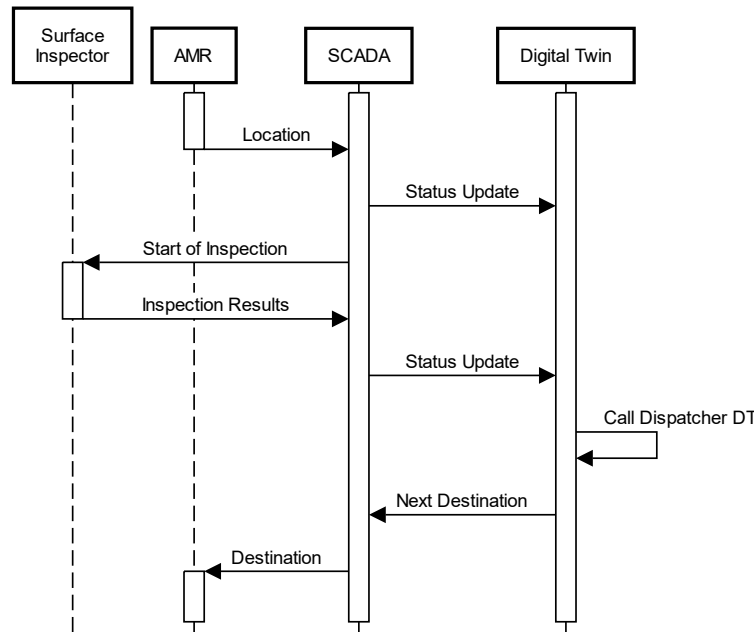


Figure 8.13 UML sequence diagram for moving products after inspection

applications. Multiple DT applications could use the same DS models as shown in the CPS warehouse's DT. The proposed methodology allows the implementation of different DS models and DT applications concurrently.

Adapting the agile development approach in the proposed methodology enables iterative DT development. Agile development emphasizes the frequent delivery of working systems and continuous improvement of the system. The proposed methodology uses iterative steps at the implementation phase, shadowing phase, and twinning phase. Sustainable development and improvement keep the stakeholders involved in all phases of the DT development.

8.6 Conclusion and outlook

Manufacturers are adopting the Digital twin (DT) concept. DT provides real-time fast-response supporting applications for decision-makers. Additionally, DT embeds intelligence into smart production systems. DT benefits and business opportunities in the industrial domain are limitless. In this paper, a methodology for DT development is proposed. The methodology defines a stepwise process for designing and developing DTs for manufacturing systems.

The proposed methodology divides DT development into an iterative process. Firstly, DT stakeholders and the physical system description are identified. Additionally, the role of DT in

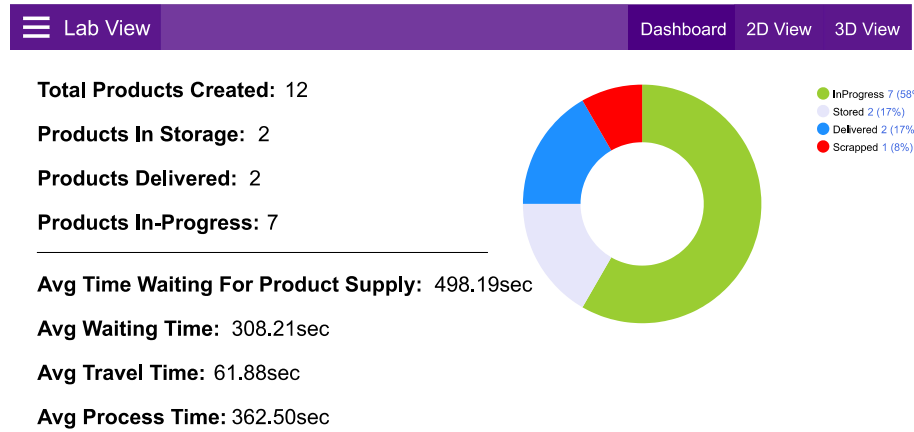


Figure 8.14 A screenshot of the CPS warehouse dashboard in *Warehouse DM*

the production system is defined alongside the DT validation process. Secondly, the physical system is transformed into a CPS system and compatible ICT technologies are selected for DT implementation. Thirdly, Digital models (DMs) of the physical system are developed and validated. Fourthly, DMs are converted into Digital shadow (DS) models and iteratively refined. Finally, DT applications are integrated into the CPS system.

A CPS warehouse testbed is used to validate the applicability of the proposed methodology. The proposed methodology was followed to develop a DT for virtual commissioning (VC) and operation management. A Kafka Pub/Sub communication system is used in collaboration with the SCADA system to pass physical system information to the DT. Several DS models and DT applications are developed to schedule the warehouse operations according to real-time status.

The proposed methodology provides the following benefits for DT designers:

- **CPS transformation:** The transformation phase emphasizes the need to transform the physical system into CPS system. The CPS communication infrastructure required for the DT is designed before starting the implementation phase. Selecting compatible technologies during the planning phase reduces the probability of integration problems.
- **Iterative implementation:** The iterative development of DMs, DS, and DT applications hastens the feedback frequency to DT designers. This enables DT designers to adopt an Agile development cycle.

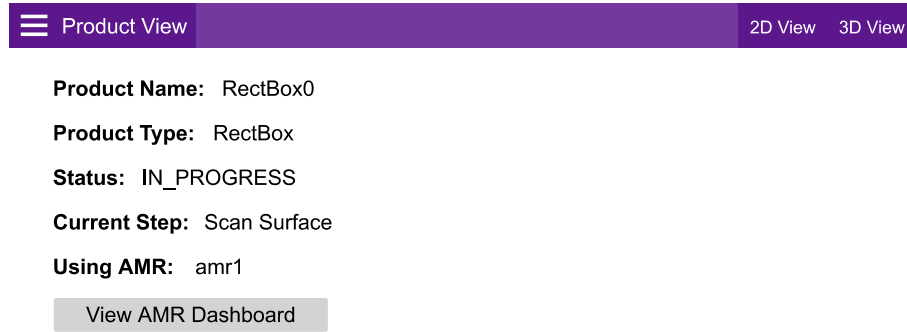


Figure 8.15 A screenshot of the product dashboard in *Warehouse DM*

- Keeping stakeholders involved: The proposed methodology requires the validation of each DT component in collaboration with stakeholders. This approach keeps different stakeholders involved in the development process. Moreover, frequent feedback to DT designers aligns the DT's scope during development.
- DT generality: Adopting a generalized DT framework during the planning phase ensures the generality and expandability of the developed DT. New DS models and DT applications can be added after the DT deployment. Pub/Sub communication systems such as Kafka enable new DT applications to connect to existing DTs. Cloud orchestration using Kubernetes allows system expansion and adding new applications.

8.7 Future work

The proposed methodology serves as a proof of concept that will be further improved upon and validated in the future. The ICPS laboratory provides an expandable workspace for researchers to study new concepts. Some of the future work to be studied are:

- Layout optimization: The DT simulation model of the warehouse enables production managers to simulate different warehouse configurations. However, the use of optimization algorithms should be investigated.
- AMR charging schedule: The AMRs are programmed to go to the charging station once the battery level drops below a certain critical threshold. AMR returns to operational

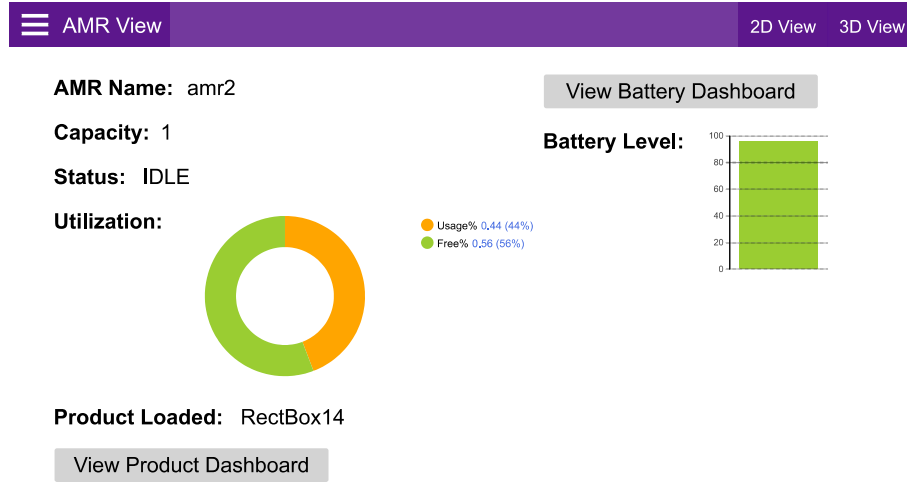


Figure 8.16 A screenshot of the AMR dashboard in *Warehouse DM*

mode after its battery is fully charged. A RL scheduling solution will be investigated to schedule the AMR charging time and thresholds to reduce their out-of-service duration.

