FIRST – Project Number: 734599

H2020-MSC-RISE-2016 Ref. 6742023

Interoperability Framework of Virtual Factory and Business Innovation

Lead Editor: Michel Medema, University of Groningen

With contributions from: Lai Xu and Paul de Vrieze from Bournemouth University; Yuewei Bai and Shuangyu Wei from Shanghai Polytechnic University; Massimo Mecella, Flavia Monti and Francesco Leotta from Sapienza University of Rome; Georg Soppa, Lucas Fucke and Norbert Eder from GK Software; and Heerko Groefsema, Mostafa Hadadian and Majid Lotfian Delouee from the University of Groningen.

Deliverable nature	Report
Dissemination level	Public
Contractual delivery date	31st of October 2022
Actual delivery date	22nd of December 2022
Version	1.0
Keywords	Interoperability, Digital Twins, Virtual Factories, Smart Manufacturing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734599
H2020-MSC-RISE-2016 Ref. 6742023

Table of Contents

1 Introduction

Virtual factories are abstractions of real factories that provide a multi-layered integration of the information related to various activities along the factory and the product lifecycle. The physical equipment of a factory is represented in the virtual factory, connected to the physical equipment of the other companies and can receive data and instructions. These virtual environments allow for the simulation and analysis of various production processes and systems and can be used for a variety of purposes, including production planning, process optimisation, training, and prototyping. Virtual factories rely on a combination of advanced digital technologies, such as Artificial Intelligence, the Internet of Things, and big data analytics, to optimise manufacturing processes in order to increase productivity and quality.

However, the implementation of virtual factories and Industry 4.0 technologies also presents a number of challenges, one of which is ensuring interoperability between the vast number of different systems and technologies that are required to realise the full potential of virtual factories and Industry 4.0. Ensuring interoperability in smart factories is critical because it enables different systems and technologies to exchange and process data effectively, allowing for the smooth operation of the factory as a whole. Without interoperability, different systems and technologies may be unable to communicate and exchange data, leading to inefficiencies and potential errors. This deliverable discusses several challenges related to interoperability in the context of virtual factories and smart manufacturing that need to be addressed in order to enable the communication among heterogeneous software components of the virtual factory, as well as the exploitation of services and data according to business process objectives.

Section 2 explores the use of Manufacturing Execution Systems for building an industrial software layer for virtual factories. The virtual factory system is focused on the management of manufacturing assets. As a result, the production workshops of a virtual factory require a full-featured, well-interactive Manufacturing Execution System to solve tasks such as dispatching manufacturing parts and materials and managing production resources. The Manufacturing Execution System is one of the indispensable systems for building an industrial software layer of such smart factories and the virtual production systems. It plays a bridge role between many different parts of the entire manufacturing system.

Section 3 discusses how digital twins, which build upon Internet of Things concepts, can be leveraged to simulate complex systems with diverse components whose simulation incorporates approaches from multiple disciplines. Such simulations are desired in many advanced systems, including modern approaches to manufacturing. Section 4 explores the automatic composition of services provided by digital twins, which resembles the composition of Web services. Since a single manufacturing process may comprise hundreds of different digital twins that may suddenly fail or provide bad performance, the manufacturing process should be able to adapt to new conditions automatically, considering new digital twins for the fulfilment of the manufacturing goals. This task cannot be performed manually when actors span multiple organisations that are possibly separated from both the geographical and organisational points of view, which is why it is crucial to have a plan for the manufacturing process to be able to manage several digital twins, taking into account their possible failures and costs.

Section 5 considers the use of verification techniques to understand the processes of virtual factories, verify their correctness and diagnose potential problems. Such verification techniques allow us to check whether a process *conforms* or *complies* with some specification, but the terms conformance and compliance are often used as synonyms. As a result, the terminology used to describe the techniques or the corresponding verification activity does not always match the precise meaning of the terms as they are defined in the area of verification, and the confusion of these terms may hamper the application of the different techniques in smart factories.

In Section 6, the issue of interoperability at the development level is considered. Nowadays, traditional software development methodologies are no longer sufficient for fulfilling business requirements. The implementation of Continuous Integration and Continuous Delivery, CICD, has enabled fast delivery of software and increased productivity. A considerable benefit of having a CICD pipeline is a separation of responsibilities that will help team members to focus on their part while the CICD pipeline takes care of integration and delivery, which results in rapid releases.

Section 7 presents Complex Event Processing as a means to provide stronger interoperability between virtual factories and the Internet of Things, one of the key technologies on which virtual factories rely. Complex Event Processing can not only be used to obtain valuable insights from the data obtained from the Internet of Things devices but also to perform real-time analysis to allow preventative actions. An important trade-off for such systems is between the quality of the results obtained from the analysis and the level of privacy that the system guarantees.

2 Interoperation and its Implementation of MES to Support Virtual Factory

The VF (virtual factory) technology can effectively be used to control product manufacturing quality, business collaboration efficiency, demand response speed, and reduce manufacturing costs on a large-scale customised collaborative business chain (Wei, Bai, and Xu 2020a). The collaborative manufacturing industry chain links based on the VF framework are composed of manufacturers or service providers with the best comprehensive competitiveness. Therefore, the establishment of a VF system is the key to achieving manufacturing business innovation. The VF is an upgrade of the digital factory and it has more connotations and functions. Because, from one hand, it is based on the automation of workshop equipment and applies technologies such as the Industrial Internet of Things to realise automatic data collection and information integration of the physical resources of the manufacturing system; on the other hand, it can use the CPS (Cyber Physics System) to construct a digital twin model to realize the control of the virtual and real two ways of the manufacturing system. In short, the VF platform/system is focused on the manufacturing assets management. As a result, the production workshops of a VF should need a full-featured, well-interactive MES (Manufacturing Execution System) to solve the tasks such as manufacturing parts/materials dispatch, production resource management etc. Otherwise, it is difficult to map the manufacturing assets of the virtual factory to the actual production tasks and realise data integration. From this perspective, the MES interoperability of the workshop is the key to the operation of the entire VF system.

The MES interoperability framework supporting VF operation plays a bridge role in many links such as product design, manufacturing process planning, production process management, and quality inspection. It organises the manufacturing assets, schedules production cycles, and optimises the entire production systems by integrating the useful data sources from different information systems. Therefore, the interoperability of MES in the VF environment is the key link of whether the VF system can operate smoothly and effectively. However, the existing MES integration framework lacks a mechanism to integrate the VF platform. Consequently, the MES is necessary to adopt an improvement development manner for this purpose, yet it is found that there are rarely applicable researches in this field. Therefore, it becomes the motivation of the research on MES interoperability to support manufacturing business innovation.

The research objective is to explore a way of how to improve the existing MES systems functions and their application ranges via integration strategies. It can effectively integrate distributed manufacturing resources (machine tools, equipment, robots, stackers, operators, etc.) and provide corresponding MES that meets the requirements of the industrial software layer in an intelligent software system. Among them, the following two specific questions need to be addressed.

1) The MES uses what method to collect and process of the real-time operating data of the distributed workshop manufacturing assets of VF reliably. By acquiring these data, the MES system can provide the shop floor production management staff with reliable data required for manufacturing resource planning and scheduling.
2) According to the integrated distributed manufacturing assets and their operating data, how the MES uses the information to evaluate the feasibility of the initially formed production plan and manufacturing resource scheduling plan of VF? That is, the production plan established by MES according to the product production process, and the corresponding machine tools, equipment, robots, operators, etc., whether the combination of these resources is optimal which can be evaluated by using some software tools, e.g., ProModel, Flexsim, etc. As a result, there is a problem with how to integrate the MES and the simulation software tools combined the distributed manufacturing assets. Then, MES can use the results of the assessment to decide whether to adjust these production plans and manufacturing resource scheduling plans.

2.1 MES Interoperability Framework Integrated with VF Platform

The VF is the key to achieving large-scale customised services in Industry 4.0. The establishment of a VF can focus on the manufacturers in the advantageous industrial chain to form a dynamic production system with high-level, reliable production and transparent management.

According to the foregoing, combined with the VF production system performance evaluation technology based on VF manufacturing assets vertical integration method, and the cloud manufacturing model based on VF manufacturing assets horizontal integration technology, the existing MES can be extended to support the integration with virtual factories platform (refer to Figure 1).

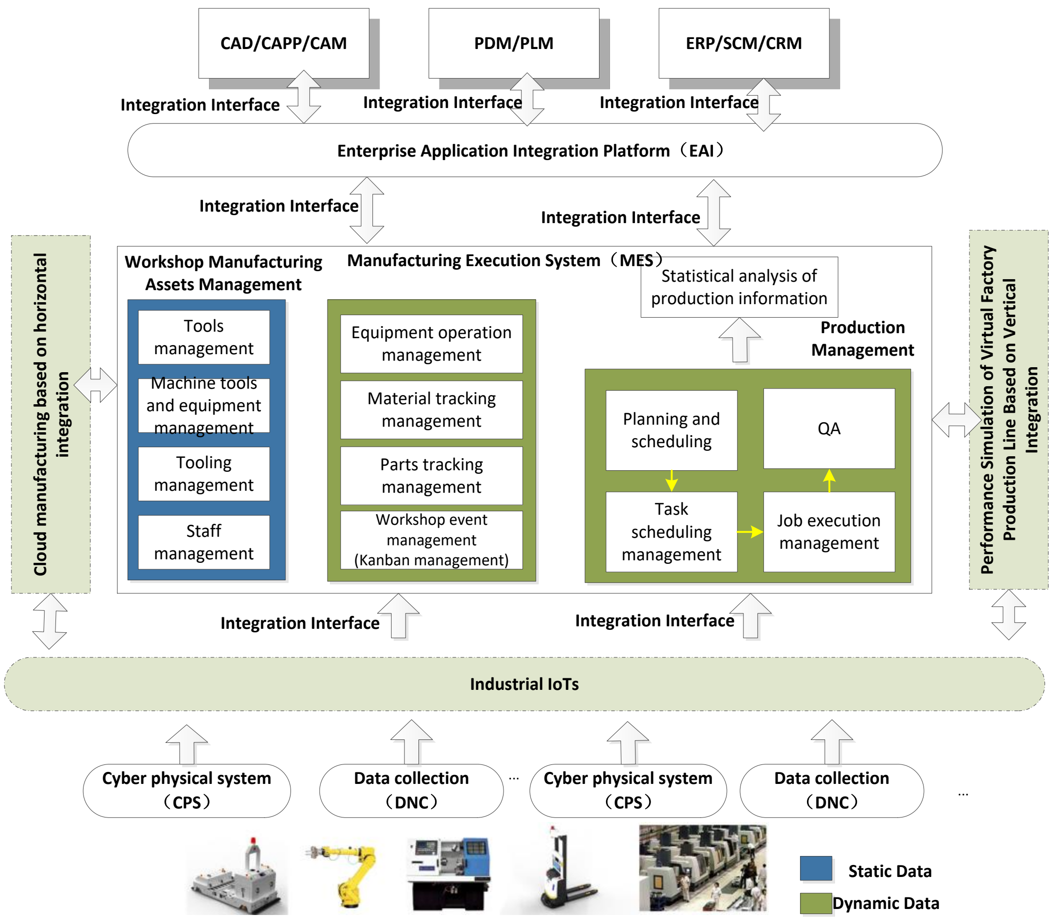

CAD/CAPP/CAM
PDM/PLM
ERP/SCM/CRM
Integration Interface
Integration Interface
Integration Interface
Enterprise Application Integration Platform（EAI）
Integration Interface
Integration Interface
Cloud manufacturing based on horizontal integration
Workshop Manufacturing Assets Management
Manufacturing Execution System（MES）
Statistical analysis of production information
Production Management
Tools management
Machine tools and equipment management
Tooling management
Staff management
Equipment operation management
Material tracking management
Parts tracking management
Workshop event management (Kanban management)
Planning and scheduling
QA
Task scheduling management
Job execution management
Performance Simulation of Virtual Factory Production Line Based on Vertical Integration
Integration Interface
Integration Interface
Industrial IoTs
Cyber physical system（CPS）
Data collection（DNC）
Cyber physical system（CPS）
Data collection（DNC）
Static Data
Dynamic Data

Figure 1: MES interoperability framework supporting VF applications.

In terms of vertical integration method, the VF uses the information model to interconnect and interoperate with equipment via the standards/protocols, e.g., MTConnect, AutomationML, and OPC-UA (Unified Architecture), to read the running data of the equipment layer in the workshop in real-time. The data from the IoT platform provides a reliable input for the evaluation of the operating effectiveness and efficiency of a VF manufacturing system. By integrating MES into the VF platform, on the one hand, the evaluation results can be utilised to provide a basis for the MES production scheduling and task scheduling of the VF, on the other hand, it can also be used as an auxiliary to dynamically build or improve the performance of the VF production system.

Using VF horizontal integration technology, it can integrate the distributed manufacturing assets and then establish a cloud manufacturing model. In addition to evaluating the performance of the collaborative manufacturing production line, the VF integration platform also manages product manufacturing business activities and related information in the VF environment. The IDEF (Integrated Definition Methodology) can be used for platform modelling. The platform supports multiple levels and multi-view integration of the manufacturing assets throughout the product life cycle and establishes a real and virtual digital twin model of mutual mapping. The MES can more reasonably perform production management, scheduling, and task scheduling, and optimize the performance of job tasks than the past, by integrating the digital twin model.

2.1.1 Case Study

A large domestic ship manufacture group that has tens of subordinate manufacturing plants and research institutes has formed an industrial chain that is similar to a VF. It has applied the intelligent manufacturing systems developed by KM-Soft Co since 2006. Recently, the group started the construction of the National Intelligent Manufacturing Demonstration Project, called Digital Workshop of Shipbuilding Engineering Mechanical and Electrical Equipment, which focused on the construction of the KM-MES based on an integration platform of the manufacturing assets from the distributed subordinate manufacturing enterprises (refer to Figure 2). Therefore, this study takes the case as an example to explain the availability of the aforementioned MES interoperability framework model which can be used to support VF operation.

First, the bottom layer is the VF platform. It uses MTConnect (MTConnect Institute 2018), AutomationML (Yemenicioğlu and Lüder 2014), OPC-OA and other protocols to collect hardware data, such as machine tools, robots, and transportation equipment of the companies on the manufacturing chain. Through CPS/DNC and other methods, the dynamic data of the hardware equipment is transmitted to the IoT database. The vertically integrated subsystem for the distributed manufacturing assets can be used for the VF production line evaluation purpose by combining with the production order information from the VF; the horizontally integrated subsystem of the distributed manufacturing assets can be established to provide services such as manufacturing assets discovery and optimised combination in the form of cloud manufacturing.

Second, it is the MES system layer. By integrating with the VF platform, it can optimise the establishment of the marine power propeller product manufacturing chain and form an optimised VF production line by utilising the manufacturing assets discovery and combination services provided by the cloud manufacturing subsystem. Also, it can integrate the subsystem which vertically integrates the manufacturing resources based on the VF platform, and to evaluate the performance of the established VF production line. Furthermore, it can continuously improve the performance of the VF production line.

The third layer is the industrial software layer. This layer is composed of CAD/CAE/CAX, the systems of PDM/PLM, the process design and management system, and the enterprise manufacturing assets management ERP/CRM/SCM. A production database of the VF is required at this layer to support the operation of the entire VF information system. The MES on the second layer can integrate the industrial software system on the layer through the enterprise application integration platform to realise the functions of production planning, organisation, and manufacturing resource management in the workshop. The top layer is the enterprise user layer. This layer uses real-time data from the VF database to mine the manufacturing data. It can help to provide users with multi-view and multi-dimensional data mining results by BI technology, which assists corporate decision-making quickly and reliably. The other functions, e.g., browse and query personalised data via a graphical interface, are also provided.