8.8 Statements and declarations

8.8.1 Competing interests

The authors declare that they have no conflict of interest.

8.8.2 Funding

This work was supported by Fondation Canadienne pour l'Innovation (FCI) for the project *Intelligent Cyber Value Chain Network (CĒOSNet)*, grant number 40115.

8.9 Author's Contribution

A. AboElHassan conducted the conceptual design of the DT development methodology, contributed to the warehouse system design, developed the DT framework and its components, and wrote the manuscript. A. Sakr contributed to the warehouse system design and its communication system. S. Yacout provided guidance for the conceptual design and methodology. All authors reviewed the final manuscript.

CHAPTER 9 GENERAL DISCUSSION

Distributed systems and Information and communication technologies (ICT) are well-studied fields in the computer science domain. Integrating ICT into manufacturing systems triggered the fourth industrial revolution. Since then, applying computer science concepts in manufacturing systems is gaining more momentum. Industrial Ethernet (IE) provides fast reliable communication channel to connect between OT and IT systems. Machine-to-machine and machine-to-cloud connectivity allowed the deployment of intelligent applications in close proximity. Investments in Industrial internet of things (IIoT) and smart production systems are growing over time.

Distributed system concepts provide connectivity for the proposed DT framework. Pub/Sub communication systems are used in connecting DT components. Pub/Sub communication systems, such as Apache Kafka®, provide real-time information-sharing infrastructure. In Chapter 6, Kafka is proven to pass information within milliseconds latency. Data duplication and distribution over multiple Kafka brokers increase service availability. Acknowledgment timeout and write-once approaches provide fault tolerance for network faults.

Additional benefits are achieved by applying distributed system concepts. Combining event-driven architecture with microservices enables the design of asynchronous DT. Microservices are used in designing DT components as loosely-coupled modules. While Kafka shares information between DT components, RESTful HTTP interfaces provide service-driven functionality. HTTP requests are used to control the logic flow of DMs by the DS coordinator. DT manager also runs and manages DT applications through RESTful HTTP requests.

Containerization is not widely used in the industrial domain despite its importance in cloud computing. Digital model (DM) modularization is achieved by deploying DM executables into Docker containers. Containerized modules are deployable on different infrastructures, such as cloud platforms, on-site servers, and IoT edge devices. Orchestration systems such as Kubernetes (K8s) automate the deployment of DTs. K8s manages the distribution of DMs on the computational resources. K8s's rolling-update provides DM version management through container replacement. K8s control plane provides fault-tolerance using a fail-safe approach. DT scaled through increasing the number of containers specified at the K8s scheduler.

Artificial intelligence (AI) usage in DT applications provides smart decision-support and data analysis tools. Machine Learning (ML) algorithms, such as Artificial neural network (ANN), are used in diagnosis and prognosis tools. ML classification models generate statistical-driven identification of behaviors and faults. While time-series regression models predict future

events and states. Reinforcement Learning (RL) agents learn to take decisions according to pre-defined reward functions. RL agents are used in recommending actions for decision-makers according to the latest system state, such as *dispatching recommendation DM* in Sect. 7.6.3 and *dispatcher agent* in Sect. 8.4.5. Simulation models are used in combination with RL agents to capture RL actions' consequences.

In addition to AI-based DT applications, non-AI DT applications are also used for decision support. Non-AI approaches are used in DT applications such as state visualization, customer relationship management, and future state forecast. During the research work, multiple non-AI DT applications were developed, such as the Human-DT interface in Sect. 7.6.3 and *Warehouse VC* in Sect. 8.4.5. In Sect. 7.6.3, a BI dashboard is used as a DT application to compile and visualize the system's state for operators from multiple DS models. In Sect. 8.4.5, a simulation model is used as a DT application to simulate different system configurations.

9.1 Research objectives

The research study achieved the objectives presented in Sect. 1.3. Chapter 4 provides a conceptual design of the DT framework used as a roadmap to achieve both objectives *A* and *B*. Distributed system concepts are used to achieve objective *A* in Chapter 6. Docker containers and K8s enable the deployment of multiple DMs to generate a holistic view of the physical system. While the use of Microservices and Apache Kafka® allow the flexibility of DS framework. In Chapter 7, the DS framework is converted into a DT to achieve objective *B*. The DS framework and its distributed system enabling concepts are used to develop a general-purpose DT. K8s and Apache Kafka® enable multiple DT applications to run concurrently using the same DS models. While AI concepts are used to develop recommendation systems for decision-makers. Finally, the Agile development approach is adapted to achieve objective *C* in Chapter 8. The incremental step-wise DT development methodology provides a fast-paced agile-driven process to develop DTs.

9.2 Research questions

The research questions asked in Sect. 1.4 are addressed throughout the research study. The questions are answered over multiple chapters of the study. A summary of the explored answers is described as follows:

RQ1 How to differentiate between DS and DT?

Chapter 4 provided a functional-driven definition of both concepts. This

definition is used to identify the main components of DSs and DTs and their relations. A DS framework is developed in Chapter 6. While DS is used as a core component in a DT framework in Chapter 7.

RQ1.1 What are the main functionalities of DS and DT?

Chapter 4 addressed the overlapping definitions and functionalities of DS and DT. In Chapter 4, a functional-driven distinction between both concepts is proposed. DS is defined as a *digital replica of the physical system that matches its physical counterpart's state in real-time*. While DT is defined as a *real-time decision-support sandbox*. This definition distinguishes between both concepts according to their roles in the decision-making operation. DS's role is to generate a digital representation of the physical system using models and simulations. While DT's role is to provide decision support using rule-based and AI algorithms.

RQ1.2 How to distinguish between data acquisition, modeling, and decision support?

Chapter 4, 6, and 7 addressed the proposed separation between data acquisition, real-time models, and decision-support applications. Where source gateways collect real-time data from the physical system and pass it through the Pub/Sub communication channel. DS models utilize the collected data to generate a digital representation of the system. Finally, DT provides decision-support recommendations using the compiled digital representation. The functional separation between these 3 components enables the systematic interaction between each other.

RQ1.3 What are the main differences between DS models and DT applications?

In Sect. 7.5, the differences in developing DS models and DT applications are investigated and discussed. The distinctions are compiled into three main focus points; model flow control, dynamic behaviors, and modeling system stochasticity.

RQ1.4 How to transform simulation models into DT applications?

In Sect. 7.5, the differences in model flow control between DS and DT shows how a simulation model is executed in a DT application. The simulation model is first initialized by the latest physical system state. Then the simulation mimics the system operations till reaching a predefined stopping condition, which is either the end of the simulation duration or a specific

event occurrence. This approach enables DT to use simulation models for different applications such as virtual commissioning (VC), deadline estimation, and scheduling.

RQ2 How to develop DTs using AI and distributed system concepts?

This question is investigated in Chapter 6 and 7. Chapter 6 presented a DS framework based on distributed system concepts. While Chapter 7 expanded the proposed framework into a DT framework to provide decision support capabilities and embedding AI.

RQ2.1 Can multiple independent DMs be used to provide a multi-physics multi-scale description of the physical system?

Chapter 6 confirmed distributed system capabilities to deploy multiple independent DMs. K8s allows the deployment of multiple DMs concurrently. While Kafka enables information sharing between these DMs. Using these technologies, multiple DMs are developed in Chapter 6 case study. Concurrent DMs developed by different tools for modeling different parts of the physical system. The DMs share their information over the Kafka Pub/Sub communication channel.

RQ2.2 Since simulation models run faster than real-time, how to control the time flow of time-dependent models such as continuous-time models and DES?

The use of DMM provides time flow control over its corresponding DM as discussed in Chapter 6. DMM is configured to control the DM time flow according to the physical system messages and time resolution. Microservices for interfacing between DMM and DM, which provides service-based interactions to update the DM variables and advance its time. Figure 6.6 shows the HTTP requests sent from DMM to control the DM time flow.

RQ2.3 DS models track the physical system, therefore synchronizing DMs with the real system's state is crucial. How to synchronize DMs with the physical system?

Similar to *RQ2.2*, DM synchronization is addressed in Chapter 6. DMMs synchronize their corresponding DMs using the physical system messages received. During DM initialization, DMM pulls the latest messages at Kafka to initialize its DM with the current physical system state. Afterward, DMM periodically updates the DM variables with physical system messages.

RQ2.4 How virtualization can be applied to DMs?

Docker containers are used to modularize DMs as shown in Chapter 6. DM is exported into a dockerized version or an executable application. The exported DM is then containerized using docker files defining the model dependencies, licenses if exist, and environment variables.

RQ2.5 How to connect a DM to the Pub/Sub communication channel?

A Kafka client module in the DMM provides access to the Kafka Pub/Sub communication channel as shown in Chapter 6. DMM subscribes to all needed topics at the Kafka cluster. DMM then updates the DM variables by formatting and forwarding the messages received to the DM.

RQ2.6 Does Pub/Sub communication channel provide real-time messaging capabilities for DTs?

Yes, the case study results in Sect. 6.7 show Kafka's capabilities to provide message latency in milliseconds enabling real-time communication.

RQ2.7 How to embed RL into simulation models?

Chapter 5 discussed the potential of developing RL algorithms to be deployed inside simulation models. This approach eases the use of RL by model designers. The proposed approach uses inter-process communication (IPC) to connect between RL and simulation models. Then an internal model library is developed to collect the model's information and pass it to the RL for training and evaluation.

RQ3 How to iteratively develop and improve DTs?

An iterative methodology for developing DTs is proposed in Chapter 8. The proposed methodology provides an iterative step-wise method based on agile development.

RQ3.1 What are the requirements for a physical system to be DT compatible?

Sect. 8.3.2 describes the main requirements needed for a physical system to become DT compatible. The physical system needs to be transformed into a CPS with a real-time ethernet communication network to enable DT development. Passive equipment, such as shelves, and offline machines are equipped with additional sensing components in order to capture their state and send their information to the DT.

RQ3.2 How to systematically develop DTs?

Chapter 8 proposes a DT development methodology to design, implement, and deploy DTs. The proposed methodology is defined as a 5-phases process. DT scope is first defined in the planning phase. Then the physical system is converted into a CPS to fulfill DT connectivity requirements. Simulation models of the physical system are designed, implemented, and validated. These models are then adopted into DS models. Finally, DT applications are developed

RQ3.3 How to run multiple DT applications concurrently using the same DS model?

In Chapter 7, distributed system concepts are used to deploy multiple independent DT applications and DS models. Kafka enables information sharing from a DS model to other DT components. This information-sharing system allows multiple DT applications to run concurrently using the information received from the same DS model.

9.3 Research hypotheses

During the research study, the research hypotheses presented in Sect. 1.5 are proved.

RH1 The case study in Sect. 6.6 proves loosely-coupled DMs capabilities to create a holistic view of the physical system using multiple models.

RH2 The experimental results of using Kafka shows its ability to provide low-latency communication, in milliseconds.

RH3 The case studies in Sect. 6.6, 7.6, and 8.4 show the potential of using Dockers to containerize DMs given their dependencies. This approach enables DT designers to use and deploy models using different modeling approaches and vendors.

RH4 K8s update policies enables runtime modification of DT components and tracking DM versions.

RH5 The design of DT applications as independent components, alongside the separation between DS models and DT applications, allows multiple decision-support applications and tools to be deployed concurrently as shown in the case studies in Sect. 7.6 and 8.4.

9.4 Summary

Distributed system concepts and Artificial intelligence (AI) applications are used in building a general-purpose Digital twin (DT) framework. The DT framework was developed over multiple phases. The framework's conceptual design was first defined. Then a DS framework is developed to generate a real-time digital representation of a physical system. The DS framework design is then expanded to become a DT framework integrating intelligent applications into industrial systems. Finally, a DT development methodology is proposed to define a step-wise process for building DTs. Multiple industrial use cases were studied during the different phases of this research. The research conclusions are as follows:

- DS and DT concepts are defined to separate between both concepts based on their roles and functionality. The proposed definition describes DS as a core component of a DT. This separation enables the system designers to identify the needed components for each concept. A DT framework conceptual design is built according to the proposed definition and functionality. The proposed DT framework design provides a wide range of DT applications to be applied in production systems.
- Distributed system concepts are used in developing a real-time DS framework. DS creates a digital representation of the physical system using multiple loosely-coupled Digital models (DMs). DS framework listens to state messages pulled from the physical system through web-based gateways. Pub/Sub communication system passes state messages between different Digital models (DMs). Containerization and orchestration integrate multiple DMs into DS pool.
- DS-based general-purpose DT framework is developed. DS provides a real-time replica for applying DT intelligence. Simulation models, data analysis tools, and Machine Learning (ML) models are used in implementing DT applications. Pattern recognition and statistical models generate data insights. Reinforcement Learning (RL) agents and virtual commissioning (VC) recommend actions to decision-makers.
- Agile-driven development methodology is proposed to define a systematic process for building DTs. The proposed methodology considers the deployment of enabling technologies into the targeted physical system. Stakeholders are involved in all phases of the DT development project. Adopting the agile development approach encourages the continuous improvement of the developed DS models and DT applications.

9.4.1 Implications

General-purpose DTs have the potential to expand DTs to different industrial domains. Integrating multiple simulation models into a multi-physics multi-scale representation allows the analysis of complex systems and processes in real-time. This allows business-level optimization of the production systems and testing different scenarios reducing the testing cost, risk of errors, and downtime.

DT market is already predicted to reach \$183 billion in revenue by 2031 [19]. The general-purpose DT framework enables businesses to build an ecosystem of simulation models, analysis tools, and decision-support applications for their systems. While standardizing the process of developing DTs improves the efficiency of these DTs and ease their integration and refinement. DT designers are able to iteratively improve DT applications based on feedback from the DT's stakeholders, connecting the DT development process with the end-users.

From the academic point of view, general-purpose DTs will ease the development of holistic system models. The DT framework tool-independency enables multiple models from different vendors to be deployed on the same architecture. Information sharing through Pub/Sub communication channel enables multiple models to co-simulate with a loosely-coupled interface. While multiple DT applications could run concurrently using the same information generated by DS models.

Modularization eases the reusability of DT components. Multiple DS models could use the same DMM configuration to connect to the same physical system. While a single source gateway could connect all the DS models and DT applications to the physical system. The reusability of different DT components reduces the time needed to develop new DT applications while enhancing their integrability.

Finally, adopting agile development in DT development methodology enables researchers to iteratively implement and improve their DTs. Agile development encourages cyclic implementation, testing, and feedback. Frequent feedback from DT stakeholders helps in identifying and addressing problems early. Agile development also allows multiple DT designers to collaborate on developing a DT through regular discussions, which keeps the stakeholders aligned with the DT project's scope.