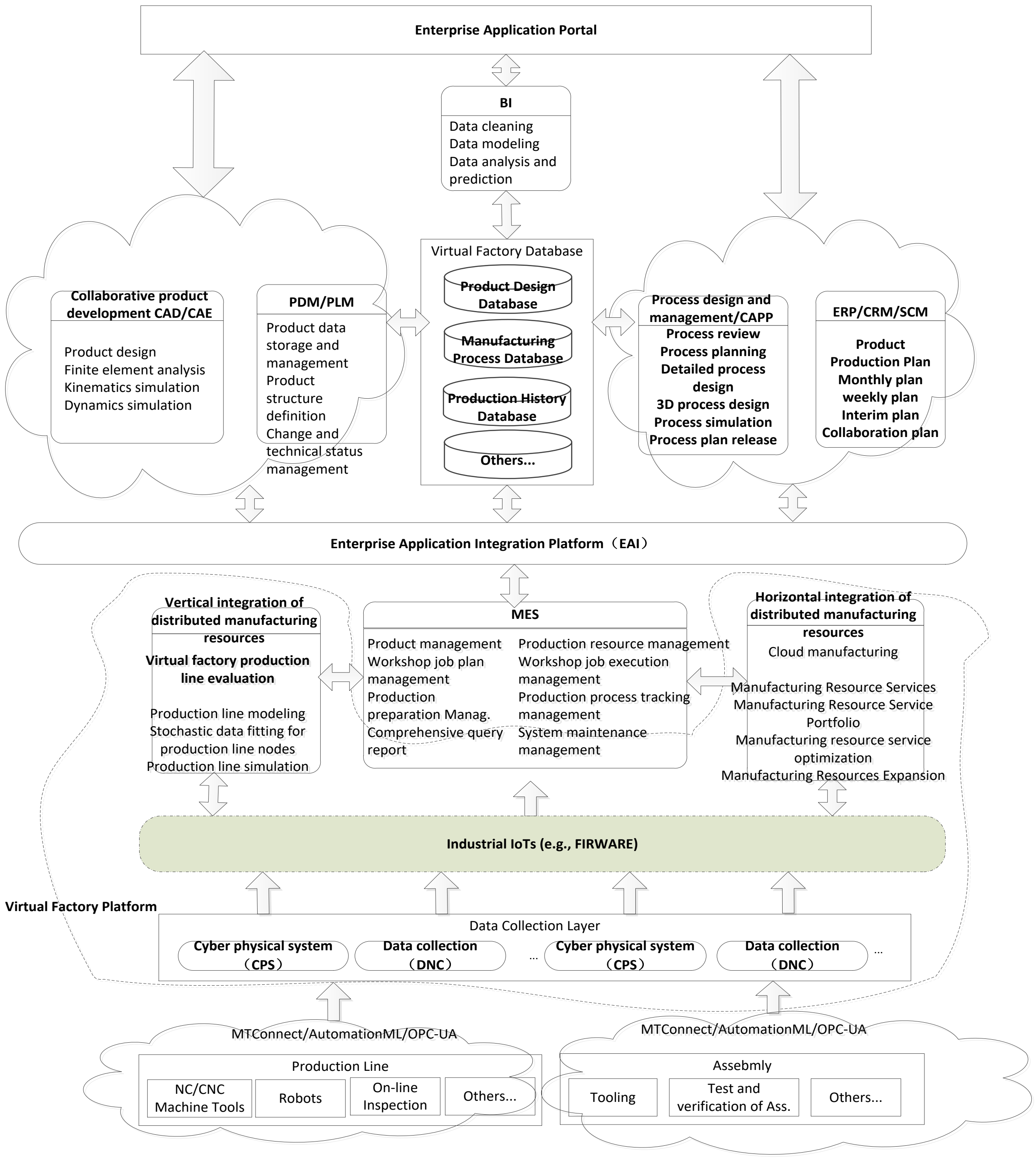

Enterprise Application Portal
BI
Data cleaning
Data modeling
Data analysis and prediction
Collaborative product development CAD/CAE
Product design
Finite element analysis
Kinematics simulation
Dynamics simulation
PDM/PLM
Product data storage and management
Product structure definition
Change and technical status management
Virtual Factory Database
Product Design Database
Manufacturing Process Database
Production History Database
Others...
Process design and management/CAPP
Process review
Process planning
Detailed process design
3D process design
Process simulation
Process plan release
ERP/CRM/SCM
Product
Production Plan
Monthly plan
weekly plan
Interim plan
Collaboration plan
Enterprise Application Integration Platform（EAI）
Vertical integration of distributed manufacturing resources
Virtual factory production line evaluation
Production line modeling
Stochastic data fitting for production line nodes
Production line simulation
MES
Product management
Workshop job plan management
Production preparation Manag.
Comprehensive query report
Production resource management
Workshop job execution management
Production process tracking management
System maintenance management
Horizontal integration of distributed manufacturing resources
Cloud manufacturing
Manufacturing Resource Services
Manufacturing Resource Service Portfolio
Manufacturing resource service optimization
Manufacturing Resources Expansion
Industrial IoTs (e.g., FIRWARE)
Virtual Factory Platform
Data Collection Layer
Cyber physical system（CPS）
Data collection（DNC）
...
Cyber physical system（CPS）
Data collection（DNC）
...
MTConnect/AutomationML/OPC-UA
Production Line
NC/CNC Machine Tools
Robots
On-line Inspection
Others...
MTConnect/AutomationML/OPC-UA
Assebmly
Tooling
Test and verification of Ass.
Others...

Figure 2: A case of virtual manufacturing platform between MES and VF.

2.2 Quality Management

Quality management is a crucial activity in discrete manufacturing (Wei, Bai, and Xu 2020b). One of the primary functions of the MES in the workshop is to provide related management functions for quality management. Quality management must not only collect the processing quality data of parts but also carry out relevant quality data analysis to find the causes of quality problems and channels and improvement methods to improve product quality in the product design, product manufacturing, and maintenance of production machine tools, etc. The analysed result can be used in different production stages to improve and enhance the quality of these activities and ultimately achieve continuous product quality improvement.

CMM (Coordinate Measurement Machine) is one of the main methods of quality inspection widely used in the manufacturing workshop. Enterprises use CMM inspection results to provide users with reliable parts manufacturing quality reports. Data analysis can provide necessary data for equipment maintenance management, manufacturing process route improvement. The reasonable use of CMM to ensure the manufacturing system quality can help to continue producing high-quality parts. As a result, CMM plays a vital role in product quality management, and it has become one of the hot issues in manufacturing technology (Mears et al. 2009). Recently, CMM technology and measurement software technology has developed rapidly, making the parts inspection process more automated. Because the technology provides the more possibility of using the inspection data, the manufacturing quality management module of MES helps improve the production process. It provides more great conditions for it (Machado et al. 2020).

2.2.1 Motivation

There is usually a long delay between manufacturing, CMM inspection, and part process evaluation. When the production system has problems meeting the design or process parameters requirements, the quality assurance team has to identify and correct it as soon as possible through quality management mechanisms. Researching the CMM data interoperation mechanism is conducive to solving three critical problems that are ubiquitous in intelligent manufacturing systems and urgently need to be addressed. (1) For the maintenance of crucial manufacturing machine tools, it is vital to establish the prediction models of machine tool accuracy and cutting tool parameter by applying big data and artificial intelligence technology via linking CMM measurement data and the use data of the cutting tools through MES. It can help carry out predictive maintenance, avoid the economic losses caused by improper maintenance schedule or the critical equipment shutting down, and reduce the maintenance cost. (2) Correlate the CMM measurement data through the MES, associate the processing data of the geometric features of the part with the process model, analyse the correlation between the process parameters and the actual part processing quality data, to improve the processing process and process parameters. (3) Through the MES correlation CMM measurement data, the geometric feature processing data is associated with the design model to realise the design-manufacturing data model associated maintenance, integrate processing cost and other factors, and improve the design model.

The format and content of the measurement report provided by the CMM software are fixed (i.e., pre-defined). If the data and evaluation items required for subsequent quality management cannot be obtained directly from the report, it should find other methods, such as manual calculation. Therefore, to support the above three key issues, it is necessary to establish a CMM data interoperation mechanism. When the measurement software provides the APIs, the original measurement data can be accessed during the measuring by calling them, then calculates the required dimensional tolerances and geometric tolerances of the specific features with the data via calling the geometric fitting algorithms module. It is only carried out when the data analysis for quality management is a need that cannot be gotten directly from the measurement report, and it does not aim to replace the CMM measurement reports but is just a supplement. The whole process is automatically done without any interference or man-machine interaction, and thus it can avoid the human error. The other way to access the original measurement data is to use the DMIS program produced by CMM software. The method of processing the obtained information is the same as above mentioned.

2.2.2 Requirements

We use a UML USER-CASE diagram to describe the requirements of the CMM data interoperation layer. In the CMM interoperation scenario, the roles involved are CMM operator, CMM data user, MES system user, CAD/CAPP designer, and workshop equipment, a maintenance engineer. The activities related to each role are described below in Figure 3.

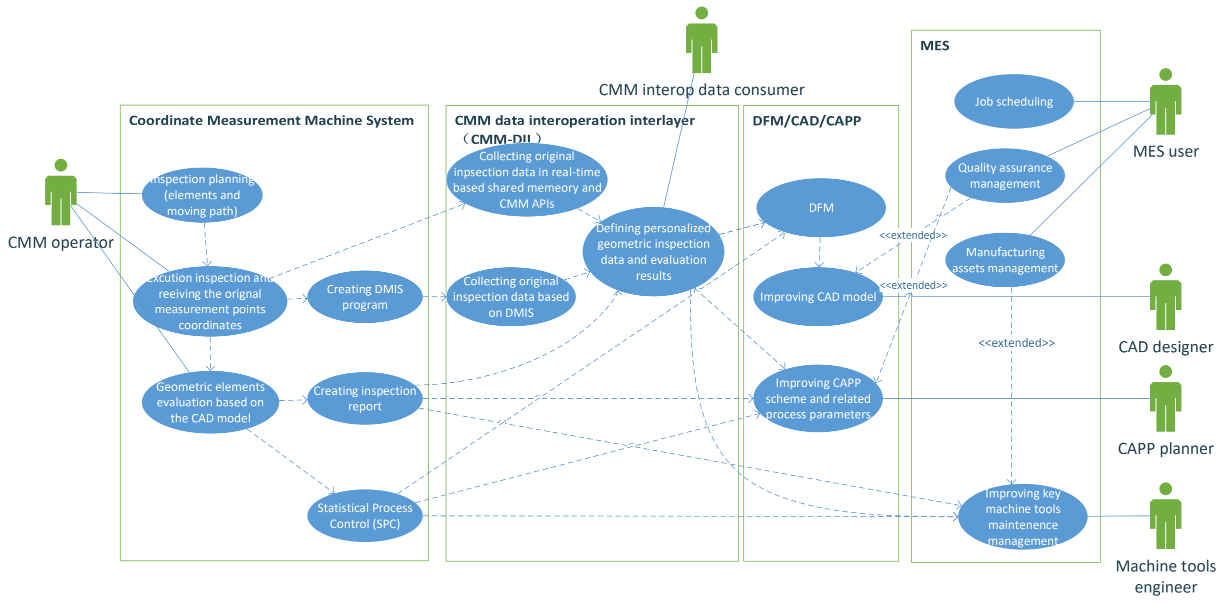

CMM interop data consumer
Coordinate Measurement Machine System
CMM data interoperation interlayer （CMM-DIL）
DFM/CAD/CAPP
MES
CMM operator
Inspection planning (elements and moving path)
Excution inspection and reeiving the orignal measurement points coordinates
Creating DMIS program
Geometric elements evaluation based on the CAD model
Creating inspection report
Statistical Process Control (SPC)
Collecting original inpsection data in real-time based shared memeory and CMM APIs
Collecting original inspection data based on DMIS
Defining personalized geometric inspection data and evaluation results
DFM
Improving CAD model
Improving CAPP scheme and related process parameters
<<extended>>
<<extended>>
<<extended>>
Job scheduling
Quality assurance management
Manufacturing assets management
Improving key machine tools maintenence management
MES user
CAD designer
CAPP planner
Machine tools engineer

Figure 3: CMM interoperation use-case diagram.

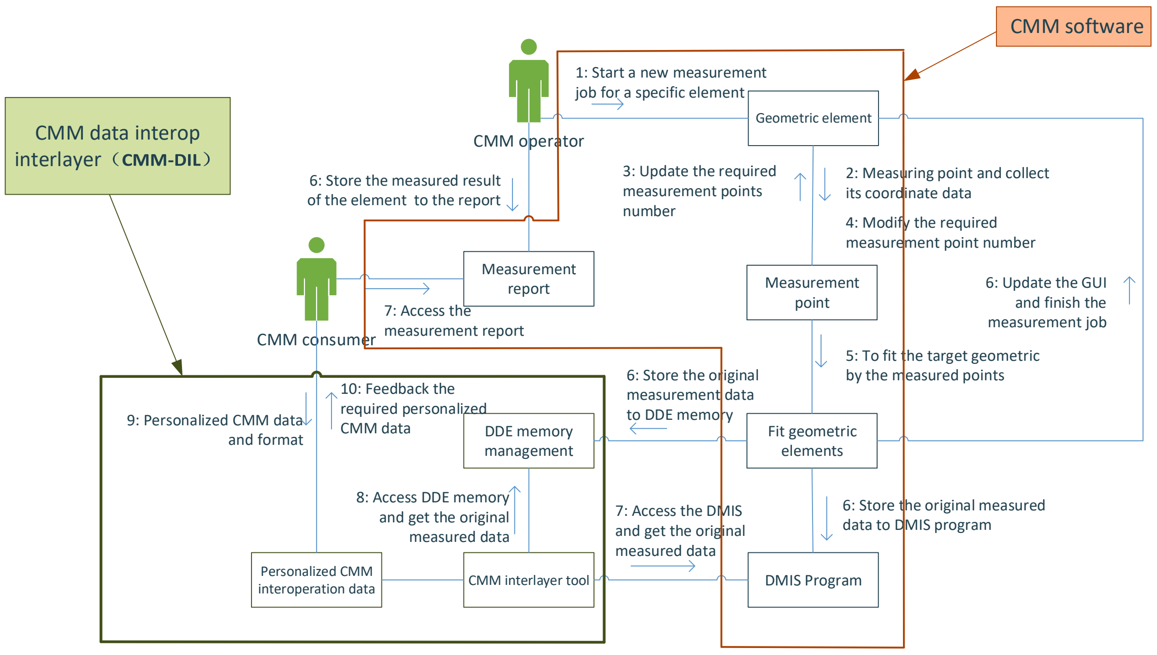
CMM operator initiates the activities, which involves measurement moving path planning, the implementation of online/offline inspection, error evaluation for the inspected parts based CAD model. When performing an online/offline check, the CMM software records the DMIS automatically that stores the commands and original data in the measurement process according to the DMIS specifications to form a DMIS file. It is designed for professional users to programming the expected inspection scheme. In contrast, the result is output to the report by the CMM software for the general users when the machining error evaluation is done based on the CAD model, which is also the content of most current research on CMM data interoperation.

CMM data users call the APIs of CMM-DIL (Data Interoperation Layer) interface to generate the additional geometric elements from the original measurement data (e.g., the axis of a cylinder, the center of a circle, or the vector of the plane from a circle element), which are not included in the CMM output inspection report but are needed for quality management, CAD/CAPP improvement, etc. These data can be output to the DFM (Design for Manufacturing) system as essential information to improve the related CAD model; CAPP can use the data to verify if it needs to improve the planned operation steps; the data also can be used by the equipment management team to analyse the operating status of critical equipment in a workshop.

Also, the MES production quality manager adopts statistical analysis methods to report the quality problems found to the relevant departments. For example, the product designers and manufacturing process planners improve the CAD/CAPP models via the personalised geometric evaluation result supported by the CMM-DIL. The workshop equipment maintenance engineer can formulate a reasonable equipment maintenance plan or improve the existing maintenance plan based on the quality report of the quality manager of the workshop via MES.

2.2.3 CMM-DIL Developments

To the problem of full use of CMM measurement information, the key is to design a reasonable CMM-DIL module to obtain the CMM original measurement commands and data reliably in real-time. In this manner, the information of the CMM system can be organised as a service for the users who require it by calling the personalised data definition tool provided by CMM-DIL. The following is an analysis of the critical activities of the CMM raw data acquisition process (see Figure 4).



CMM software
CMM data interop interlayer（CMM-DIL）
CMM operator
1: Start a new measurement job for a specific element
Geometric element
6: Store the measured result of the element to the report
3: Update the required measurement points number
2: Measuring point and collect its coordinate data
4: Modify the required measurement point number
CMM consumer
Measurement report
7: Access the measurement report
Measurement point
6: Update the GUI and finish the measurement job
5: To fit the target geometric by the measured points
6: Store the original measurement data to DDE memory
9: Personalized CMM data and format
10: Feedback the required personalized CMM data
DDE memory management
Fit geometric elements
8: Access DDE memory and get the original measured data
7: Access the DMIS and get the original measured data
6: Store the original measured data to DMIS program
Personalized CMM interoperation data
CMM interlayer tool
DMIS Program

Figure 4: The raw data acquisition logic versus the key activities.

Activity 1: For a specific element required to inspect, the CMM operator runs an online/offline measurement task through human-computer interaction. Online mode is for a real inspection, and offline is for DMIS programming. If it did automatically, the DMIS program (complete or partial) run could be managed by the CMM software. The measurement commands and data are sent to MDE (Measurement Device Equipment) one by one according to the I++ protocol, and the MDE will guide the measurement machine to perform the inspections.