9.5 Challenges and limitations

The use of distributed system concepts and technologies opens multiple opportunities and benefits. However, there are still some challenges and limitations to the proposed framework and its usage. The main challenge is the overhead complexity due to applying distributed

system concepts. DT designers need to have the prior technical expertise to design, set up, and maintain distributed systems. K8s clusters are created to host the DT framework on on-site servers or cloud platforms. While DS models and DT applications are dockerized before deploying them on the K8s cluster. Future studies will investigate the automation of DT deployment process in order to ease the development of DTs using the proposed DT framework. The automation of the K8s setup and DM configuration files generation would reduce the complexity overhead and improve the reusability of previous DTs. While the introduction of CD/CI tools would ease the deployment of updated DT versions and new DT applications.

DTs requires real-time connectivity to the targeted physical system in order to operate. The lack of benchmark physical systems and their specifications makes conducting empirical comparisons of DT frameworks difficult. Hence, it is unfeasible to evaluate the performance of different DT frameworks without implementing and deploying them on the same physical system. Rather than quantitatively compare the proposed framework's performance with existing frameworks in literature, the research work highlighted the feasibility, capabilities, and potential benefits of utilizing distributed system technologies. This approach elaborates on the proposed framework's strengths and benefits from a conceptual point of view. Through the support of Fondation Canadienne pour l'Innovation (FCI) and the collaboration of multiple universities in the *Intelligent Cyber Value Chain Network (CĒOSNet)*, the intelligent CPS warehouse system could be transformed into a DT benchmark physical system for future studies. Conducting comparative studies will be then feasible with the cooperation of involved research groups.

9.6 Future Research

The proposed DT framework's generality opens multiple possibilities to apply DT in different industrial fields. While further improvements to the DT framework are possible. The following points are future opportunities for improving and expanding the proposed DT framework:

- Human-in-the-loop Modeling

The industrial domain has focused on automating manufacturing processes to improve both productivity and efficiency. Automation doesn't necessarily replace human workers [148, 149]. Machinery is used where it is impossible or dangerous for human operators to work. Automating manufacturing processes increases efficiency and precision, reduces production time, and avoids human error.

Human operators can not be replaced in some operations. For example, human workers

can not be replaced in the poultry industry due to the slippery and non-rigid property of the product [150]. The high range of variability in product sizes makes precision positioning for automated processing hard. Poultry workers cut birds and remove bones since the development of robots' end-effectors for such tasks is difficult.

Efficiency-driven designs of production lines add more stress on human operators and increase workers' injury rates. Workers are required to keep up with the production line and robot speeds. Production line designers tend to define repetitive tasks to increase workers' efficiency. This increases muscle and tendon pain and reduces workers' movement. For example, poultry production workers develop painful injuries due to high conveyor speed, repetitive tasks, and static postures [151].

Human-in-the-loop is a modeling paradigm that considers human factors in model design [152]. Human operators are allowed to change the model's outputs in a human-in-the-loop system. Interactive simulation models, such as flight simulators and driving simulators, generate a virtual environment for human operators for training on different tasks. Such systems are also used to capture human behavior in complex situations for training AI controllers.

Human-in-the-loop paradigm introduces the consideration of human factors and knowledge in system design. Adopting the human-in-the-loop paradigm in designing DT applications would improve workers' safety and introduce new ideas for DT applications. Instead of full removal of human operators or manual control of machinery, a balanced human-machine interaction is desired [153]. In the AI domain, the human-in-the-loop paradigm utilizes human prior knowledge to improve ML models, especially in the medical domain [154]. Explainable AI adapts the human-in-the-loop paradigm in order to increase trust in AI and keep human operators involved in the decision process [155]. Explaining ML models decisions helps operators identify principal factors affecting the system. Additionally, expert operators are able to provide feedback to the ML model to improve and adjust their decisions.

- Fault Tolerance

Errors and faults are inevitable in any system. Multiple types of faults could occur in DTs such as physical machine faults, network faults, and simulation model faults. Physical machines fail to operate properly due to mechanical and electrical faults. The communication network fails to convey messages due to network congestion, high noise, and hardware faults. Simulation models fail due to model abstraction, unmodeled situations, and software faults.

Fault tolerance techniques should be adopted to ensure DT availability. For example, physical machine faults must be discovered to reflect the system status in the virtual representation. While faults in network messages and DMs should not affect the physical system or the decision-making process. Faults in DT models and applications should not affect the physical system or the decision-making process. A fail-safe mechanism needs to be provided to isolate DT model faults and ensure fast system recovery.

- Automatic detection of model deviation

Models are abstract descriptions of their real counterparts. Hence, model deviation due to abstraction errors is inevitable. Detection of model deviation is important for updating the designers' beliefs of the system behavior. In addition to detecting model deviation, automatic adjustment of model configuration is used to correct the model's outputs and reduce the need for designer intervention.

CHAPTER 10 CONCLUSION AND RECOMMENDATIONS

Applying AI and distributed system concepts enables the development of flexible scalable general-purpose DTs. While the distinction between DS and DT modularizes the DT framework's components. The realization of general-purpose DT eases the adaption of DT in a wide range of industrial domains. Continuous improvement of DTs is possible through the flexibility and scalability of the proposed DT framework. Multiple use cases are investigated throughout the research work to assess the proposed framework's feasibility and capabilities.

The proposed DT framework is designed using DS at its core. DS tracks the physical system information in order to generate a real-time digital replica of the system's state and behavior. Gateways capture the physical system status and sent it to DS. DMs then model the system behavior and generate a knowledge-graph representation of the system. Multi-physics multi-scale description of the system is created through DMs information aggregation. DT applications use the DS digital representation to generate insights for decision-makers. Recommendation systems suggest operational actions alongside their predicted consequences.

Distributed systems and AI are key enabling technologies for the proposed DT framework. Containerization allows DMs to run on any DT infrastructure. DT components are modularized using containerization and Microservices. Pub/Sub communication provides an information-sharing backbone to connect DT components. Orchestration enables the deployment of new models and applications during runtime without disrupting the DT operations. Simulation and AI technologies are used in developing DT applications and recommendation systems. These technologies provide diagnosis, prognosis, and virtual commissioning for decision-makers.

An agile-driven development methodology for DTs is introduced using the proposed DT framework. The development methodology involves stakeholders in DT design and testing gaining valuable feedback throughout the development process. Cyber-physical system (CPS) infrastructure and key enabling technologies are identified and applied for transforming the target physical system into a CPS. The agile development approach is adopted to allow fast deployment and continuous improvement of DS models and DT applications. DMs are implemented and validated before their deployment as DS models. DS models and DT applications are refined iteratively to reduce the simulation-to-reality gap and improve their performance.

REFERENCES

- [1] Cognite, “Open industrial data: Inspiring innovation & fueling collaboration in the oil & gas ecosystem,” Cognite, Tech. Rep., July 2018. [Online]. Available: <https://content.cognite.com/open-industrial-data>
- [2] A. AboElHassan and S. Yacout, “A digital shadow framework using distributed system concepts,” *Journal of Intelligent Manufacturing*, Sep 2022.
- [3] J. Moosavi, A. M. Fathollahi-Fard, and M. A. Dulebenets, “Supply chain disruption during the covid-19 pandemic: Recognizing potential disruption management strategies,” *International Journal of Disaster Risk Reduction*, vol. 75, p. 102983, 2022.
- [4] K. Karwasra, G. Soni, S. K. Mangla, and Y. Kazancoglu, “Assessing dairy supply chain vulnerability during the covid-19 pandemic,” *International Journal of Logistics Research and Applications*, vol. 0, no. 0, pp. 1–19, 2021.
- [5] M. Azizsafaei, A. Hosseinian-Far, R. Khandan, D. Sarwar, and A. Daneshkhah, “Assessing risks in dairy supply chain systems: A system dynamics approach,” *Systems*, vol. 10, no. 4, 2022.
- [6] H. Kagermann, W. Lukas, and W. Wahlster, “Industry 4.0: with the internet of things on the way to the 4th industrial revolution,” *VDI news*, vol. 13, 2011.
- [7] R. Stark, S. Kind, and S. Neumeyer, “Innovations in digital modelling for next generation manufacturing system design,” *CIRP Annals*, vol. 66, no. 1, pp. 169–172, 2017.
- [8] C. Cimino, E. Negri, and L. Fumagalli, “Review of digital twin applications in manufacturing,” *Computers in Industry*, vol. 113, p. 103130, dec 2019.
- [9] C. Monsone, E. Mercier-Laurent, and J. János, “The overview of digital twins in industry 4.0: Managing the whole ecosystem,” in *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. SCITEPRESS - Science and Technology Publications, 2019.
- [10] G. Yadav and K. Paul, “Architecture and security of scada systems: A review,” *International Journal of Critical Infrastructure Protection*, vol. 34, p. 100433, 2021.
- [11] W. Babel, *Automation Pyramid and Solutions Business*. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, pp. 75–147.

- [12] M. Sverko, T. G. Grbac, and M. Mikuc, “Scada systems with focus on continuous manufacturing and steel industry: A survey on architectures, standards, challenges and industry 5.0,” *IEEE Access*, vol. 10, pp. 109 395–109 430, 2022.
- [13] T. Ji, H. Huang, and X. Xu, “Digital twin technology — a bibliometric study of top research articles based on local citation score,” *Journal of Manufacturing Systems*, vol. 64, pp. 390–408, 2022.
- [14] J.-F. Uhlenkamp, K. Hribernik, S. Wellsandt, and K.-D. Thoben, “Digital twin applications : A first systemization of their dimensions,” in *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, jun 2019.
- [15] R. Stark and T. Damerau, “Digital twin,” in *CIRP Encyclopedia of Production Engineering*, S. Chatti and T. Tolio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 1–8.
- [16] M. Grieves and J. Vickers, “Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems,” in *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds. Cham: Springer International Publishing, 2017, pp. 85–113.
- [17] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, and L. Wang, “Draft modeling, simulation, information technology & processing roadmap,” *Technology Area 11*, 2010.
- [18] E. Glaessgen and D. 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, apr 2012.
- [19] A. Velosa and P. Middleton, “Emerging technologies: Revenue opportunity projection of digital twins,” Gartner, Tech. Rep., 2022.
- [20] R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen, “About the importance of autonomy and digital twins for the future of manufacturing,” *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 567–572, 2015.
- [21] A. Sharma, E. Kosasih, J. Zhang, A. Brintrup, and A. Calinescu, “Digital twins: State of the art theory and practice, challenges, and open research questions,” *Journal of Industrial Information Integration*, vol. 30, p. 100383, 2022.

- [22] J. Jósvali, A. Pfeiffer, N. Szántó, and G. Monek, “The role of digital twin in a cyber-physical production environment with prescriptive learning,” in *17th International Conference on Modeling and Applied Simulation, MAS 2018*, 2018, pp. 180–184. [Online]. Available: <http://eprints.sztaki.hu/id/eprint/9592>
- [23] Y. Lu, C. Liu, K. I.-K. Wang, H. Huang, and X. Xu, “Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues,” *Robotics and Computer-Integrated Manufacturing*, vol. 61, p. 101837, 2020.
- [24] C. Qian, X. Liu, C. Ripley, M. Qian, F. Liang, and W. Yu, “Digital twin—cyber replica of physical things: Architecture, applications and future research directions,” *Future Internet*, vol. 14, no. 2, 2022.
- [25] H. Park, A. Easwaran, and S. Andalam, “TiLA: Twin-in-the-loop architecture for cyber-physical production systems,” in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, nov 2019.
- [26] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, “Digital twin-driven product design, manufacturing and service with big data,” *The International Journal of Advanced Manufacturing Technology*, vol. 94, no. 9-12, pp. 3563–3576, mar 2017.
- [27] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, and A. Nee, “Enabling technologies and tools for digital twin,” *Journal of Manufacturing Systems*, vol. 58, pp. 3–21, 2021.
- [28] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, A. Rettberg, and C. E. Pereira, “Digital twin connectivity topologies,” *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 737–742, 2021, 17th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2021.
- [29] J. Trauer, S. Schweigert-Recksiek, C. Engel, K. Spreitzer, and M. Zimmermann, “What is a digital twin? – definitions and insights from an industrial case study in technical product development,” *Proceedings of the Design Society: DESIGN Conference*, vol. 1, p. 757–766, 2020.
- [30] M. Riesener, G. Schuh, C. Dölle, and C. Tönnies, “The digital shadow as enabler for data analytics in product life cycle management,” *Procedia CIRP*, vol. 80, pp. 729–734, 2019.

- [31] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, “A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics,” *Sensors*, vol. 21, no. 19, 2021.
- [32] S. M. E. Sepasgozar, “Differentiating digital twin from digital shadow: Elucidating a paradigm shift to expedite a smart, sustainable built environment,” *Buildings*, vol. 11, no. 4, 2021.
- [33] R. Paredis and H. Vangheluwe, “Exploring a digital shadow design workflow by means of a line following robot use-case,” in *2021 Annual Modeling and Simulation Conference (ANNSIM)*, 2021, pp. 1–12.
- [34] A. Phua, C. Davies, and G. Delaney, “A digital twin hierarchy for metal additive manufacturing,” *Computers in Industry*, vol. 140, p. 103667, 2022.
- [35] M. van Steen and A. S. Tanenbaum, *Distributed systems*, 3rd ed. distributed-systems.net, 2017.
- [36] Y. Xing and Y. Zhan, “Virtualization and cloud computing,” in *Future Wireless Networks and Information Systems*, Y. Zhang, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 305–312.
- [37] N. Jain and S. Choudhary, “Overview of virtualization in cloud computing,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1–4.
- [38] A. Bhardwaj and C. R. Krishna, “Virtualization in cloud computing: Moving from hypervisor to containerization—a survey,” *Arabian Journal for Science and Engineering*, vol. 46, no. 9, pp. 8585–8601, Sep 2021.
- [39] R. Keller Tesser and E. Borin, “Containers in hpc: a survey,” *The Journal of Supercomputing*, vol. 79, no. 5, pp. 5759–5827, Mar 2023.
- [40] M. P. Papazoglou and W.-J. van den Heuvel, “Service oriented architectures: approaches, technologies and research issues,” *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, Jul 2007.
- [41] J.-P. Gouigoux and D. Tamzalit, “From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture,” in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 62–65.
- [42] P. N. Stuart Russell, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2021.

- [43] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Springer New York, NY, 2006.
- [44] D. W. Cearley, “Top 10 strategic technology trends for 2019,” Gartner, Tech. Rep., 2018. [Online]. Available: <https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/>
- [45] F. Ameri and R. Sabbagh, “Digital factories for capability modeling and visualization,” in *IFIP Advances in Information and Communication Technology*. Springer International Publishing, 2016, pp. 69–78.
- [46] S. Yun, J.-H. Park, and W.-T. Kim, “Data-centric middleware based digital twin platform for dependable cyber-physical systems,” in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, jul 2017.
- [47] K. J. Kuehner, R. Scheer, and S. Strassburger, “Digital twin: Finding common ground – a meta-review,” *Procedia CIRP*, vol. 104, pp. 1227–1232, 2021.
- [48] F. Pires, V. Melo, J. Almeida, and P. Leitão, “Digital twin experiments focusing virtualisation, connectivity and real-time monitoring,” in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, 2020, pp. 309–314.
- [49] R. Williams, J. A. Erkoyuncu, T. Masood, and R. Vrabic, “Augmented reality assisted calibration of digital twins of mobile robots,” *IFAC-PapersOnLine*, vol. 53, no. 3, pp. 203–208, 2020.
- [50] Z. Jiang, Y. Guo, and Z. Wang, “Digital twin to improve the virtual-real integration of industrial iot,” *Journal of Industrial Information Integration*, vol. 22, p. 100196, 2021.
- [51] P. Brauner, M. Dalibor, M. Jarke, I. Kunze, I. Koren, G. Lakemeyer, M. Liebenberg, J. Michael, J. Pennekamp, C. Quix, B. Rumpe, W. van der Aalst, K. Wehrle, A. Wortmann, and M. Ziefle, “A computer science perspective on digital transformation in production,” *ACM Transactions on Internet of Things*, vol. 3, no. 2, pp. 1–32, feb 2022.
- [52] G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, “Digital twin data modeling with AutomationML and a communication methodology for data exchange,” *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 12–17, 2016.
- [53] C. Weber, J. Königsberger, L. Kassner, and B. Mitschang, “M2ddm – a maturity model for data-driven manufacturing,” *Procedia CIRP*, vol. 63, pp. 173–178, 2017.