Activity 2: After receiving the measurement command, MDE drives the probe, touches the surface of the part, and triggers the measurement signal. The motion controller latches the measurement point data, and then reliably feeds it back to the CMM software according to the I++ protocol. The CMM measurement software will save the current measurement point data.

Activity 3: After the CMM GUI receives the measurement points feedback from the motion controller, the number of measurement points on the measurement dialog interface starts to count in reverse order (n=n-1). If the number of measurement points required at this time is zero, the measurement software will automatically jump to activity five and complete the current measurement; otherwise, it will wait for the next measurement point of the MDE.

Activity 4: CMM operators can increase or decrease the current number of measurement points in the measurement dialog box through human-computer interaction.

Activity 5: After the measurement software obtains the current measurement point, the reverse count is zero. The measurement software will automatically finish the current measurement task, perform geometric fitting based on the acquired measurement point, and perform error calculations based on the theoretical value and the actual measurement value.

Activity 6: After the geometric element is fitted in measurement software, the original measurement data (including measurement commands and actual measurement point data) are transferred to the DDE memory through the APIs of the CMM software.

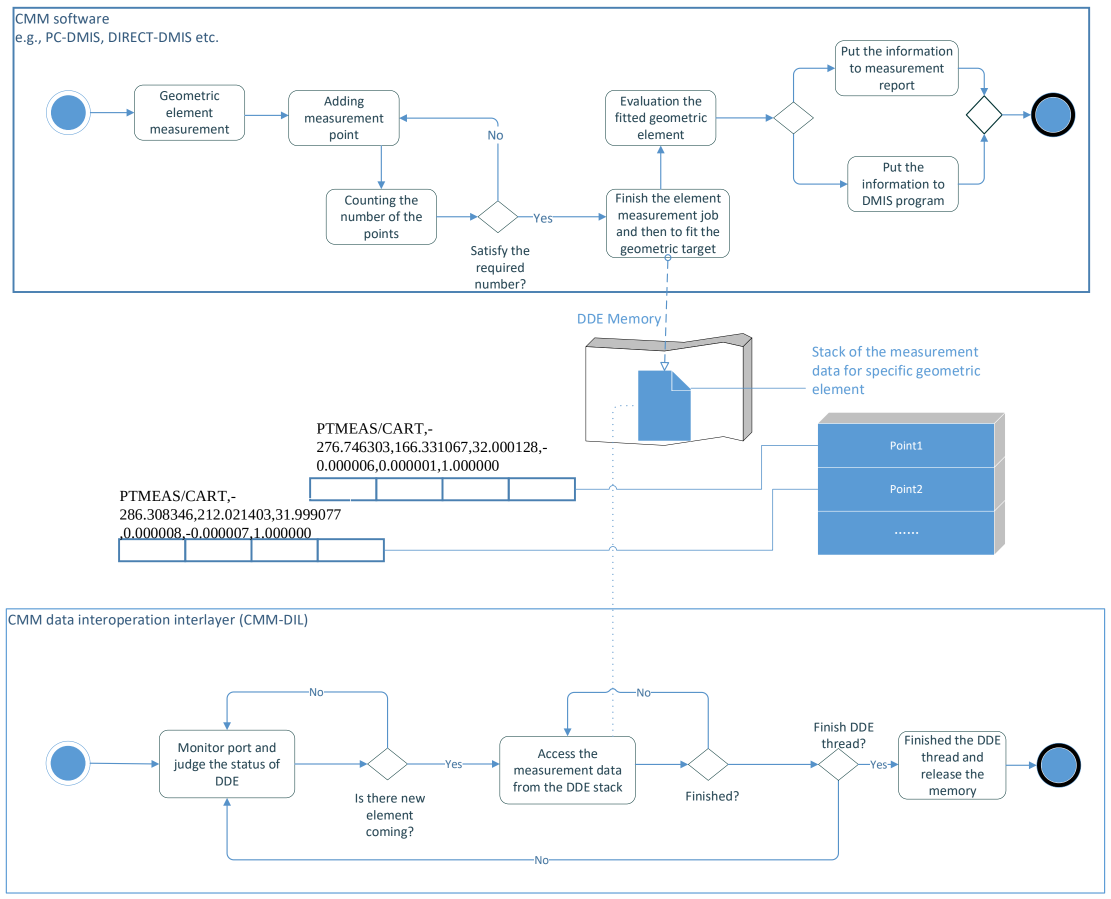
Activity 7: CMM-DIL can access the DMIS program of the measurement software through the DMIS interface and obtain the current measurement object of the measurement system and its original measurement data by analysing DMIS.

Activity 8: Use the APIs to access the DDE memory, obtain the current measurement object and original data, and save it in the personalised format.

Activity 9: Customise the required measurement data through the GUI, and then store the current measurement raw data in the DDE memory obtained by CMM-DIL in a predefined format.

Activity 10: According to the items subscribed in Activity 9, the geometric element evaluation result by the CMM-DIL is automatically obtained.

The following study of the CMM-DIL module logic design is given below through the above discussion and analysis of the raw measurement data acquisition activity.



CMM software
e.g., PC-DMIS, DIRECT-DMIS etc.
Geometric element measurement
Adding measurement point
No
Counting the number of the points
Yes
Satisfy the required number?
Finish the element measurement job and then to fit the geometric target
Evaluation the fitted geometric element
Put the information to measurement report
Put the information to DMIS program
DDE Memory
Stack of the measurement data for specific geometric element
PTMEAS/CART,-276.746303,166.331067,32.000128,-0.000006,0.000001,1.000000
PTMEAS/CART,-286.308346,212.021403,31.999077,0.000008,-0.000007,1.000000
Point1
Point2
......
CMM data interoperation interlayer (CMM-DIL)
Monitor port and judge the status of DDE
No
Is there new element coming?
Yes
Access the measurement data from the DDE stack
No
Finished?
Finish DDE thread?
Yes
Finished the DDE thread and release the memory
No

Figure 5: Flow of CMM-DIL access the measurement raw data via DDE memory.

At the end of the element measurement, the measurement software writes the measure raw data into the DDE memory according to a particular standard format, and then the CMM-DIL monitors the changes in the DDE memory data and calls API to access the measure raw data in the DDE memory. Therefore, the process for CMM-DIL to access these data can be summarised as follows.

First of all, the CMM-DIL monitors the data changes in the DDE memory through the API of the CMM software. When the CMM software finishes a measurement task and writes the original data into the DDE memory, the CMM-DIL can detect this change; then, the CMM-DIL creates a temporary measurement original object according to the read measurement command and later measures the original one by one. After reading a complete measure task and its unique data, the CMM-DIL will end the current data reading activity (refer to Figure 5).

2.3 Conclusion

Through the integration of the VF platform, its vertical integration (production line performance evaluation) and horizontal integration technology (cloud manufacturing) are taken as two aspects. (1) MES can effectively manage the distributed manufacturing resources of the VF by integration with the virtual manufacturing assets discovery, combination, and management services. (2) Combining the production plan information of ERP/MES, it can be used to evaluate the performance of the VF production lines, to continuously optimise and improve the performance of VF production lines, and to realise manufacturing business innovation.

Another important function of MES is to provide related management functions for quality management, for which CMM is generally one of the main method. The CMM measurement report (format and content) provided by the CMM software is usually fixed. If the required data used for quality analysis cannot be obtained directly from the report, it should be solved in other ways, e.g., manual calculation, but it may introduce calculation errors. This research aims at the quality management of critical business activities in the discrete manufacturing industry. By collecting real-time measurement raw data from the CMM software, it provides more abundant inspection data for quality analysis. This research aims not to replace the measurement results/reports of the CMM software, but to provide a lower-level and more comprehensive real-time data for quality analysis in MES to support better quality analysis and improvement based on the measurement data analysis.

3 Interoperable Collaborative Manufacturing Process Simulation for Digital Twins

Digital twins are a key concept, building upon Internet of Things concepts, in many advanced systems, including modern approaches to manufacturing. As a concept, a digital twin provides a digital representation of a physical twin (Jones et al. 2020), allowing for digital interaction with the physical twin, enhanced access to its properties and simulation of the twin in future or speculative contexts (Schluse et al. 2018). As such, the digital twin concept combines adaptive modelling with simulation and gains additional capabilities from this combination.

The physical entities represented by digital twins do not exist in isolation but are part of larger systems and processes (configurations). These configurations can also be represented in a configuration of digital twins that extends the advantages of digital twins to the larger context.

These configurations could themselves be fully functional (composite) digital twins and part of a layered configuration or hierarchy of digital twins. In this context, where the higher-level digital twins are used to represent (including to validate) an entire hierarchy, this implies that the physical counterparts (or the operational aspects of the digital twins) are interoperable.

The constituent elements (digital twins) must communicate using some protocol with transport and application layers for the configuration to exist (even without any digital twins). The simulation aspect of the digital twins must also be interoperable but could take advantage of the existing interoperability of the physical counterparts, ideally retaining the application layer protocol and only replacing the transport layer.

As done in (Platenius-Mohr et al. 2020), interoperability for digital twins can be understood on the basis of IEC21823-1 (International Electrotechnical Commission 2019). IEC21823-1 recognises five interoperability aspects: Transport, Syntactic, Semantic, Behavioural and Policy interoperability. Most work looks at syntactic, semantic and behavioural interoperability. When looking at simulation of interconnected configurations of physical/digital twin pairs, interoperability outside the simulation context (which is to be simulated) addresses most of these issues, with the exception of transport (and in some parts syntactic) interoperability.

In the context of Industry 4.0, it is desired to simulate complex systems with diverse components whose simulation incorporates approaches from multiple disciplines. Many digital twins represent extensive investment in their development and may have confidential aspects. In addition, there may be specific simulators for specialist equipment, needing to be integrated in the simulation of the surrounding manufacturing and business processes. While it is possible to reduce the behaviour of components to probability distributions, this requires additional work, loses precision and cannot account for interactions between different process steps.

For example, in the context of industry 4.0, where multiple organisations are collaborating with each other to achieve a common objective, different operations such as manufacturing, supply chain, logistics and services are handled by different partners. In this case, multiple simulation components for each of the operations present a more reliable and achievable solution as compared to a monolithic simulation. Interactions and data sharing of these components as desired by the partners can be facilitated. The combination of these components is envisaged as a complete Digital Twin of the whole collaboration.

The alternative to using probability distributions for overall simulations is to use more specific simulation models (the simulation aspect of the digital twins) for the components. One approach to solve this is co-simulation (Gomes et al. 2018), federation is a variation of this where the component simulators are not required to be run on the same system or in the same process, and certainly not in the same simulator (to execute the simulation model).

Co-simulation and federated simulation both have components (simulators which are called federates or simulation units) which are governed by an entity which sets some rules for the communication and synchronisation of the simulator components. Co-simulation, which is standardised by FMI, uses a governing strategy where a orchestration algorithm is responsible for synchronisation and other aspects of simulation units. Whereas in federated simulation, standardized by HLA ('IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules' 2010), provides more freedom to federates, but its runtime interface has some services which provides the synchronisation and data exchange between the federates. FMI has some pre-defined data exchange points, and only at those points, the data is exchanged between the simulators, whereas in federated simulation, communication can be more frequent.

With the lack of interoperability of simulation approaches and related underlying models and execution environments (simulators), the realisation of such system-level simulations incorporating all individual simulations is technically hard to achieve within a single simulation system. Concepts like the virtual factory (Xu et al. 2020) aim at simulating a collaborative manufacturing network, which requires the interoperability of the systems and related underlying models in the collaborative manufacturing network where different parts are controlled by different organizations and the independence (and confidentiality) is a desired quality. In addition, where a simulator is provided by a third party (or the manufacturer), there is often a commercial interest in keeping the details confidential.

In short, many tools and interfaces are available to solve parts of the digital twin simulation problem, but simulating complex systems such as a collaborative manufacturing network still is a big challenge. This chapter provides an overall approach to integration and interoperability of different simulation systems, i.e. federated simulation.

In terms of simulation interoperability, there is an additional layer of both behavioural interoperability of the simulation system as well as policy interoperability in terms of avoiding unlimited sharing of simulation configuration and parameters.

Instead of looking at interoperability of configurations of digital twins per se, we look at the interoperability of the simulation (that are enabled/driven by the digital twins). Looking at making simulation interoperability, it allows detecting incompatibility of the digital twins in the configuration (and thus an operational incompatibility), rather than loosing this information due to independent interoperability. While simulations are generally defined using higher level constructs, these constructs are specific to their execution environments and changing the simulations to match a different standard is not trivial, requiring the environment to be part of the simulation definition, effectively requiring the simulation environment to function as a virtual machine.

Instead of the single simulation environment approach we propose federated simulation as an alternative approach that provides practical interoperability combined with flexibility and an opportunity for policy enforcement at the level of each digital twin.

3.1 Concepts of interoperable digital twin simulation

To present a formal approach for interoperable digital twin simulation, we first need to consider the conceptual problem that needs to be addressed. Figure 6 presents this conceptualisation.

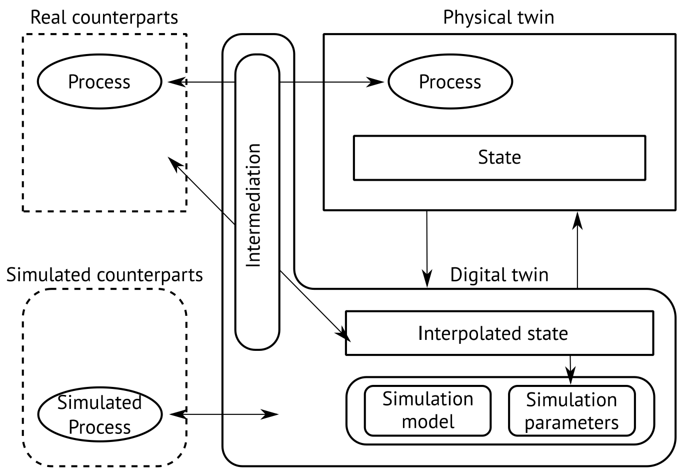


Figure 6: Digital twins' simulation.

As a starting point of the conceptualisation, a digital twin is the digital representation of a "physical" twin (Jones et al. 2020) forming a digital twin pair. We would note that this "physical" twin could be something abstract such as a business process (manually executed or managed automatically using a BPMS). In any case, most physical twins have some sort of process they perform (which could be doing nothing), as well as some state.

The digital twin then maintains its own model/interpolated representation of the state of the physical twin through interaction with the physical twin. To maintain the model, it may be using sensors and actuators as well as interpolation, prediction and other soft-sensor techniques. To maintain the information and allow for monitoring and control, the digital twin, where possible, intermediates the interactions the physical twin has with its (real) counterparts in the overall processes it is involved in.

The intermediation provides interfaces and ability for the digital twin to be an effective stand-in for the physical twin when the digital twin has a sufficient simulation model for its process/behaviour. Replacing all physical twins with their simulation capable digital twins effectively provides for the simulation of the entire process. When simulating, the simulations would interact with simulated counterparts (generally also digital twins-based simulations) rather than "real" counterparts.

There are various challenges to updating simulation models and parameters as the state of a physical twin changes (especially when it represents factors such as wear). In addition, in order to be able to simulate the full process, it is necessary that the digital twin's simulation does have the ability to interact with its environment (simulated counterparts) in line with the behaviour of the physical twin.

In a manufacturing context there are many moving parts, conceptually depicted in Figure 7. Not only is there an overall process handling the manufacturing of items through various assets (e.g. machines) that transform (semi)products in various ways, either modifying, combining, or a combination, but there are many component twins for the assets involved.

From a process perspective, manufacturing is performed along some process. This could be manual, ad hoc, managed through a manufacturing execution system or even including some sort of business process management. From a digital twin perspective, to be able to have an effective digital twin, some sort of structured process is needed that can be measured (for example, using process mining (Van Der Aalst 2012)) and modelled. The Digital (Process) twin here represents an abstract process rather than a physical entity. However, even in case of automated process execution, the integrated simulation afforded by a digital twin does still have advantages.