- [54] F. Biesinger, D. Meike, B. Kraß, and M. Weyrich, “A digital twin for production planning based on cyber-physical systems: A case study for a cyber-physical system-based creation of a digital twin,” *Procedia CIRP*, vol. 79, pp. 355–360, 2019.
- [55] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, and F. Sapio, “A conceptual architecture and model for smart manufacturing relying on service-based digital twins,” in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, jul 2019.
- [56] M. Redeker, J. N. Weskamp, B. Rössl, and F. Pethig, “Towards a digital twin platform for industrie 4.0,” in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021, pp. 39–46.
- [57] K. T. Park, S. H. Lee, and S. D. Noh, “Information fusion and systematic logic library-generation methods for self-configuration of autonomous digital twin,” *Journal of Intelligent Manufacturing*, Jun 2021.
- [58] A. Göppert, L. Grahn, J. Rachner, D. Grunert, S. Hort, and R. H. Schmitt, “Pipeline for ontology-based modeling and automated deployment of digital twins for planning and control of manufacturing systems,” *Journal of Intelligent Manufacturing*, Nov 2021.
- [59] “Automation systems and integration — digital twin framework for manufacturing,” International Organization for Standardization, Geneva, CH, Standard, Oct. 2021.
- [60] “Industrial-process measurement, control and automation - digital factory framework,” International Electrotechnical Commission, Geneva, CH, Standard, Oct. 2020.
- [61] G. Shao, “Use case scenarios for digital twin implementation based on iso 23247,” 2021-05-04 04:05:00 2021.
- [62] C. Zhuang, J. Liu, and H. Xiong, “Digital twin-based smart production management and control framework for the complex product assembly shop-floor,” *The International Journal of Advanced Manufacturing Technology*, vol. 96, no. 1-4, pp. 1149–1163, feb 2018.
- [63] Z. Zhu, X. Xi, X. Xu, and Y. Cai, “Digital twin-driven machining process for thin-walled part manufacturing,” *Journal of Manufacturing Systems*, vol. 59, pp. 453–466, 2021.
- [64] S. Martinez, A. Mariño, S. Sanchez, A. M. Montes, J. M. Triana, G. Barbieri, S. Abolghasem, J. Vera, and M. Guevara, “A digital twin demonstrator to enable flexible

- manufacturing with robotics: a process supervision case study,” *Production & Manufacturing Research*, vol. 9, no. 1, pp. 140–156, 2021.
- [65] J.-G. Duan, T.-Y. Ma, Q.-L. Zhang, Z. Liu, and J.-Y. Qin, “Design and application of digital twin system for the blade-rotor test rig,” *Journal of Intelligent Manufacturing*, Aug 2021.
- [66] B. A. Talkhestani and M. Weyrich, “Digital twin of manufacturing systems: a case study on increasing the efficiency of reconfiguration,” *at - Automatisierungstechnik*, vol. 68, no. 6, pp. 435–444, 2020.
- [67] K. Tliba, T. M. L. Diallo, O. Penas, R. Ben Khalifa, N. Ben Yahia, and J.-Y. Choley, “Digital twin-driven dynamic scheduling of a hybrid flow shop,” *Journal of Intelligent Manufacturing*, Mar 2022.
- [68] K. Mykoniatis and G. A. Harris, “A digital twin emulator of a modular production system using a data-driven hybrid modeling and simulation approach,” *Journal of Intelligent Manufacturing*, vol. 32, no. 7, pp. 1899–1911, Oct 2021.
- [69] F. Tao, Q. Qi, L. Wang, and A. Nee, “Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison,” *Engineering*, vol. 5, no. 4, pp. 653–661, aug 2019.
- [70] F. Tao and M. Zhang, “Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing,” *IEEE Access*, vol. 5, pp. 20 418–20 427, 2017.
- [71] Y. Qamsane, C.-Y. Chen, E. C. Balta, B.-C. Kao, S. Mohan, J. Moyne, D. Tilbury, and K. Barton, “A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems,” in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, aug 2019.
- [72] F. Becker, P. Bibow, M. Dalibor, A. Gannouni, V. Hahn, C. Hopmann, M. Jarke, I. Koren, M. Kröger, J. Lipp, J. Maibaum, J. Michael, B. Rumpe, P. Sapel, N. Schäfer, G. J. Schmitz, G. Schuh, and A. Wortmann, “A conceptual model for digital shadows in industry and its application,” in *Conceptual Modeling*, ser. Lecture Notes in Computer Science (LNCS), A. Ghose, J. Horkoff, V. E. Silva Souza, J. Parsons, and J. Evermann, Eds., vol. 13011. Cham: Springer International Publishing, 2021, pp. 271–281.
- [73] A. Ladj, Z. Wang, O. Meski, F. Belkadi, M. Ritou, and C. Da Cunha, “A knowledge-based digital shadow for machining industry in a digital twin perspective,” *Journal of Manufacturing Systems*, vol. 58, pp. 168–179, 2021.

- [74] C. Brecher, M. Buchsbaum, A. Müller, K. Schilling, M. Obdenbusch, S. Staudacher, and M. C. Albasatineh, “Gaining iiot insights by leveraging ontology-based modelling of raw data and digital shadows,” in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021, pp. 231–236.
- [75] G. Schuh, C. Kelzenberg, J. Wiese, and T. Ochel, “Data structure of the digital shadow for systematic knowledge management systems in single and small batch production,” *Procedia CIRP*, vol. 84, pp. 1094–1100, 2019.
- [76] S. Konstantinov, F. Assad, W. Azam, D. Vera, B. Ahmad, and R. Harrison, “Developing web-based digital twin of assembly lines for industrial cyber-physical systems,” in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021, pp. 219–224.
- [77] S. Kassen, H. Tammen, M. Zarte, and A. Pechmann, “Concept and case study for a generic simulation as a digital shadow to be used for production optimisation,” *Processes*, vol. 9, no. 8, 2021.
- [78] J. Dobaj, M. Krisper, and G. Macher, “Towards cyber-physical infrastructure as-a-service (CPIaaS) in the era of industry 4.0,” in *Communications in Computer and Information Science*. Springer International Publishing, 2019, pp. 310–321.
- [79] V. Kamath, J. Morgan, and M. I. Ali, “Industrial iot and digital twins for a smart factory : An open source toolkit for application design and benchmarking,” in *2020 Global Internet of Things Summit (GIoTTS)*, 2020, pp. 1–6.
- [80] X. Tong, Q. Liu, S. Pi, and Y. Xiao, “Real-time machining data application and service based on imt digital twin,” *Journal of Intelligent Manufacturing*, vol. 31, no. 5, pp. 1113–1132, Jun 2020.
- [81] C. E. B. López, “Real-time event-based platform for the development of digital twin applications,” *The International Journal of Advanced Manufacturing Technology*, vol. 116, no. 3, pp. 835–845, Sep 2021.
- [82] A. B. A. Alaasam, G. Radchenko, and A. Tchernykh, “Stateful stream processing for digital twins: Microservice-based kafka stream dsl,” in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2019, pp. 0804–0809.