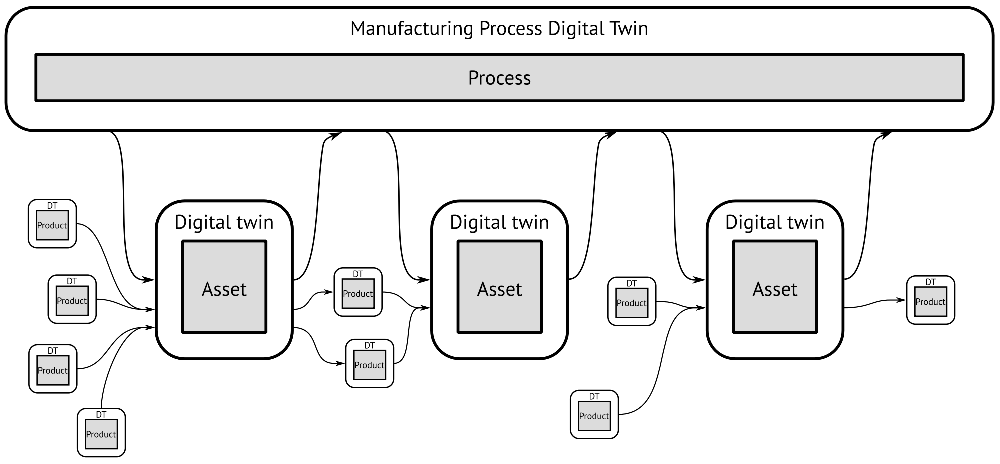


Figure 7: Potential digital twins in a manufacturing process.

The manufacturing assets have a straightforward relation with digital twins in monitoring their performance, as controlled by the process. In a digital twin context, the interaction is ideally intermediated by the digital twin (for monitoring) but using the same interfaces as the physical twin has. The (semi)products consumed and produced (or transformed) may also have digital twins. Here, interactions may very well be physical, and explicit interaction with digital twins may be needed to allow the digital twins to update their models.

Product digital twins can range from very simple to very complex. At the base level, it can be some data stored against a part serial number, where at an advanced level it can be a digital twin of a complex machine that is based upon (and has tracking of) its original creation, but then is also related to the products it makes in turn.

A key observation to make here is that monitoring and modelling of processes requires interactions to be explicit to be most effective, for example, to verify the validity of a configuration. Explicit interactions require explicit interfaces to exist for these interactions. Given such interfaces for the physical twins, the simulation can inherit these interfaces, as well as the solutions to syntactic, semantic and behavioural interoperability. It may be feasible to use simulated transports for transport interoperability, but generally, simulation would use simulation specific transports (in cases where transport simulation is not desired this reduces simulation overheads). Policy level interoperability would be orthogonal, but where digital twins are used for enforcement, this can be incorporated in the simulation where required.

Where digital twins differ from normal simulation, this stems from the combination of modelling, intermediation and simulation. This combination allows the simulation to be adaptive, starting from a general simulation model based upon the general asset class, over time it will observe the properties of the actual physical twin and be able to adapt to provide a higher accuracy prediction of the specific twin.

In practical terms, there are two restrictions upon the implementation of the simulation. First, it would be required that digital twin snapshots could be used so that not only does the simulation model not change during a simulation, but also so that simulations are replicable. The second restriction is that simulation can be done isolated away from the intermediation and monitoring part of the twin. When using simulation to determine the properties of a setup or to determine the best parameters, this involves many rounds of simulation, something resource intensive. The simulation should not be allowed to interfere with the remainder of the digital twin and the operational process.

3.2 Extended digital twin simulation support

As we wish to allow the use of existing simulators, further tools in the form of interaction primitives are beneficial. In addition, looking at simulating configurations of digital twins, these configurations may have standard components where any difference in implementation does not make a difference. Some of the standard twins that may help in creating simulated configurations of digital twins is also discussed.

3.2.1 Interaction primitives

A key aspect of interaction in digital twin simulation is that the interactions mirror the physical twins as much as possible as this allows for the most complete verification of the configuration of digital twins such that the configuration and its simulation effectively form a digital twin pair by themselves.

Interaction between two physical twins can either be normally mediated through a digital twin, for example, through automated management and interaction. In this case, the simulated interactions automatically match the physical twin counterparts, except for their target as being simulated.

Interactions between physical twins can also take the form of physical interactions (i.e. a cutter interacting with a steel plate by cutting it). This may already have been modelled as part of creating the digital twin but may also require some additional constructs for simulation.

Based upon the principle of mirroring the physical twin interactions where possible, common interaction patterns should be supported by (federated) digital twin simulation platforms. Here we assume that the interaction is modelled as some sort of service interaction. Different patterns are discussed below.

The basis for our suggestions is that there would be a platform for federated digital twin simulation. Even independent digital twins' simulation would be based upon existing platforms such as SimPy3, a python based simulation library. Such platforms provide various higher level interaction primitives. To support federation without requiring the simulation models themselves to be modified, it is needed to ensure that the interaction primitives are adapted to work in the federated context. In effect, this means transforming the libraries/platforms to be built on top of the federation platform, ideally with minimal changes. In addition, some primitives allow for increased efficiency of the simulations.

Below we present a list of such higher-level interaction primitives based upon the features typically found in simulation libraries and inter-process communication (IPC) approaches. It is worth noting that, with a basis in IPC, implementation of higher-level communication structures on top of base primitives is a well-studied field with good solutions that is out of scope of this chapter.

3.2.1.1 Unidirectional messaging

In practical context, different events are defined, including messaging events with payload, destination and an optional sender. This increases efficiency by simulators not needing to be synchronized for irrelevant events. When including standard support for receiving messages, it also simplifies the overall modelling of messaging between digital twin simulations.

Simulation must be synchronised at a point of interaction. As such, the synchronisation should be performed through the federation system to ensure that the interaction is received at the correct time point. Messages as a restriction upon events embed.

As messages from one simulator may have consequences in another, even cyclically, it is important from a correctness perspective that there is an order to these messages. In line with physical constraints, this implies that message delivery must be strictly after message sending in simulated time (even if the time difference is infinitely small). Similarly, any consequences of an event/message should be strictly after its receipt.

In our implementation of federated digital twin simulation on top of simply the model was such that unidirectional messaging was sufficient, but clearly necessary. Overall, messages are straightforward to implement, mainly requiring some sort of identity/address for the receiver (and sender if a reply is expected).

3.2.1.2 Bidirectional messaging
Most uses of communication in the real world, including service invocations, are bidirectional, or even conversational. A full bidirectional reliable messaging system could be defined upon unidirectional messaging, but as a minimum, a message identity needs to be added together with an optional "in-reply-to" attribute that links two messages as being in the sequence of the same conversation.

As federation is per definition asynchronous, the bidirectional messaging is asynchronous itself. For a simulation platform, we would wish to provide/substitute synchronised messaging primitives so that simulators themselves do not need to adapt to the asynchronous nature of the simulation.

3.2.1.3 Buffers and Queues
If we consider that a machine working on a part is per definition also a buffer for that part, as well as the need for temporary storage in the interaction between physical twins, buffers and queues are a very common interaction construct. For simple machines, they may even be the only interaction needed in their normal execution flow beyond any simulated monitoring events. As an example, SimPy provides built-in container and store constructs for the purpose of modelling buffers and queues.

In a federated context, buffers and queues can be used to communicate between digital twin simulations, so they need to be implemented on top of the event system. Such buffer and queue implementations allow the simulation to pause when the buffer is exhausted or,full, depending on the operation of adding or removing items. A quantified resource system for uniform resources as provided by SimPy could be implemented on top of a buffer, but also directly implemented.

3.2.1.4 State access
Beyond direct messaging constructs, another form of interaction is through state access, in particular the observation of the state of another entity. Providing basic CRUD primitives (Martin 1983) upon events or messages is straightforward, although in most cases it should be limited to read operations (write operations are typically actions controlled by the physical or digital twin and their simulation).

3.2.1.5 Subscription
In addition to the ability to monitor state, it is beneficial to have support for subscription to state changes. This allows for increased efficiency of the simulation due to reduced polling but may also be a construct already used in simulations. As such, we would support subscription to both changes in state, as well as events. The latter event subscription being primarily an optimisation from the perspective that synchronisation upon an event is only needed if the event is being observed by another simulator.

3.2.2 Standard twins

Beyond providing support for various interactions between digital twins, a full digital twin configuration may include various standard components where specific details are not required. Examples may include undifferentiated storage space (e.g. floorspace, warehouse, or shipping container). Perhaps more interesting could be an electricity supply twin that would allow for modelling of energy consumption even if not modelling the actual network and potential supply limits incurred (max amperage through specific connections).

Modelling always has specific purposes, and constraints available resources. As such there are parts of digital twin configurations that have limited impact on the goals of the modelling or simulation. For those parts, standard "off-the-shelf" twins can provide such non-specific capabilities to complete (the simulation model of) the digital twin configurations.

These twins could either be used at the edge of the configuration/simulation or to model simple physical entities needed for completeness. Example categories of such standard twins are:

- **Storage**
  Undifferentiated storage space (of given capacity) that functions as a buffer between components. This could be a machine output.
- **Transport**
  Transport components that represent simple transport operations (with a capacity), for example, to model a conveyor belt, but potentially also a human moving parts from one place to another.
- **Utilities**
  Real world processes generally consume utilities and utility supplies. Examples are water and electricity, but it could also include supplies such as pens, paper, or even lubrication oil. Generally, the processes involved would be out of scope of a model of a (production) process.
- **Suppliers & Consumers**
  Production processes involve suppliers of various goods at one end, and consumers of the production at the other end. They would generally be sufficiently modelled in a generic way, not being a part of the process of interest.

In practical application, a base formalism is not sufficient. As such, this section has discussed both interaction primitives build upon the event/messaging system, as well as standard "twins" that can be used to model both the boundaries of a simulation and physical elements required for a complete simulation, but with little impact upon the overall simulation outcomes.

3.3 Evaluation

To experimentally evaluate the federation, we have implemented the federation algorithms to extend the SimPy simulation library. This allows multiple SimPy simulations to run federated, only mediated through event/message passing. While it would be possible to run each simulation in a separate thread or even fully independently in different processes, the implementation does not attempt that (as this does not further validate the approach).

The simulation used for validation is a (slight) variation upon the Machine Shop example (Scherfke 2013) provided in the SimPy documentation. This is a simulation where (by default) 10 similar devices are used to make products. These devices fail periodically and require a repairman to fix them. The repairman can only repair one device at a time and takes a while to do so. Machines cannot produce parts until fixed. For experimental purposes, we added log messages and the ability to use random seeds for reproducible variability.

For the federated library to be valid, the simulation results should be identical between a federated setup and a monolithic setup. We show this for the monolithic and three different federated configurations by having the simulations produce detailed simulation logs and comparing them.

The different simulation configurations are based upon a common basis. The simulation of the machines and repairman are shared, but for federation a subclass of the normal simulation environment is used. This subclass supports messaging between simulators, but is otherwise identical in its behaviour. For the third configuration (where machines are split between simulators) there are some additional complications due to the need for replication with identical seeds (this case creates a single configuration and then transplants the machines into two separate ones).

Table 1: Partial log of federated simulation.

Time	Simulator	Machine	Event	Partno.
1605	Repairman 1	8	Start Repairing	N/A
1605	1	5	start making part	148
1606	1	3	finish making part	150
1606	1	3	start making part	151
1608	1	4	finish making part	149
1608	1	4	start making part	151
1611	1	2	finish making part	142
1635	Repairman 1	8	Finish repairing machine	N/A

Table 1 and Table 2 show slices of the logs. With these adaptations, we executed the simulations in various different configurations, where different parts of the simulation were federated and compared them with each other and then ran the simulation in various configurations, comparing the resulting logs, including the non-federated version.

Table 2: Partial log non-federated simulation.

Time	Simulator	Machine	Event	Partno.
1605	Repairman 1	8	Start Repairing	N/A
1605	1	5	start making part	148
1606	1	3	finish making part	150
1606	1	3	start making part	151
1608	1	4	finish making part	149
1608	1	4	start making part	151
1611	1	2	finish making part	142
1635	Repairman 1	8	Finish repairing machine	N/A

The base configurations used were: non-federated; the repairman federated separate from a simulator for all machines, federating the repairman and splitting the machines into two separate simulators. The logs for all three configurations show equal simulation results.

In addition, there are two configurations that have two full machine simulators but either share or have separate repairmen. Both these duplicate configurations (that are not constrained by the repairman) are also equivalent.

There are various observations that can be made from the implementation. The SimPy framework provides access equivalent to step and nextTime without any need for modification of the library. There is also an event system that can be used to implement deliver without modification of the library. The actual simulation makes some shortcuts in interactions between components, so a messaging system is needed as an extension, and it needs to be invoked at appropriate places. Similarly, the simulation parameters needed encapsulation into an object to allow for multiple instances to co-exist (not an issue when not splitting up a simulation).

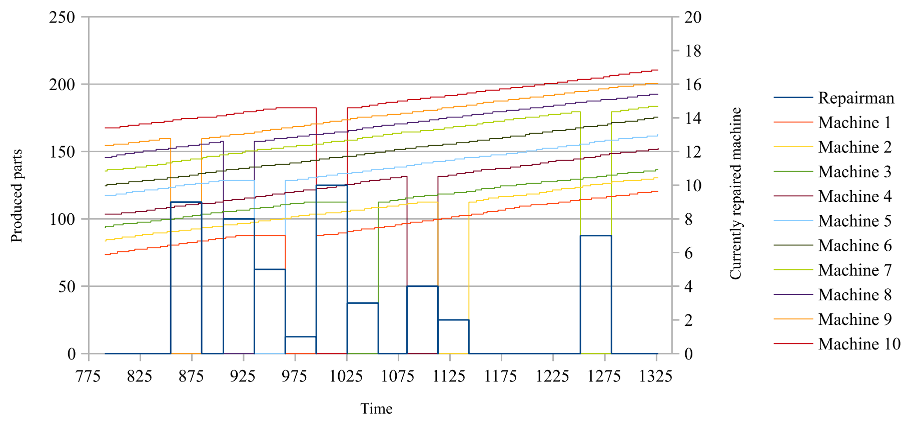


Figure 8: Simulation snapshot of simple federation.

Looking at the actual simulation result, Figure 8 provides a cut of the productivity logs (the entire log is too large to see any details). The figure is adapted in a number of ways: Initial production counts have been offset by 10 parts for each machine, so the graphs do not overlap. In addition, when a machine is under repair, this is depicted as having made 0 parts. On the side of the repairman, the vertical axis shows the machine under repair at that point, or 0 if there is no machine under repair. The figure shows the pausing of production when machines are broken (awaiting repair) as horizontal bars. It also shows the repairman contention in the repairman graph. Note that as all simulations have equal results, the graphs for the other cases would be equal and are therefore not included.

Overall, we found that the log messages were equal (except where they indicate which simulator executes the event, something expected to be different) in most cases. The simulation, however, models communication as instantaneous rather than involving some random delay. To allow for comparison, we retained this instantaneous communication, despite it not matching the requirements the formalism puts upon events (that requires consequences to be strictly after the event initiating them). In some cases, this causes differences in ordering log messages associated with the same time.

Figure 9 compares the time differences for the three different base simulation configurations, each repeated 20 times. Non-federated simulations took on average 571 ms, simple federated took 597ms and double federated took 609 ms. These results show that while there is some additional overhead by having federated simulators, with additional simulation this is limited and within expected variance for individual invocations. In addition, this is in a case where the simulations do not involve computation more complex than determining random action durations.

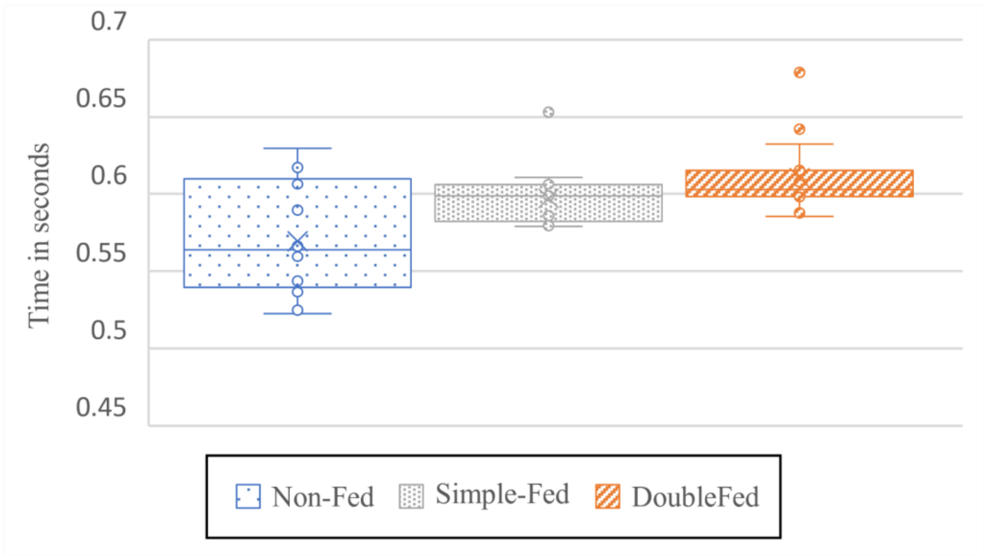


Figure 9: Comparison of run times between different scenarios.

One observation from initial comparison runs (not included as paper logs) is that there is a limit with event ordering. Due to a lack of randomisation, events (machines breaking) would occur at exactly the same time, and the order of their processing would differ between the simulator configurations (be nondeterministic). To resolve this limitation of the original simulation, the relevant times/delays were randomised, effectively providing a deterministic ordering of events. In a real-world context, a lack of determinism issue should not be an issue as simulations are normally not of such accuracy that they can accurately predict ordering within a tight time-frame.

3.4 Conclusion

Based upon a review of related work, we have set out a conceptual basis and a set of requirements for interoperable digital twin simulation concluding with a strong motivation for supporting interoperable simulation of digital twin configurations through the use of federated simulation. The interface required for federated simulation is small, maps almost directly upon the commonly used Discrete Event Simulation model and allows other simulation approaches to be supported easily while providing minimal restrictions upon simulators themselves (consequences of events must be strictly after their cause). How simulation libraries can be extended with constructs that support the implementation of digital twin simulations. Finally, the evaluation is also provided through experimental implementation for interoperable digital twins.

The framework, with its formal basis in (de Vrieze, Arshad, and Xu 2023), presents a common ground to enable the interoperability of simulations in the context of Industry 4.0 processes. The assumptions made and restrictions posed are minimal, and the base communication constructs used for simulation provide a sound starting point for simulation frameworks to add support for federated digital twin simulation. The implementation shows that coordination can be limited to only those times where needed, and simulations can execute in parallel in between. As the prototype implementation mainly amends the SimPy interface, and this is a general-purpose simulation library, it demonstrates that many simulations can be supported without, or with minimal change in a federated approach. The changes needed are related to the communication with other twins, and as such would be needed in other comprehensive simulation approaches as well and federation allows the needed changes to be restricted to such, without requiring switching simulation frameworks, or even simulation approaches.

4 Digital Twin Composition in Smart Manufacturing via Markov Decision Processes for a Resilient Factory

The term *Industry 4.0*, which denotes the fourth industrial revolution, was first introduced in Germany in 2011 at the Hanover fair, where it was used for denoting the transformation process in the global chains of value creation. At present Industry 4.0 is a result of the emergence and distribution of new technologies — digital technologies and Internet technologies — which allow the development of fully automatised production processes, in which only physical objects that interact without human participation take part. *Smart Manufacturing* is nowadays a term highly used in conjunction with the concept of Industry 4.0. Smart Manufacturing aims at improving the manufacturing processes in order to increase productivity and quality, to ease workers' lives, and to define new business opportunities. This is enabled by leveraging innovative techniques like Artificial Intelligence (AI), big data analytics, Machine Learning (ML), and Business Decision Support Systems (BDSS). The employment of these technique has made it possible to create new possibilities of interoperability, modularity, distributed scenario processing, and integration in real time with other industrial processes.

Essentially in the same period, the concept of Digital Twin (DT) was introduced as a key technology used in the industrial context. Many different definitions for digital twin can be found in the literature, mainly caused by various application areas. The first clear definition was given by NASA in 2012. They define digital twin as “an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin”. A generalized definition for a Digital Twin defines it as “a virtual representation of a physical system (and its associated environment and processes) that is updated through the exchange of information between the physical and virtual systems”.

The application of DT in the manufacturing sector impacts the way the products are designed, manufactured and maintained. On a high level, the DT can evaluate the production decisions, access the product performance, command and reconfigure machines remotely, handle the troubleshoot equipment remotely and connect systems/processes to improve monitoring and optimise their control (Kitain 2018). DTs can also be applied for process control, process monitoring, predictive maintenance, operator training, product development, decision support, real-time analytics, and behaviour simulation (Pires et al. 2019). A single manufacturing process may include hundreds of different actors, i.e., digital twins, that may suddenly fail or provide bad performance. In reality, due to their continuous use, actors could wear out and therefore have not only a greater probability of breakage but also higher costs to complete their job. At any moment in time, in order to provide resilience, the manufacturing process should be able to automatically adapt to new conditions, considering new actors (with lower cost and low probability of breaking) for the fulfilment of the manufacturing goals. This task cannot be performed manually when actors span multiple organisations that are possibly separated from both the geographical and organisational points of view. For this reason, it is crucial to have a plan for the manufacturing process to be able to manage several actors, taking into account their possible failures and costs.

In that regard, very little research effort has been put in defining automatic techniques to orchestrate manufacturing actors towards a final goal. An important step towards the development of new automated techniques in smart manufacturing is modelling DTs in terms of the provided services. This would allow to partly reuse the results obtained in the area of Web Services (WSs), such as the automatic composition and orchestration of software artefacts. Particularly, the idea is to capture the analogies and differences between DTs and WSs, and enables the integration composition of DTs through offered services and data available in the data space.

The inherent limitation of such approaches, though, is the assumption that the available services, i.e., the services that can be used to realize the target service, behave *deterministically*. This assumption is unrealistic in the case of DTs for smart manufacturing, because in practice the underlying physical system modelled as a set of services might show a stochastic behaviour due to the complexity of the domain, or due to an inherent uncertainty on the dynamics of such a system. In these cases, the deterministic service model is not expressive enough to capture crucial facets of the system under consideration.

Service composition techniques can, therefore, be used to orchestrate digital twins in order to generate a plan for a manufacturing process to reduce the costs while preserving the quality of the final outcome. In particular, the service composition techniques can be generalised in a stochastic setting, in which the services, (both machines and humans) have an unpredictable behaviour and are subject to wear. An optimal solution can be found by solving an appropriate probabilistic planning problem (solving a Markov decision process - MDP), taking into account the probability of breaking and the cost of employing specific actors. Such an approach is enabled by leveraging the capability of DTs to assess the status and the wearing of the underlying physical entity (Melesse, Pasquale, and Riemma 2020; Aivaliotis, Georgoulias, and Chryssolouris 2019). In this way, it is possible, *autonomously*, i.e., without human intervention, to obtain a production planning which is *adaptive*, as it changes every time that the manufacturing of a new product or batch is started, and *context-aware*, as it is dependent on the current status of involved actors.

4.1 Smart Manufacturing Architecture

Figure 10 represent the general architecture in Smart Manufacturing based on DTs (Catarci et al. 2019). It is composed of the following components: supervisor, orchestrator, the DTs of involved actors and the data space.

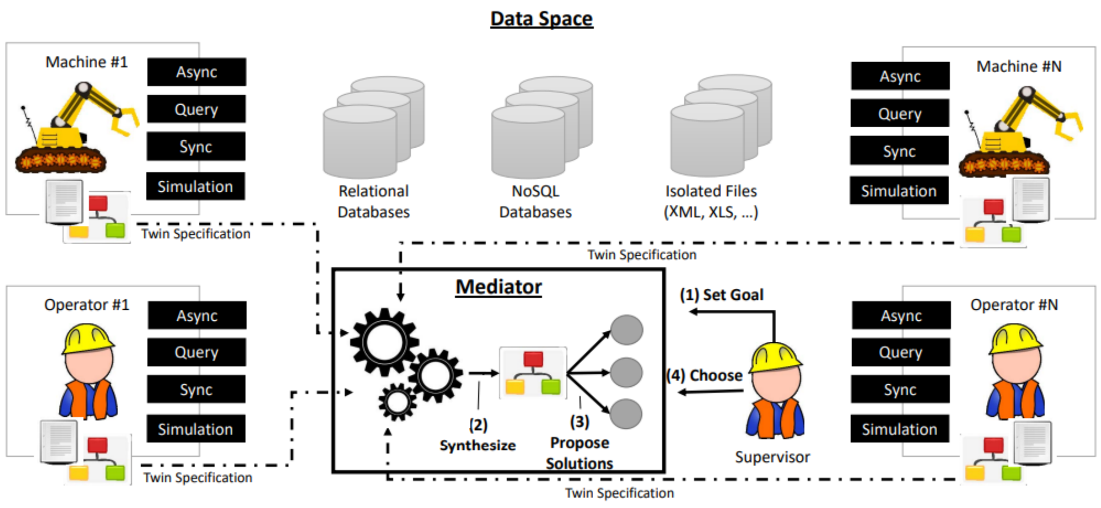


Figure 10: Smart Manufacturing architecture based on digital twins.

The DT was originally intended to denote a digital model that faithfully reproduces a physical entity and allows to perform physical simulations (e.g., mechanical solicitations). In the last years, the term has been used to more generically denote a digital interface allowing not only for simulations but also for an all-round control of the physical entity during run time. It wraps physical entities (actors such as manufacturing machines, human operators/workers and external suppliers) involved in the process and exposes a Web API consisting, in general, of three parts: the synchronous one, the query interface and the asynchronous one. The synchronous interface allows to give instructions to the physical entity. These instructions may, for example, produce a state change in a manufacturing machine (in case the twin is over a machine) or ask a human operator to perform a manual task (in case the twin is over a manufacturing worker). The query interface allows for asking information to the physical entity about its state and related information; noteworthy, these latter can be obtained by applying diagnostic and prognostic function results of machine learning. The asynchronous interface generates events available to subscribers. In addition, each DT is equipped with a specification of the provided functionalities. This specification may take a form that depends on the specific framework employed to implement the DT.

The data space contains all the data available to the process. These data are heterogeneous in their nature from the access technology point of view, the employed schema (or its absence) and the employed vocabulary. It is important to note how the DTs contribute to the data space with both the query API and the asynchronous one. Other sources for the data space may include relational and no-SQL databases or unstructured sources such as spurious files, which constitute the factory information system.

The human supervisor is the one defining the goals of the process in terms of both final outcomes and key performance indicators to be obtained.

In order to reach the goal defined by the human supervisor, available twins and data must be integrated. This task is fulfilled by the mediator, or more specifically orchestrator. The orchestrator acts in two phases: the *synthesis phase* and the *execution phase*. During the *synthesis phase*, the specifications of the APIs exposed by digital twins and the meta-data (e.g. data source schemas) available in the data space, are composed in order to construct a manufacturing process. During the execution phase, the orchestrator runs its program by preparing the input messages for the single twins involved in the proper sequencing/interleaving. Indeed, as each twin may potentially adopt a different language and vocabulary, in order to compose required input/output messages, the orchestrator translates and integrates the data available in the data space to comply with the format requested by the specific called service. An important aspect of the described architecture is that multiple companies can participate in the process (typically those ones involved in the value chain) and it is not reasonable to have twins directly communicating with one another. Therefore, the role of the orchestrator is fundamental, being the component that can access the services offered by the twins available in the different companies.

4.2 Manufacturing Orchestrator

The *orchestrator* (see Figure 11) is a software program intended to guarantee that the manufacturing process fulfils the goals imposed by a human *supervisor* selecting services to be employed maximizing a set of Key Performance Indicators (KPIs) (Catarci et al. 2019). Specifically, it aims at describing a manufacturing process, which needs to be executed while reducing the total cost, where the concept of cost embodies both the economic cost and quality loss due to the specific choices made by the orchestrator. For the sake of space, Figure 11 depicts a single machine #i and a single operator #j, but it is possible to imagine a factory including a multitude of them. In addition, each action (task) of a manufacturing process can be performed by different machines or humans. The choice of the specific actor to be employed for a specific action is driven by different factors, including the cost and the potential quality loss due to wear or obsolescence. As an example, humans are usually more expensive than machines for a specific action in terms of economic and time cost, but a worn machine could negatively impact the quality of the final product.

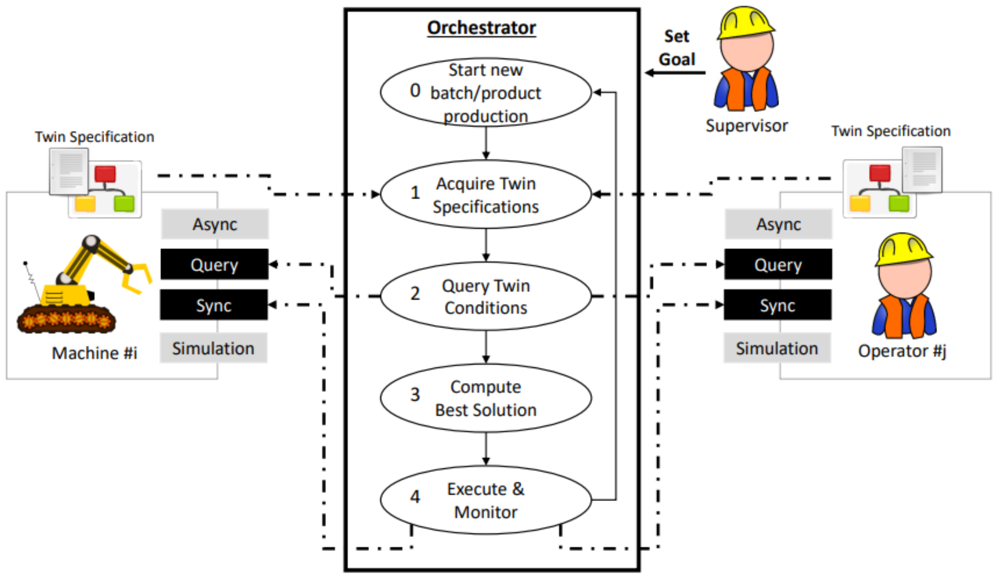


Figure 11: Orchestrator architecture.

The orchestrator is called every time that the production of a new product or a batch of products is started. At this point, the orchestrator:

- Gathers the specifications of all the available DTs.
- According to these specifications, the orchestrator employs the query interface of each DT to obtain the current status of the machine. The status is continuously kept updated by the DT and may include the probability of breaking and the wear level of the correspondent device.
- Computes the optimal solution given the current status and capabilities of each available actor, as reported by the respective DTs. A solution, or plan, consists of a sequence of manufacturing actions, consistent with the description provided by the supervisor, and of an assignment of a specific actor (machine or human) to each action.
- Automatically executes the manufacturing process following the obtained plan, by leveraging the synchronous interface of each involved twin. In addition, it monitors the execution, by taking countermeasures when needed (e.g., when a specific machine breaks).
- After a product or batch is completed, the orchestrator starts over, waiting for a new production start event.

Noticeably, every time that a new production starts and the orchestrator is invoked, the decision this latter will make will be influenced by the production history, as the DT behind each service involved in the production will update information about costs and likelihood of a breaking event.

Noteworthy, the definition of the orchestrator is open to several different implementation strategies. In particular, an orchestrator can be implemented as a tool that finds an optimal policy to a Markov Decision Process (MDP) which is constructed by combining in an innovative way the different MDPs modelling available actors/services.

4.3 Composing the Digital Twins

As DTs and corresponding physical actors can be modelled in terms of their service interface, their composition to obtain a manufacturing goal can be performed similarly to what has been done for Web service composition in the area of classical information systems. The problem of service composition, i.e., the ability to generate new, more useful services from existing ones, has been considered in the literature for over a decade (Hull 2008; Medjahed and Bouguettaya 2011; De Giacomo, Mecella, and Patrizi 2014).

The goal is, given a specification of the behaviour of a (complex) target service, to build a controller, known as an *orchestrator*, that uses existing (composing) services to satisfy the requirements of the target service. Such target service is the manufacturing process, whereas the composing services are the DTs wrapping physical actors.

It is possible to formalise the orchestrator by taking inspiration from the approach known as the "Roman model" (Berardi et al. 2003; 2005). Particularly, in the Roman model, each available Web service is modelled as a finite-state machine (FSM), in which at each state, the service offers a certain set of actions, where each action changes the state of the service in some way. The designer is interested in generating a new service (specified using an FSM, too), referred to as the target service, from the set of existing services.

A limitation of such an approach, when applied to the world of smart manufacturing, is that whereas Web services behaviour is predictable, i.e., the execution of an action in a specific state deterministically takes the service from one state to another, the execution of an action from an industrial actor (either machine or human) can have unpredictable effects (e.g., the machine breaks), which must be taken into account while computing a solution. In addition, the behaviour of a physical actor in manufacturing may degrade over time due to wearing.

Moreover, it is not always possible to synthesise a service that fully conforms to the requirement specification. This zero-one situation, where we can either synthesise a perfect solution or fail, is often restrictive. Rather than returning no answer, the notion of the "best-possible" solution is preferred. A solution to this last issue has been proposed in (Brafman et al. 2017), where the authors discuss and elaborate upon a probabilistic model for the service composition problem, first presented in (Yadav and Sardina 2011). In this model, an optimal solution can be found by solving an appropriate probabilistic planning problem (e.g., a Markov Decision Process) derived from the services and requirement specifications. Still, the proposed solution is applicable only to deterministic and non-degrading services such as Web services.