- [83] G. Radchenko, A. B. Alaasam, and A. Tchernykh, “Micro-workflows: Kafka and kepler fusion to support digital twins of industrial processes,” in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 83–88.
- [84] G. Steindl and W. Kastner, “Semantic microservice framework for digital twins,” *Applied Sciences*, vol. 11, no. 12, 2021.
- [85] R. Wagner, B. Schleich, B. Haefner, A. Kuhnle, S. Wartzack, and G. Lanza, “Challenges and potentials of digital twins and industry 4.0 in product design and production for high performance products,” *Procedia CIRP*, vol. 84, pp. 88–93, 2019.
- [86] Y. Lu and X. Xu, “Resource virtualization: A core technology for developing cyber-physical production systems,” *Journal of Manufacturing Systems*, vol. 47, pp. 128–140, apr 2018.
- [87] M. Eckhart and A. Ekelhart, “A specification-based state replication approach for digital twins,” in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and Privacy - CPS-SPC’18*. ACM Press, 2018.
- [88] R. Wagner, B. Schleich, B. Haefner, A. Kuhnle, S. Wartzack, and G. Lanza, “Challenges and potentials of digital twins and industry 4.0 in product design and production for high performance products,” *Procedia CIRP*, vol. 84, pp. 88–93, 2019, 29th CIRP Design Conference 2019, 08-10 May 2019, Póvoa de Varzim, Portugal.
- [89] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, “Review of job shop scheduling research and its new perspectives under industry 4.0,” *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.
- [90] W.-K. Jung, H. Kim, Y.-C. Park, J.-W. Lee, and E. S. Suh, “Real-time data-driven discrete-event simulation for garment production lines,” *Production Planning & Control*, pp. 1–12, 2020.
- [91] G. S. Fishman, *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.
- [92] Y. Carson and A. Maria, “Simulation optimization: methods and applications,” in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 118–126.
- [93] C. Liu, J. Ding, and J. Sun, “Reinforcement learning based decision making of operational indices in process industry under changing environment,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2727–2736, 2021.

- [94] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [95] C. M. Macal and M. J. North, “Tutorial on agent-based modelling and simulation,” *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.
- [96] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, “Intelligent scheduling of discrete automated production line via deep reinforcement learning,” *International Journal of Production Research*, vol. 58, no. 11, pp. 3362–3380, 2020.
- [97] A. Greasley, “Implementing reinforcement learning in simio discrete-event simulation software,” in *Proceedings of the 2020 Summer Simulation Conference*, ser. SummerSim ’20. San Diego, CA, USA: Society for Computer Simulation International, 2020.
- [98] L. Capocchi, J. Santucci, and B. P. Zeigler, “Discrete event modeling and simulation aspects to improve machine learning systems,” in *2018 4th International Conference on Universal Village (UV)*, 2018, pp. 1–6.
- [99] T. Ushio and T. Yamasaki, “Supervisory control of partially observed discrete event systems based on a reinforcement learning,” in *SMC’03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, vol. 3, 2003, pp. 2956–2961.
- [100] C. Lopez, J. R. Marti, and S. Sarkaria, “Distributed reinforcement learning in emergency response simulation,” *IEEE Access*, vol. 6, pp. 67 261–67 276, 2018.
- [101] T. Altenmüller, T. Stüker, B. Waschneck, A. Kuhnle, and G. Lanza, “Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints,” *Production Engineering*, vol. 14, no. 3, p. 319–328, 2020.
- [102] M. Rabe, F. Dross, A. Wuttke, A. Duarte, A. Juan, and H. Lourenço, “Combining a discrete-event simulation model of a logistics network with deep reinforcement learning,” in *Metaheuristics: Proceedings of the MIC and MAEB 2017 Conference*, 2017, pp. 765–774.
- [103] A. AboElHassan, A. Sakr, and S. Yacout, “A framework for digital twin deployment in production systems,” in *Advances in Automotive Production Technology – Theory and Application*, P. Weißgraeber, F. Heieck, and C. Ackermann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 145–152.

- [104] M. Kunath and H. Winkler, “Integrating the digital twin of the manufacturing system into a decision support system for improving the order management process,” *Procedia CIRP*, vol. 72, pp. 225–231, 2018.
- [105] B. Leng, H. Sun, G. Si, T. Xia, and H. Wang, “Digital twin and manufacturing simulation integrated platform embedded in cyber-physical system,” *Journal of Physics: Conference Series*, vol. 1983, no. 1, p. 012117, jul 2021.
- [106] A. J. H. Redelinghuys, A. H. Basson, and K. Kruger, “A six-layer architecture for the digital twin: a manufacturing case study implementation,” *Journal of Intelligent Manufacturing*, vol. 31, no. 6, pp. 1383–1402, Aug 2020.
- [107] K. Alexopoulos, N. Nikolakis, and G. Chryssolouris, “Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing,” *International Journal of Computer Integrated Manufacturing*, vol. 33, no. 5, pp. 429–439, 2020.
- [108] F. Hu, X. Qiu, G. Jing, J. Tang, and Y. Zhu, “Digital twin-based decision making paradigm of raise boring method,” *Journal of Intelligent Manufacturing*, Apr 2022.
- [109] H. Zhang, Q. Qi, and F. Tao, “A multi-scale modeling method for digital twin shop-floor,” *Journal of Manufacturing Systems*, vol. 62, pp. 417–428, 2022.
- [110] C. Zhuang, J. Gong, and J. Liu, “Digital twin-based assembly data management and process traceability for complex products,” *Journal of Manufacturing Systems*, vol. 58, pp. 118–131, 2021.
- [111] G. P. Tancredi, G. Vignali, and E. Bottani, “Integration of digital twin, machine-learning and industry 4.0 tools for anomaly detection: An application to a food plant,” *Sensors*, vol. 22, no. 11, 2022.
- [112] S. Liu, X. V. Wang, and L. Wang, “Digital twin-enabled advance execution for human-robot collaborative assembly,” *CIRP Annals*, vol. 71, no. 1, pp. 25–28, 2022.
- [113] L. Stan, A. F. Nicolescu, C. Pupăză, and G. Jiga, “Digital twin and web services for robotic deburring in intelligent manufacturing,” *Journal of Intelligent Manufacturing*, May 2022.
- [114] M. Groshev, C. Guimarães, J. Martín-Pérez, and A. de la Oliva, “Toward intelligent cyber-physical systems: Digital twin meets artificial intelligence,” *IEEE Communications Magazine*, vol. 59, no. 8, pp. 14–20, 2021.

- [115] A. Borghesi, G. Di Modica, P. Bellavista, V. Gowtham, A. Willner, D. Nehls, F. Kintzler, S. Cejka, S. R. Tisbeni, A. Costantini, M. Galletti, M. Antonacci, and J. C. Ahouangonou, "Iotwins: Design and implementation of a platform for the management of digital twins in industrial scenarios," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 625–633.
- [116] M. J. Sax, "Apache kafka," in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Zomaya, Eds. Cham: Springer International Publishing, 2018, pp. 1–8.
- [117] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.
- [118] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Transactions on Internet Technology*, vol. 20, no. 2, apr 2020.
- [119] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "A kubernetes controller for managing the availability of elastic microservice based stateful applications," *Journal of Systems and Software*, vol. 175, p. 110924, 2021.
- [120] M. de Castro-Cros, M. Velasco, and C. Angulo, "Machine-learning-based condition assessment of gas turbines—a review," *Energies*, vol. 14, p. 8468, 2021.
- [121] K. H. Lüdtke, *Process Centrifugal Compressors*, 1st ed. Springer, Berlin, Heidelberg, 2004.
- [122] L. A. Kareem, T. M. Iwalewa, and M. Al-Marhoun, "New explicit correlation for the compressibility factor of natural gas: linearized z-factor isotherms," *Journal of Petroleum Exploration and Production Technology*, vol. 6, no. 3, pp. 481–492, Sep 2016.
- [123] H. Boyes and T. Watson, "Digital twins: An analysis framework and open issues," *Computers in Industry*, vol. 143, p. 103763, 2022.
- [124] S. Reed, M. Löfstrand, and J. Andrews, "Modelling cycle for simulation digital twins," *Manufacturing Letters*, vol. 28, pp. 54–58, 2021.
- [125] J. Moyne, Y. Qamsane, E. C. Balta, I. Kovalenko, J. Faris, K. Barton, and D. M. Tilbury, "A requirements driven digital twin framework: Specification and opportunities," *IEEE Access*, vol. 8, pp. 107 781–107 801, 2020.