The solution relies on the concept of Markov Decision Process (MDP). An MDP M is a discrete-time stochastic control process containing *(i)* a set of states, *(ii)* a set of actions, *(iii)* a transition function that returns for every state and action a distribution over the next state, *(iv)* a reward function that specifies the reward (resp. the cost), a real value received (resp. paid) by the agent when transitioning from state s to state s' by applying action a, and *(v)* a discount factor in $(0, 1)$. A solution to an MDP is a function, called a *policy*, assigning an action to each state, possibly with a dependency on past states and actions. The *value* of a policy r at a state is the expected sum of rewards when starting at state s and selecting actions based on the policy. This expected sum of rewards could possibly be discounted by a factor l, with $0 < l < 1$. Typically, the MDP is assumed to start in an initial state s_0, so policy optimality is evaluated with respect to $r(s_0)$. Every MDP has an optimal policy r^*. In discounted cumulative settings, there exists an optimal policy that is Markovian, i.e., that depends only on the current state, and deterministic. Among the techniques for finding an optimal policy of an MDP, there are *value iteration* and *policy iteration*.

4.3.1 Modelling Digital Twins as Stochastic Services

In order to overcome the limitations of the Roman model when applied to smart manufacturing, *each DT and the underlying physical actor can be modelled as a stochastic service*. A *stochastic service* is a tuple containing the finite set of the service states, the initial state, the set of the service's final state, the finite set of services' actions, the transition function, and the reward function.

The stochastic service is the stochastic variant of the service defined in the classical Roman model, and it can be seen as an MDP itself. Such a solution allows for the flexibility required to model a physical machine operating in manufacturing environments. As an example, specific states can be defined to model unavailability conditions (e.g., a broken machine) and the probability of ending in such states. In addition, rewards can be used to model the degradation of service quality in time. Repair costs to recover from unavailability states can also be modelled to take into account in the solution the possibility to fix broken devices if they guarantee an high quality. All of these parameters can be computed and continuously refreshed by the DTs by using models trained by the equipment manufacturers.

The *stochastic system service* C is defined as the community of stochastic services. Intuitively, the stochastic system service represents, in a single MDP, all the stochastic services, i.e., all the DTs and underlying physical actors. As a consequence, its status includes the current status of all the composing services. A specific action performed on the system service changes only one component of the current state, corresponding to the service selected to execute that action.

4.3.2 Modelling the Manufacturing Process

In order to model the manufacturing process, the concept of *target service* T, introduced by the Roman model, is needed. The term denotes a complex service that can be obtained by composing simpler services. In this case, the manufacturing process must be obtained by composing the functions of available DTs. In particular, the definition adapted to the stochastic settings by (Brafman et al. 2017) is used.

A *target service* is defined as a tuple containing the finite set of service states, the initial state, the set of the service's final state, the finite set of services' actions, the service's deterministic and partial transition function, the action distribution function, and the reward function.

Noticeably, the target service itself, as the stochastic services modelling the DTs, is a particular case of MDP. In the vast majority of cases, manufacturing processes (differently from manufacturing actors) are deterministic.

4.3.3 The Composition Problem

The set of joint histories of the target and the system service is defined as $H = S_t \times S_z \times (A \times S_t \times S_z)^*$. An orchestrator, is a mapping from a state of the target-system service and user action to the index of the service that must handle it.

Since the stochastic nature also comes from the services, the orchestrator *does* affect the probability of a history. Moreover, in general, there are *several* system histories associated with a given target history.

A target history is realisable by an orchestrator if for all joint histories, the orchestrator is well-defined, i.e. it can perform all the actions requested by the target for every possible (stochastic) evolution of the system service. The orchestrator is said to *realise* a target service if it realises all the histories.

The value of a joint history under orchestrator is the sum of discounted rewards, both from the target and the system services.

Intuitively, what is taken into account are both the reward that comes from the execution of action in the target service, but also the reward associated with the execution of that action in service chosen by orchestrator.

The expected value of an orchestrator $v(y)$ is the value of the realizable histories under orchestrator (i.e. all the possible target histories which are processed correctly). An optimal orchestrator is defined as $y = argmax\ v(y')$.

Theorem: assuming that (1) the target is realisable, and (2) every target-system history has strictly positive value, if the orchestrator is optimal, then the orchestrator realizes the target.

Proof: assuming (2), if the set of target histories realisable using orchestrator y contains the set realisable using orchestrator y', then $v(y) \geq v(y')$. Moreover, if the set of histories realizable by y but not by y' has positive probability, then $v(y) > v(y')$. If a target history is not realizable by y', there exists a point in ht where y' does not assign the required action to a service that can supply it. Thus, any history that extends the corresponding prefix of ht is not realizable, and the set of such histories has non-zero probability. Since we assume all histories have positive value, the optimal orchestrator would always prefer realizing all possible target histories (which, by assumption (1), are all the ones to realize), possibly optimizing for the rewards coming from the services' actions, and therefore realize the target. Note that by definition of $v(y)$ all the joint histories whose associated target history is not realizable by the orchestrator do not contribute to the value of an orchestrator (even the ones where y is well-defined). ∎

4.3.4 The Solution Technique

The solution technique is based on finding an optimal policy for the *composition MDP*. The composition MDP is a function of the system service and the target service. It is composed by the set of states, the set of action, the transition function and the vector of reward functions.

Even if the composition MDP is obtained by combining the system service and the target service, it has completely different characteristics. Indeed, the next action to perform is part of the state of the MDP, whereas the "action" is the selection of a specific service to execute that action. This means that by solving the composition MDP, an assignment of manufacturing actors to actions (manufacturing tasks) as well as a sequence of actions is found.

This definition is pretty similar to the construction proposed in (Brafman et al. 2017), with the difference that the transition function also needs to take into the account the probability of transitioning to the system successor state when doing a system action. Moreover, in the reward function, it is needed to take into account also the reward observed from doing system action and sum it to the reward signal coming from the target. The state sink is an absorbing state, that transitions only to itself and that generates only rewards of value 0, and it is needed to make the transition function of the MDP well-defined. If a trajectory reaches that state, it means it represents an unrealizable (joint) history.

Theorem: assume that for all policies and target histories, an orchestrator found a solution. If it is an optimal policy, then the orchestrator is an optimal orchestrator.

Proof: Observe that for realisable joint histories, for some policies and orchestrator associated to the policy, there is an obvious one-to-one relationship between the joint histories and non-failing trajectories of the composition MDP. By construction, for any joint history and policy, the value of the orchestrator is the total return of a trajectory obtained by following the policy divided by the discount factor (this is because the MDP requires an initial auxiliary action needed for the equality). Then, the value of an orchestrator is proportional to the value of the initial state of the MDP by following policy $p\ v(y) = vp$, where vp is the value of the policy. Given that, the thesis holds because $argmax\ v(y) = argmax\ vp$. ∎

To summarize, given the specifications of the set of stochastic services and the target service, the orchestrator first computes the composition MDP, then finds an optimal policy for it, and then deploys the policy in an orchestration setting and dispatches the request to the chosen service according to the computed policy.

4.4 Case Study

In order to show the suitability of the approach, a real-world application scenario of a ceramics manufacturing company is described. Figure 12 depicts a snippet of the process to be automated and monitored expressed as a target service.

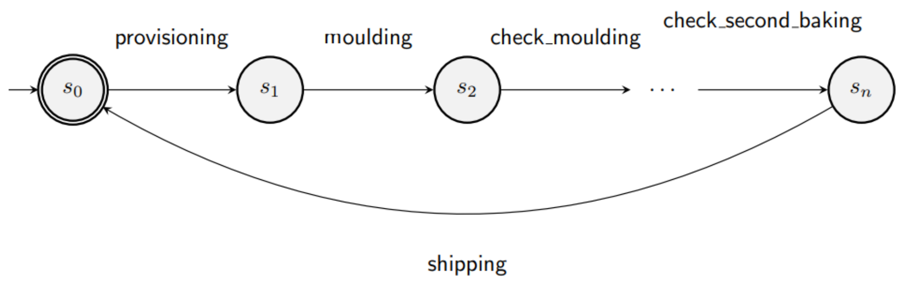


Figure 12: State machine of the target.

The process is a deterministic sequence of actions. The complete sequence is the following: (1) provisioning, (2) moulding, (3) drying, (4) first baking, (5) enamelling, (6) painting, (7) second baking, and (8) shipping. Some of the actions are followed by the corresponding checking actions, which verify the correctness of the output (e.g., the check moulding action checks the outcome of moulding). Each action in the manufacturing process can be executed by different machines or human workers. In particular, it is possible that the same action is provided by different models of the same machine, and that these machines can be replaced by human operators. The DTs corresponding to these actors must be modelled as stochastic services. Each actor is associated with a unique identifier i, which allows to specify the parameters of the associated MDPs. Services are classified into three categories, according to their complexity and provided actions.

Simplest services like those exposed by actors with no possibility to break and not provide any checking functionality. Actors in this category are external suppliers, which are seen as black boxes providing an action with a specific cost. Such services have a single state and a self-loop deterministic (with probability 1.0) transition triggered by the operation action. The transition is associated with a cost to perform the action.

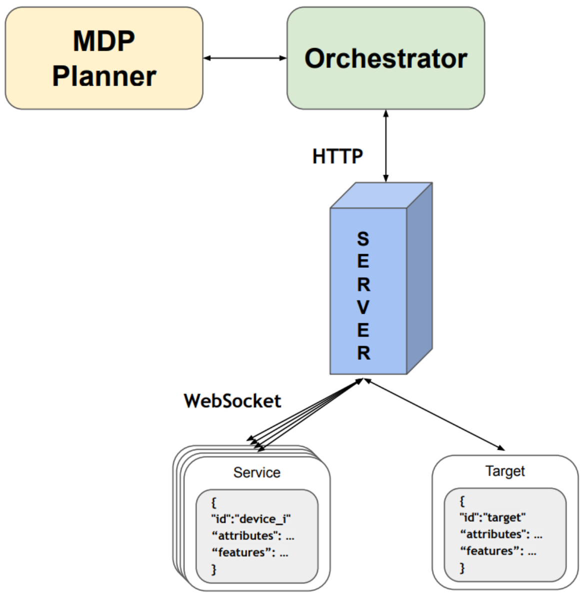
Services that represent human workers have two states and no possibility to break. The service starts in the available state and the operation action available. Executing operation action the service deterministically (with probability 1.0) ends in the done state, with a certain cost that is smaller than zero. In the done state, the check_operation action is available, assumed to be executed by the target right after operation action to make the service available again after it has completed an action.

A complex service that has the possibility to break is initially in the available state. The execution of the operation action takes with probability bi to the broken state, and with probability $1 - bi$ to the done state. In both cases, the cost of performing operation action is $ci < 0$. The probability bi models the chances of the machine to break while performing operation . The action check_operation is assumed to be executed by the target right after the operation in order to make the service available again, and additionally, to force the repairing in case the service is in the broken state. In this latter case the repair cost for the service is $ci, r < 0$.

The goal of the orchestrator is to first find a plan such that the overall expected sum of rewards is maximised (or, equivalently, that the expected sum of costs is minimised), even if the orchestration is not guaranteed to succeed in all the cases. The plan assigns an actor to each action taking into account breaking probabilities and action and repair costs provided by the DTs. It is not straightforward indeed to determine a-priori which service a certain action must be assigned to. For example, it might be the case that despite the action cost of a machine is low, its breaking probability might be high, and considering the repair cost it might let us to prefer a human worker for that action.

4.5 Software Architecture

Figure 13 depicts the software architecture for the DTs composition. The server allows devices (both available services and target service) to connect to the server via Web Sockets, while the orchestrator can interact with the server via HTTP requests. The services can connect to the server in order to register themselves and then wait for requests of action execution or maintenance tasks. The orchestrator can interact with the server in the following ways: query the server to retrieve both the specification and the current state of the available services and the current active target service, request an action from the target, request the execution of an action to be performed by a service, and request maintenance of the services.



```
{
"id":"device_i"
“attributes": ...
“features”: ...
}
```

```
{
"id":"target"
“attributes": ...
“features”: ...
}
```

Figure 13: Software architecture.

Each actor, the target process (service), and the orchestrator, which are implemented as separate processes, do not communicate directly with each other. Their behaviour specification is recorded by the server at registration time, while the features of the current state are updated during the execution of the process. The description of the services is provided as a JSON document containing:

- an id that uniquely identifies the device;
- a set of attributes containing the static properties of a DT;
- a set of features modeling the dynamic properties of a DT.

The orchestrator works as a client for the server and communicates with it through the HTTP protocol. The Server then dispatches messages from the orchestrator and the DTs and vice versa.

Every time that an available actor is used to perform a certain action, it undergoes a slight wear. Obviously, this does not happen for the services that cannot break, like services with a single state and services provided by human workers. After the service executes the action, its MDP parameters change, in particular the probability that the service will end in a broken state grows.

Moreover, a machine that is wearing out is less performing, so the cost of executing an action also increases. In particular, it is assumed that at the beginning of the manufacturing production every machine starts from a low broken probability and a low cost to perform a certain action, as it is not worn. At each iteration, i.e., at each use of the machine, it will gradually start to degrade, so both the broken probability and the cost increase.

Different services (i.e., actors) that can perform the same action. The orchestration is able to execute actions following the optimal policy that allows the choice of the best service that has a low cost and a minimal chance of breaking. Since at every call the probability of breaking and the cost increase gradually, the optimal policy must be recalculated at every repetition of the manufacturing process. What the provided implementation shows is that despite initially the machine is chosen for the painting action (because it has low-cost respect to the human), at a certain point the human will become more convenient.

Every machine that breaks can be repaired, returning available, at a certain cost. Beside this aspect, which is taken into account while computing the new policy, a scheduled maintenance strategy is implemented. In particular, the orchestrator periodically sends a maintenance event to each actor, which restores its quality, i.e., resetting the breaking probability and the action execution cost. In this way, all the machines that have reached a significant state of wear can be restored and return to their initial status. This allows to overcome the problem that once a machine degrades it will no longer be chosen. Through scheduled maintenance, all the machines are checked and repaired periodically, in such a way as to ensure their optimal functioning.

4.6 Conclusion

Composition techniques offer many possibilities in smart manufacturing. The intuition is that, like a Web service, a DT, which is a fundamental concept in smart manufacturing, can be described as a stateful automaton and, as a consequence, DTs can be combined following approaches that have been proposed to combine Web services aiming at a specific goal.

DTs are considered key components in smart manufacturing. They bridge the virtual and real world with the goal to model, understand, predict, and optimize their corresponding real assets. Such powerful features can be exploited in order to optimise the manufacturing process. Web service composition and Markov Decision Processes (MDPs) can be combined together to automatically propose an assignment of devices to manufacturing tasks. A stochastic service composition was described, in which also the services are allowed to have stochastic behaviour and rewards on the state transitions. This assignment takes into account the uncertainty typical of the manufacturing scenario, thus overcoming limitations of approaches based on classical planning. Obtained policies are proven to be optimal with respect to cost and quality, and are continuously updated in order to adapt to an always evolving scenario.

5 Compliance and Conformance for Processes in Smart Factories

A wealth of techniques have been developed to help organisations understand their processes, verify correctness against requirements and diagnose potential problems. Such understanding and verification is even more vital to smart factories due to their inherent reliance on adaptive processes. In general, these verification techniques allow us to check whether a process *conform* or *complies* with some specification, and each of them is specifically designed to solve a particular business problem at a stage of the process life cycle. However, the terms conformance and compliance are often used as synonyms and their distinct differences in verification goals is blurring (Groefsema, Beest, and Governatori 2022). As a result, the terminology used to describe the techniques or the corresponding verification activity does not always match with the precise meaning of the terms as they are defined in the area of verification. Consequently, confusion of these terms may hamper the application of the different techniques in smart factories. In this section, we aim to provide definitions and a unified terminology of compliance and conformance throughout the process life cycle. Moreover, we explore the dangers when the related techniques are used incorrectly. In doing so, we aim to improve adoption of these techniques within smart factories by clarifying the relation between techniques and their intended goals.

5.1 Formal Verification

Validation and verification are well-known evaluation procedures used to investigate whether a software or hardware product fulfils its intended purpose (International Organization for Standardization 2017). *Validation* investigates whether the product fulfils the needs of the user, that is, it tries to answer if the correct product is being made. *Verification*, on the other hand, investigates if the product matches with its specifications, or whether the product is being made correctly. When applying formal methods of mathematics to verification, the procedure is called *formal verification*. Formal verification entails proving or disproving the correctness of a model with respect to a *specification* using formal methods of mathematics. In this case, the model is a representation of the actual system (e.g., based on a specification), just like a business process model is a representation and specification of the actual business process that is being performed.

The procedure of verification is an important aspect of the life cycle of processes (van der Aalst, ter Hofstede, and Weske 2003). An overview is given in **Error! Reference source not found.**, where we map the process artefacts of the life cycle - represented by the circles - with the verification techniques, represented by the arrows connecting different artefacts. For each verification technique, the artefact used as the specification is connected to the artefact used to represent the model using an arrow. For example, the design properties (specification) are verified against the business process model (model) when checking process correctness. For completeness, two dashed arrows representing the validation relations have also been included.

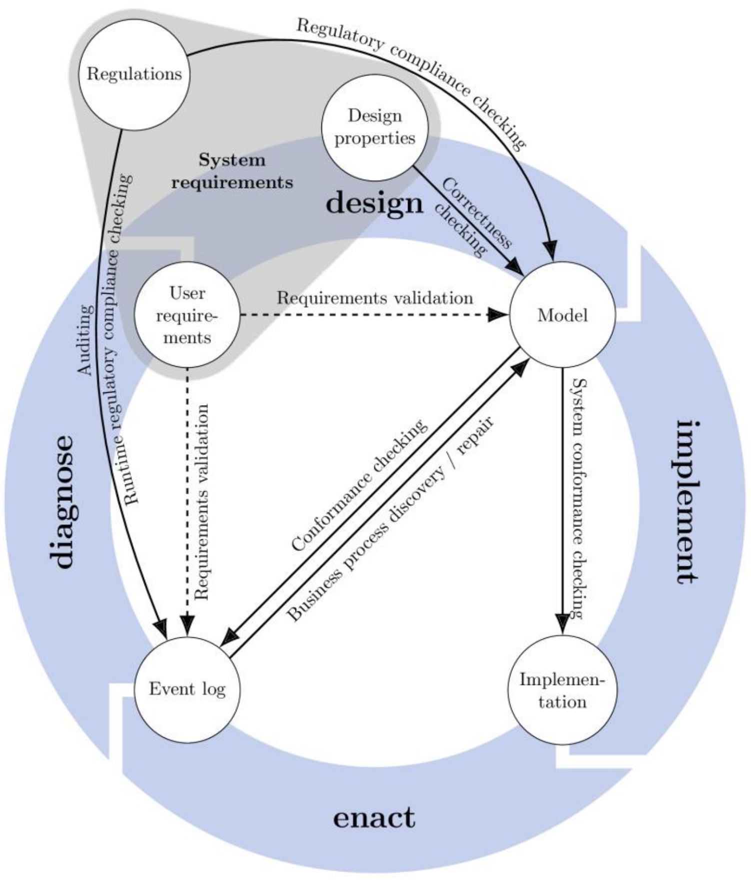


Figure 14: Verification techniques applied during the life cycle of processes.

Given the process of verification, between artefacts two possible relations can be proven: (i) relations that establish *conformance*, and (ii) relations that establish *compliance*. The first defines a relation between a specification and an implementation, while the latter defines a relation between two specifications. More formally:

Definition 1 (Conformance) *A relation between a specification and an implementation that holds when (observed behaviour of) the implementation fulfils all requirements of the specification (when the implementation conforms to the specification) (International Organization for Standardization 1998; Milosevic and Bond 2016).*

Definition 2 (Compliance) *A relation between two specifications, A and B, that holds when specification A makes requirements which are all fulfilled by specification B (when B complies with A) (International Organization for Standardization 1998).*

5.2 Techniques for Process Verification

Business processes are verified towards a number of different goals. Existing verification techniques can be classified into those that have the goal of system conformance, process conformance, model conformance, model compliance, or regulatory compliance. Note that the strict definition of compliance (Definition 2) describes a relation between two specifications and not a relation between a specification and an implementation. As a result, the goals of system and process compliance are included under regulatory compliance. Each of these goals may have multiple supporting techniques. Such techniques have the same goal, but often use different artefacts at different stages of the life cycle of processes. We define the following techniques:

Definition 3 (System conformance checking) The process of verifying conformance of the implementation towards the business process model.

Definition 4 (Process conformance checking) The process of verifying the conformance of the observed behaviour of the implementation, as recorded in the event log, towards the business process model.

Definition 5 (Conformance checking for repair) The process of verifying the conformance of the normative behaviour of the business process model towards the observed behaviour of the implementation, as recorded in an event log.

Definition 6 (Correctness checking) The process of verifying compliance of the business process model towards the design properties.

Definition 7 (Regulatory compliance) Doing what has been asked or ordered, as required by rule or law *(International Organization for Standardization 2017)*.

Definition 8 (Regulatory compliance checking) The process of verifying compliance of the business process model towards the regulations in order to prove or disprove regulatory compliance of the modelled behaviour.

Definition 9 (Runtime regulatory compliance checking) The process of verifying the conformance of the currently observed behaviour, as recorded in the event log, towards the regulations in order to prove or disprove regulatory compliance of the currently observed behaviour.

Definition 10 (Auditing) The process of verifying the conformance of the observed behaviour towards the regulations in order to prove or disprove regulatory compliance.

Within the area of business process management, the term business process conformance is mostly referred to in the context of the popular mining technique, while the term business process compliance generally refers to the context of regulatory compliance. In the context of verification, however, conformance and compliance are defined in the contexts of their *relations* (i.e., Definition 1 and Definition 2). When comparing perspectives, the use of the conformance and compliance terms does not match, as the *relation* and the *goal* of verification are used interchangeably. Table 3 summarizes the verification techniques illustrated in **Error! Reference source not found.**. The table lists each technique together with the stage of the life cycle it is applied, the artefacts used as the model and specification, the type of relation (i.e., Definition 1 or Definition 2), and the goal of verification.

Table 3: Overview of verification techniques.

Verification technique	**Life cycle stage**	**Model artefact**	**Specification artefact**	**Relation type**	**Verification goal**
System conformance checking	Implement	Implementat ion	Prescriptive model	Conformance	System conformance
Conformance checking	Enact	Event log	Prescriptive model	Conformance	Process Conformance
Conformance checking	Diagnose	Event log	Prescriptive model	Conformance	Process Conformance
Conformance checking for repair	Diagnose	Descriptive model	Event log	Conformance	Model conformance
Correctness checking	Design	Model	Design Properties	Compliance	Model compliance
Regulatory compliance checking	Design	Model	Regulations	Compliance	Regulatory compliance
Regulatory compliance checking	Enact	Event log	Regulations	Conformance	Regulatory compliance
Auditing	Diagnose	Event log	Regulations	Conformance	Regulatory compliance

From Table 3, it can be observed that, between all verification techniques, only two relations are compliance relations, and both of these techniques use the business process model as the model for verification. Secondly, out of the other six techniques that have a conformance relation, only four have a conformance related goal. Finally, although three different verification techniques have the goal of regulatory compliance, only one has an actual compliance relation, while the others have conformance relations.

Given these observations, it is clear that there exists a grey area between the use of the conformance and compliance keywords among the verification relations and goals. The main `offenders' are the techniques of *regulatory compliance checking* during enactment and *auditing*. These techniques both define *conformance* relations with the goal of checking regulatory *compliance*. Both these techniques were naturally developed out of the realisation that proving a compliance relation between two specifications (i.e., model and regulations) could only provide so many preventative guarantees, and that runtime data and temporal information is required for definitive and complete results. It is not that these techniques are at fault. They very much prove regulatory compliance while defining a conformance relation. The conformance relation does not, suddenly, become a compliance relation when one has the goal of verifying regulatory compliance, nor does the goal suddenly become verifying regulatory conformance. However, even though the compliance and conformance terms are effectively synonyms in everyday language, it remains especially important that both research and application have clearly defined lines between developed and applied techniques and their related *keywords*.

5.3 A Unified Terminology

To ameliorate the issue, clear boundaries for the use of the conformance and compliance keywords must be established within the context of verification during the life cycle of processes within smart factories. To do so, we must first combine the subset of artefacts used and created during the enactment phase of the life cycle, into a larger process execution artefact. Given this artefact, it is clear correct boundaries can be defined through the use of three keywords instead of two. These keywords are (i) compliance, (ii), conformance, and (iii) regulatory compliance. The result is illustrated in Figure 15, and should help the correct application of the different verification techniques in smart factories. To explain, when we speak of *compliance*, we are applying verification using a specification from the system requirements and the business process model as the model for verification. On the other hand, when we speak of *conformance*, we are applying verification using the business process model with artefacts within the business process execution area. Finally, when we speak of *regulatory compliance*, we are applying verification using the regulations as the specification and artefacts within the business process execution area as the model for verification. Note that we use *compliance* (instead of regulatory compliance) to cover the verification of a model against regulations. Although this creates an overlap, this is not harmful since it correctly refers to compliance on both the relation and the regulatory goal. When verifying the system requirements against a more refined set of such requirements, or a business process model against a more refined business process model, it is also *compliance*.

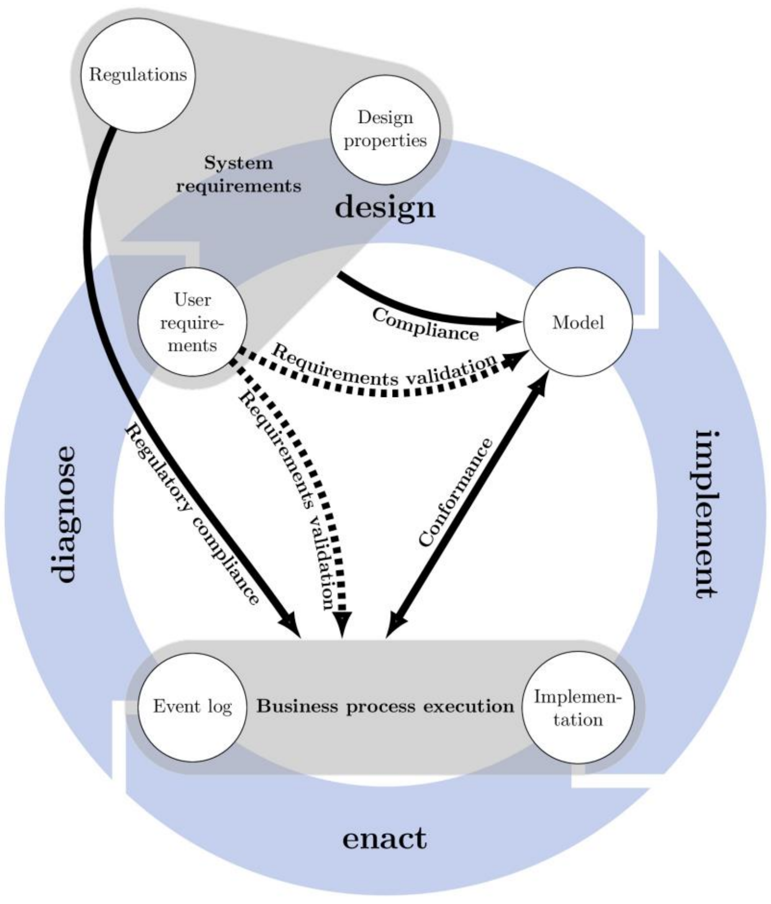


Figure 15: Conformance and compliance during the life cycle of processes.

From this, it is clear that when using these three terms, it introduces clear boundaries that should be used to distinguish between verification techniques. For example, consider an approach that obtains a business process model from an event log using a process mining technique and checks system requirements (e.g., regulations or user requirements) against the obtained model. That is, it obtains a model that describes the business process as it is performed in the real world (i.e., a descriptive model) from observed behaviour of the implementation, and checks it against a specification. In this case, the approach would be a *regulatory compliance* approach when it verifies against regulations, a *compliance* approach when it verifies against design properties, and a *requirements validation* approach when it checks user requirements.

5.4 The Dangers of Applying Techniques to Other Goals

The definition of clear boundaries between available techniques and tools is important for both researchers and practitioners. For researchers, it is not only important to ensure that the right terminology is used when describing their techniques and tools, but also to assist practitioners to select the correct tool for its intended purpose. Furthermore, such boundaries allow researchers to properly position their work, including the use of examples, selection of relevant related work, and evaluating against relevant work. For practitioners, on the other hand, it is important to ensure the validity of the results. That is, to ensure that the applied technique or tool verifies what was intended to be verified and be able to rely on the results and draw correct conclusions from those results. Consequently, more precise terminology allows to select the right portfolio of tools to collectively verify each aspect of the design and its implementation against each aspect of the set of system requirements, including user requirements, design properties, and regulations.

The question, however, remains what the dangers are when techniques appear relevant towards other goals. To do so, we discuss the relevance of some techniques to the goals set for the other techniques. That is, we discuss whether the technique of process conformance checking is relevant to the goal of regulatory compliance. Similarly, we discuss whether the technique of regulatory compliance checking is relevant to the goal of process conformance, and finally, we discuss whether the technique of process conformance checking is always relevant to conformance from a legal point of view. We discuss these questions, highlight any advantages or limitations that such applications yield, and present any analysis gaps that such applications may permit.

5.4.1 Applying Process Conformance to Prove Regulatory Compliance

As the popularity of process mining increased, the idea slowly evolved that proving a *conformance* relation between an event log and a business process model can prove *regulatory compliance*. As such, the use of conformance checking techniques has been suggested as valuable to, for instance, agile compliance management and GDPR. Although technically conformance checking can be applied to prove regulatory compliance, it should be made clear that this approach is not ideal and can only prove regulatory compliance up to some point.

When using this approach, several strict conditions must be met, while results often lead to non-obvious inconclusive outcomes. First, a prescriptive business process model is required to check conformance. Second, this prescriptive model must be proven regulatory compliant using design time regulatory compliance checking (Definition 8). One should be careful to note that, although design time regulatory compliance checking can check prescriptive models, it generally uses descriptive models. Third, the conformance checking must report any unfitting behaviour (i.e., any deviations from the prescribed model). We must stress here that any unfitting behaviour is not necessarily a violation of regulations. It simply means that a deviation was made from the possible executions described by the prescriptive model. As a result, this type of checking effectively denies any form of process flexibility.

Therefore, regulatory compliance can be proven through conformance checking by proving there is no unfitting behaviour. However, it cannot prove that any unfitting behaviour is an actual violation of regulations. One would still require additional regulatory compliance checking or auditing to prove this. In addition, it can only prove regulatory compliance along the control flow perspective, because the design time regulatory compliance checking techniques used to check the prescriptive model only has access to design time information and lacks process enactment information, such as data, resources, multiple instances etc. In this way, the limitations of the preventative measure of design time regulatory compliance checking (Definition 8) is transferred to an approach that in fact has process enactment information.

Although further model annotations of regulations are possible to consider other perspectives than that of the control flow, these approaches edge more towards also doing regulatory compliance checking while conformance checking, than just conformance checking — and would still deny any process flexibility. On the other hand, conformance checking approaches that enable process flexibility by allowing a certain level of unfitting behaviour can never prove regulatory compliance without applying some form of actual regulatory compliance checking. As a result, the approach of using conformance to check regulatory compliance will always remain sub-optimal and should ideally be avoided.

5.4.2 Applying Regulatory Compliance to Prove Process Conformance

The application of regulatory compliance (Definition 8) to prove process conformance may, at first sight, seem completely irrelevant. However, it is possible but requires an unconventional approach. Again, it should be made clear that this approach is not ideal and can only prove conformance up to some point. That is, the approach can only obtain a degree of fitness (i.e., the fraction of behaviour that is in the event log but not possible according to the model) and not a degree of precision (i.e., the fraction of behaviour that is in the model but never observed in the event log). To obtain a degree of fitness of an event log with respect to a process model using regulatory compliance, we must first obtain a declarative specification of the prescriptive business process model. That is, we must obtain a set of declarative rules (e.g., temporal logic expressions) that together describe all possible paths within the business process model.

One example to automatically obtain such a declarative specification includes obtaining an event structure from (sets of) process model(s) and extracting a specification in the form of computation tree logic expressions (van Beest et al. 2019). Once a declarative specification is obtained, execution traces of the business process (captured by the event log) can be evaluated against the declarative specification using formal regulatory compliance verification techniques such as existing model checking tools and packages (Groefsema, van Beest, and Aiello 2018; Groefsema, van Beest, and Armas-Cervantes 2017). To obtain a degree of fitness for an execution trace, or all execution traces within the event log, we can divide the number of satisfied temporal logic expressions by the total number of temporal logic expressions being verified. In this way, the degree of fitness decreases as more temporal logic expressions are violated.

Next to the degree of fitness, results include sets of satisfied and violated temporal logic expressions. Consequently, these results will be difficult to interpret by non-experts. As a result, the approach to use regulatory compliance to check conformance is non-ideal due to partial and difficult to interpret results, and should be avoided.

5.4.3 Applying Process Conformance to Prove Legal Conformance

In a previous section, we gave a short outline how to use what we called process conformance to prove regulatory compliance from the process oriented information systems point of view. In this section, we are going to look at the issue from a legal point of view. First of all, in legal documents there is often no real distinction between compliance and conformance (and, sometimes the two English terms are translated to a single term in other languages). The two terms both generically mean to obey to a set of prescriptions. For instance, consider the proposal for the European Union's Artificial Intelligence (AI) Act. According to the current proposal, AI (and more generally) systems operating in specific sectors have to comply with the Act, as the explanatory text recites:

> Those AI systems will have to *comply* with a set of horizontal mandatory requirements for trustworthy AI and follow *conformity* assessment procedures before those systems can be placed on the Union market.