- [126] Y. Qamsane, J. Moyne, M. Toothman, I. Kovalenko, E. C. Balta, J. Faris, D. M. Tilbury, and K. Barton, “A methodology to develop and implement digital twin solutions for manufacturing systems,” *IEEE Access*, vol. 9, pp. 44 247–44 265, 2021.
- [127] T. Moreno, A. Almeida, F. Ferreira, N. Caldas, C. Toscano, and A. Azevedo, “Digital twin for manufacturing equipment in industry 4.0,” *Advances in Manufacturing Technology XXXIV*, vol. 15, pp. 362–367, 2021.
- [128] F. Wilking, C. Sauer, B. Schleich, and S. Wartzack, “Sysml 4 digital twins – utilization of system models for the design and operation of digital twins,” *Proceedings of the Design Society*, vol. 2, p. 1815–1824, 2022.
- [129] V. Havard, B. Jeanne, M. Lacomblez, and D. Baudry, “Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations,” *Production & Manufacturing Research*, vol. 7, no. 1, pp. 472–489, 2019.
- [130] A. H. Sakr, A. Aboelhassan, S. Yacout, and S. Bassetto, “Simulation and deep reinforcement learning for adaptive dispatching in semiconductor manufacturing systems,” *Journal of Intelligent Manufacturing*, Oct 2021.
- [131] S. Konstantinov, F. Assad, B. Ahmad, D. A. Vera, and R. Harrison, “Virtual engineering and commissioning to support the lifecycle of a manufacturing assembly system,” *Machines*, vol. 10, no. 10, 2022.
- [132] Kronos, “The future of manufacturing: 2020 and beyond,” Tech. Rep., 2016.
- [133] X. Lai, H. Shui, D. Ding, and J. Ni, “Data-driven dynamic bottleneck detection in complex manufacturing systems,” *Journal of Manufacturing Systems*, vol. 60, pp. 662–675, 2021.
- [134] D. Guo, S. Ling, Y. Rong, and G. Q. Huang, “Towards synchronization-oriented manufacturing planning and control for industry 4.0 and beyond,” *IFAC-PapersOnLine*, vol. 55, no. 2, pp. 163–168, 2022, 14th IFAC Workshop on Intelligent Manufacturing Systems IMS 2022.
- [135] J. Wang, Y. Liu, S. Ren, C. Wang, and S. Ma, “Edge computing-based real-time scheduling for digital twin flexible job shop with variable time window,” *Robotics and Computer-Integrated Manufacturing*, vol. 79, p. 102435, 2023.
- [136] B. Marr, “The 10 biggest future trends in manufacturing,” *Forbes*, 2022. [Online]. Available: <https://www.forbes.com/sites/bernardmarr/2022/01/25/the-10-biggest-future-trends-in-manufacturing/>

- [137] M. Kumbhar, A. H. Ng, and S. Bandaru, “A digital twin based framework for detection, diagnosis, and improvement of throughput bottlenecks,” *Journal of Manufacturing Systems*, vol. 66, pp. 92–106, 2023.
- [138] A. AboElHassan, A. Sakr, and S. Yacout, “General purpose digital twin framework using digital shadow and distributed system concepts,” *Computers & Industrial Engineering*, 2023, under review.
- [139] D. M. Botín-Sanabria, A.-S. Mihaita, R. E. Peimbert-García, M. A. Ramírez-Moreno, R. A. Ramírez-Mendoza, and J. d. J. Lozoya-Santos, “Digital twin technology challenges and applications: A comprehensive review,” *Remote Sensing*, vol. 14, no. 6, 2022.
- [140] M. Perno, L. Hvam, and A. Haug, “Implementation of digital twins in the process industry: A systematic literature review of enablers and barriers,” *Computers in Industry*, vol. 134, p. 103558, 2022.
- [141] G. Barbieri, A. Bertuzzi, A. Capriotti, L. Ragazzini, D. Gutierrez, E. Negri, and L. Fumagalli, “A virtual commissioning based methodology to integrate digital twins into manufacturing systems,” *Production Engineering*, vol. 15, no. 3, pp. 397–412, Jun 2021.
- [142] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, and Z. Hu, “A comprehensive review of digital twin — part 1: modeling and twinning enabling technologies,” *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, p. 354, Nov 2022.
- [143] A. M. Kuhn, M. C. May, Y. Liu, A. Kuhnle, W. Tekouo, and G. Lanza, “Towards narrowing the reality gap in electromechanical systems: error modeling in virtual commissioning,” *Production Engineering*, Sep 2022.
- [144] A. Chakroun, Y. Hani, A. Elmhamedi, and F. Masmoudi, “Digital transformation process of a mechanical parts production workshop to fulfil the requirements of industry 4.0,” in *2022 14th International Colloquium of Logistics and Supply Chain Management (LOGISTIQUA)*, 2022, pp. 1–6.
- [145] “Industrial communication networks - profiles - part 2: Additional fieldbus profiles for real-time networks based on iso/iec/ieee 8802-3,” International Electrotechnical Commission, Geneva, CH, Standard, Oct. 2019.

- [146] M. Dangana, S. Ansari, S. M. Asad, S. Hussain, and M. A. Imran, “Towards the digital twin (dt) of narrow-band internet of things (nbiot) wireless communication in industrial indoor environment,” *Sensors*, vol. 22, no. 23, 2022.
- [147] S. Robinson, *Simulation: The Practice of Model Development and Use*, 2nd ed. Palgrave Macmillan, 2014.
- [148] S. Y. Nof, “Automation: What it means to us around the world,” in *Springer Handbook of Automation*, S. Y. Nof, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 13–52.
- [149] I. Graessler and A. Poehler, “Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system,” in *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2017, pp. 289–293.
- [150] G. McMurray, “14 - robotics and automation in the poultry industry: current technology and future trends,” in *Robotics and Automation in the Food Industry*, ser. Woodhead Publishing Series in Food Science, Technology and Nutrition, D. G. Caldwell, Ed. Woodhead Publishing, 2013, pp. 329–353.
- [151] J. G. Ramsey, K. Musolin, C. Mueller, and N. 2015, “Health hazard evaluation report: Evaluation of carpal tunnel syndrome and other musculoskeletal disorders among employees at a poultry processing plant. noish hhe report no. 2014-0040-3232,” U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, National Institute for Occupational Safety and Health, Tech. Rep., 2015.
- [152] S. Narayanan and P. Kidambi, “Interactive simulations: History, features, and trends,” in *Human-in-the-Loop Simulations: Methods and Practice*, L. Rothrock and S. Narayanan, Eds. London: Springer London, 2011, pp. 1–13.
- [153] W. Li, D. Sadigh, S. S. Sastry, and S. A. Seshia, “Synthesis for human-in-the-loop control systems,” in *Tools and Algorithms for the Construction and Analysis of Systems*, E. Ábrahám and K. Havelund, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 470–484.
- [154] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He, “A survey of human-in-the-loop for machine learning,” *Future Generation Computer Systems*, vol. 135, pp. 364–381, 2022.

- [155] F. M. Zanzotto, “Viewpoint: Human-in-the-loop artificial intelligence,” *Journal of Artificial Intelligence Research*, vol. 64, pp. 243–252, 2019.