As we can see, the Act does not differentiate between the model of an AI system and its implementation. Furthermore, the Act seems to indicate that compliance refers to the behaviour of day-to-day operations of the implementation; on the contrary, systems have to obtain conformity certificates before the system is placed on the market or operates in the European Union. Accordingly, conformance certificates are based on the evaluation of the systems before the systems are deployed. This poses the question if process and system conformance as understood in the business process community (as discussed in the previous sections) offer suitable techniques for providing conformance certificates for AI systems against the requirements set by the Act. The answer seems to be negative, since the requirements for conformance certificates appears to be closer to what we called regulatory compliance. Thus, while some of the techniques and methodologies developed for business processes appear adequate for the AI Act, the terminology used to describe them might not correspond to the terminology used by the legal and business communities; therefore, there is risk that process management solutions will not fit for some applications or are evaluated with negative results, and effective techniques not to be adopted, limiting the impact of the for this important market.

5.5 Conclusion

Verification techniques help smart factories to understand their processes, verify correctness against requirements and diagnose potential problems. For smart factories to adopt these techniques, it is important to use the correct keywords to both determine the verification problem to be addressed, and then match the required technical capabilities that can solve the problem.

In the field, and in the broader research community, the keywords of compliance and conformance have often been used interchangeably. However, from a technical point of view, they have been proposed with a different meaning. In general, compliance and conformance are two types of verification of systems, relating two artefacts. Effective methods for one verification type, however, cannot guarantee a successful verification for another. Consequently, there is a need for a uniform set of definitions and unified terminology.

In this section, we first provided comprehensive definitions of the two notions and their related activities. We then proposed such a unified terminology to enable adoption of the techniques in smart factories. Finally, to avoid potential problems during adoption, we explored the dangers of applying specific techniques to goals that they were not intended for.

6 Enabling Interoperability using Git

Traditional software development methodologies are not enough to fulfil business requirements nowadays. Adaptation of Agile practices enables flexibility, efficiency, and speed of the Software Development Life Cycle (SDLC), which is attracted by software development companies (Dzamashvili Fogelström et al. 2010). As per the Agile manifesto (Beck et al. 2001), the twelve principles have defined the integrity of processes and practices and Agile Project Management, which applied to Extreme Programming (XP), Scrum, and Kanban methodologies. Implementation of "Continuous Integration Continuous Delivery," CICD, pipeline on agile has enabled fast delivery of software (Olsson, Alahyari, and Bosch 2012) and increased productivity. In the year 2000, Martin Fowler (Fowler 2006) presented the idea of Continuous Integration (CI), and later J. Humble and D. Farley (Humble and Farley 2010) extended these ideas into the approach of Continuous Delivery (CD) as a concept of a deployment pipeline. The main benefits of CI practices are reducing the risk and making software bug free and reliable, which removes the barrier of frequent delivery. Accelerated time to market, improved product quality, improved customer satisfaction, reliable release, improved productivity, and efficiency are key benefits (Chen 2015), motivating companies to invest in CD (Arachchi and Perera 2018). A considerable benefit of having a CICD pipeline is a separation of responsibilities that will help team members to focus on their part while the CICD pipeline takes care of integration and delivery, which results in rapid releases.

6.1 Agile Software Development to CICD

Individuals and interactions over the processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan are the critical agile values that are motivated to follow the twelve principles defined in the Agile manifesto (Beck et al. 2001).

Manually delivering the software application is one of the hardest things in SDLC. It takes more time, it needs experts in the field who can handle operational tasks, mistakes are inevitable, and teams have more responsibility. CICD encourages and motivates teams to deliver software frequently due to automated builds and deployments. With CICD practices, organizations deploy software updates 10, 100 or even 1000 times a day (Savor et al. 2016).

6.1.1 CICD Pipeline

When an organization tries to adopt the CICD pipeline, they may not be able to adopt it at once. First, they have to practice CI to adopt CD. When moving from CI to CD and then Continuous Delivery to Continuous Deployment, this pipeline has reduced the manual process execution, and finally, the entire process becomes automated. The main difference between Continuous Delivery and Continuous deployment is automation at production deployment.

6.1.2 Continuous Integration

Continuous Integration is a software development practice where team members integrate their work regularly and automate build, test, and validation. It helps to find and fix bugs quickly and improve software quality. Krusche et al. (Krusche and Alperowitz 2014) used Rugby, which is an agile process model carried out by students. Continuous Integration helped improve the code quality by about 50% and helped to find and fix broken commits more than 65% faster. About 70% of students have claimed that Continuous Integration enabled them to improve the overall development workflow. The main components of Continuous Integration are the source repository, version control system, and CI server. More frequent commits to a shared codebase, maintaining a single source repository, automating builds, and automating testing are challenges when following CI practices (Stacy et al. 2017). Build automation, Code stability, Analytics, CD enablement, faster releases and cost saving, improved productivity and code quality are realized benefits when adopting CI practices (Kumbhar, Shailaja, and Anupindi 2018; Stacy et al. 2017).

6.1.3 Continuous Delivery

Continuous Delivery is the ability to get all types of changes, including new features, configuration changes, bug fixes, and experiments, into production safely and quickly in a sustainable way (Humble and Farley 2010). Krusche has introduced CD into multi-customer project courses and evaluated its usage, experience, and benefits (Krusche and Alperowitz 2014). There is a rapid trend in investment in CD due to its benefits, such as improving productivity and efficiency, reliable releases, customer satisfaction, accelerated time to market, and making the right product (Chen 2015).

6.1.4 Continuous Deployment

Continuous Deployment means the committed changes are production-ready and to be applied in production automatically (Ariola and Dunlop 2015; Stacy et al. 2017). Many organizations use a continuous deployment automation approach to make their software development life cycle more efficient (Savor et al. 2016). Rahman et al. point out that continuous deployment has sped up the processes in agile methods noting Facebook, GitHub, Netflix, and Rally Soft as organizations that use continuous deployment efficiently on their production deployments (Rahman et al. 2015).

6.2 Git

Git is a distributed revision control system available on all mainstream development platforms through a free software license. An important difference between Git and its older ancestors is that it elevates the revisions of software to first-class citizens. Developers care deeply about software revisions, and Git supports this by giving each developer a complete private copy of the software repository and numerous ways to manage revisions within its context. The ability to associate a local repository with numerous remote ones allows developers and their managers to build various interesting distributed workflows, most of which are impossible to run on a traditional centralised version control system. The local repository also makes Git responsive, easy to set up, and able to operate without Internet connectivity (Spinellis 2012).

6.3 Tracking Artefacts with Git

Git is mainly used for software code. However, it is not the only use case and can also help with other types of artefacts. Git can be used to manage them separately and in various combinations for different use cases, such as maintaining lab notebooks, presentations, datasets, and manuscripts. The following artefact descriptions are derived from an article on how Git can help with reproducibility and transparency (Ram 2013).

6.3.1 Manuscripts and Notes

Version control can operate on any file type, including the ones most commonly used in academia, such as Microsoft Word. However, since these file types are binary, Git cannot examine the contents and highlight sections that have changed between revisions. In such cases, one would have to rely solely on commit messages or scan through file contents. The full power of Git can best be leveraged when working with plain-text files. These include data stored in non-proprietary spreadsheet formats (e.g., comma-separated files versus XLS), scripts from programming languages, and manuscripts stored in plain text formats (LaTeX and markdown versus Word documents). With such formats, Git tracks versions and highlights which sections of a file have changed. In Microsoft Word documents, the track changes feature is often used to solicit comments and feedback. Once those comments and changes have either been accepted or rejected, any record of their existence disappears forever. When changes are submitted using Git, a permanent record of author contributions remains in the version history and is available in every repository copy.

6.3.2 Datasets

Git can be a good fit for small datasets. These include manually entered data via spreadsheets, recorded as part of observational studies, or retrieved from sensors. With each significant change or addition, commits can record a log of those activities (e.g., "Entered data collected between 12/10/2012 and 12/20/2012" or "Updated data from temperature loggers for December 2012"). Over time, this process avoids the proliferation of files, while the Git history maintains a complete provenance that can be reviewed at any time. When errors are discovered, earlier versions of a file can be reverted without affecting other assets in the project.

6.3.3 Statistical Code and Figures

In addition to software development, Git can also be used for analytical codes. When data are analysed programmatically using software such as R or Python, code files start small and often become more complex over time. Somewhere along the process, inadvertent errors such as misplaced subscripts and incorrectly applied functions can lead to severe errors. When such errors are discovered well into a project, comparing versions of statistical scripts can provide a way to quickly trace the source of the problem and recover from them.

Similarly, figures in documentation often undergo multiple revisions before resulting in a final version that gets published. Without version control, one would have to deal with multiple copies and use imperfect information, such as file creation dates to determine the sequence in which they were generated. Without additional information, figuring out why specific versions were created becomes more difficult. When figures are managed with Git, the commit messages (e.g., "Updated figure in response to Ethan's comments regarding the use of normalized data.”) provide an unambiguous way to track various versions.

6.3.4 Complete Manuscripts

When all of the above artefacts are used in a single effort, such as writing a manuscript, Git can collectively manage versions in a powerful way for both individual authors and groups of collaborators. This process avoids the rapid multiplication of unmanageable files with uninformative names as illustrated by the famous cartoon strip Ph.D. Comics (Figure 16).

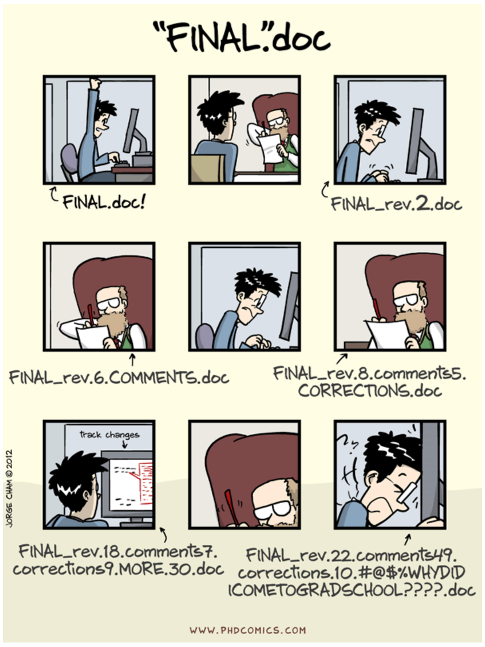

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Figure 16: Manual Versioning Meme.

6.4 GitOps

In short, GitOps is doing all the best practices of the infrastructure as code correctly. This concept defines the infrastructure as code instead of manually creating it to improve reproducibility and replicability. Note that infrastructure as code evolved into defining not only infrastructure but also network as code, policy as code, configuration as code, and security as code, which are called X as code.

For example, instead of manually creating servers and networks and all the configuration around them on AWS and creating Kubernetes clusters with particular components, define all these as code, i.e., Terraform or Ansible code and Kubernetes manifest files. Therefore, we will have many YAML or other definition files describing the infrastructure, the platform, and their configurations.

6.5 Working with X as Code

DevOps engineers will probably create all the required files locally on their machines. Then, they will try to test their codes. Finally, if the test passes successfully, they will execute the codes also from their computer. So all these files are stored locally on their computer. Sometimes, they may even create a Git repository for their infrastructures' code and store all these files on Git. Consequently, they have a version control system for the infrastructure code, and other team members can also fetch the code and collaborate.

However, when they make any changes to the code, they may not have a defined procedure like Pull Requests. They may have only a main branch, and everybody commits directly to it. Therefore, there are no code reviews and no collaboration on the changes. Moreover, when they commit X as Code changes into the repository, there will be no automated test to detect invalid YAML files, typos, wrong attribute names, at an earlier stage. The code changes may break the infrastructure or an environment without proper testing.

Once the changes are done, how do they apply them to the actual infrastructure or a platform? How do they execute them? They will do it manually from their machines by executing "kubectl apply" or "terraform apply" comands. So to execute the code changes, each team member must access the

Kubernetes cluster or AWS infrastructure to apply changes from their local machine. This can make it hard to trace who executed what on the remote servers a have a history of changes applied to the infrastructure. Therefore, if someone makes any mistake in the code, they will know about these problems only once it is applied.

Consequently, even though we have an Infrastructure as Code, which already has many benefits, our process is still mostly manual and inefficient. This is where the concept of GitOps comes to treat the Infrastructure as Code the same way as the application code.

6.6 Working of GitOps

In GitOps practice, we have a separate repository for the X as code project with a complete DevOps pipeline. As the initial setup, X as code is hosted on a Git repository where it is version controlled and allows team collaboration. When you make changes, instead of just pushing to the main branch, you go through the same pull request process as you do for the application code. Therefore, anyone in the team, including junior engineers, can create a pull request to make changes to the code and collaborate with other team members on that pull request. For these changes, you will have a CI pipeline that will validate and test the configuration files just like you test application code changes. After testing these commits, other team members can approve the final changes. These other team members could be developers, security professionals, or senior operations engineers who will review and approve the pull request. So, the changes will only be merged back into the main branch after tests and reviews.

Afterwards, the changes will be deployed to the environment through a CD pipeline, whether changing something in the Kubernetes cluster or updating the underlying infrastructure. Consequently, you have an automated and more transparent process that produces high-quality infrastructure. This enables multiple people to collaborate on the changes, and things get tested rather than one engineer manually doing everything from their laptop that others do not see or cannot review.

6.6.1 Automatically Applying Changes to the Infrastructure

We have two ways of applying the changes in the main branch to the infrastructure in GitOps practices: push- and pull-based deployments. Push-based deployments are what we traditionally know from the application pipeline. When an application is built, the pipeline executes a command to deploy the new application version into the environment. Jenkins and Gitlab CICD are two example tools that implement the push-based deployment mechanism.

In pull-based deployment, we have an agent installed in the environment, like the Kubernetes cluster, that actively pulls the changes from the Git repository. The agent will regularly check the state of the X as Code repository and compare it to the actual state of the environment where it is running. If it sees a difference in the repository, it will pull and apply these changes to get the environment from the actual state to the desired state defined in the repository. Examples of GitOps tools that work with the pull-base model are Flux CD and Argo CD, which run inside the Kubernetes cluster and sync the changes from the Git repository to the cluster.

6.6.2 Rollbacks with GitOps

When you have the version control for your code and the changes in the repository are automatically synced to the environment, you can easily roll back your environment to any previous state in your code. This is another significant advantage of using GitOps. For example, if changes break the environment and the cluster does not work anymore, the environment can get back to the last working state by executing "git revert" to undo the latest changes.

6.6.3 Advantages of GitOps

Generally, this means that instead of spreading the X as code in different locations and machines, everything is stored centrally in a Git repository and the environment is always synced with what is defined in that repository. Therefore, the Git repository becomes the single source of truth for the environment, which makes managing the infrastructure or the platform much easier.

Moreover, an important additional benefit is that GitOps also increases security. There is no need to give direct infrastructure access to everyone in the team who wants to apply changes because it is the CD pipeline that deploys the changes, not individual team members from their laptops. However, team members can propose changes to the infrastructure in the Git repository through pull requests. Once it is time to merge that pull request and apply those changes, we can have a narrower group of people who are allowed to approve and merge those changes into the main branch. As a result, we have fewer permissions to manage and a more secure environment.

6.7 Conclusion

The agile manifesto, the de facto standard project management theme for software development, encourages rapid delivery. It cannot happen without efficient procedures and automation, such as continuous integration and continuous delivery (CICD) pipelines. Git is the core component in modern software development for storing almost everything. It enables auditing and collaboration. A recent concept that uses Git in the best possible way is GitOps which is an X as code with version control, pull requests, and CICD pipelines. It is noteworthy that we applied all these best practices and many more features for deploying applications and infrastructure in our latest project, ECiDA[1]. Its goal is to bring ease for developers and data scientists to deploy their applications in any environment without requiring knowledge about the underlying infrastructure while focusing more on the logic of their applications.

[1] https://www.cs.rug.nl/ds/Research/ECiDA

7 Interoperability in IoT using Event Processing – A Trade-Off between Quality and Privacy

The Internet of Things (IoT) is a well-known paradigm that has attracted enormous interest from academia and commercial sectors in the last decades. In the IoT environment, billions of devices (e.g., sensors) are employed to perform various tasks (e.g., sensing a phenomenon) and produce a huge amount of data. In Figure 17, the growing utilisation of IoT devices is demonstrated, confirming the concern of required resources to store and analyse their generated data.

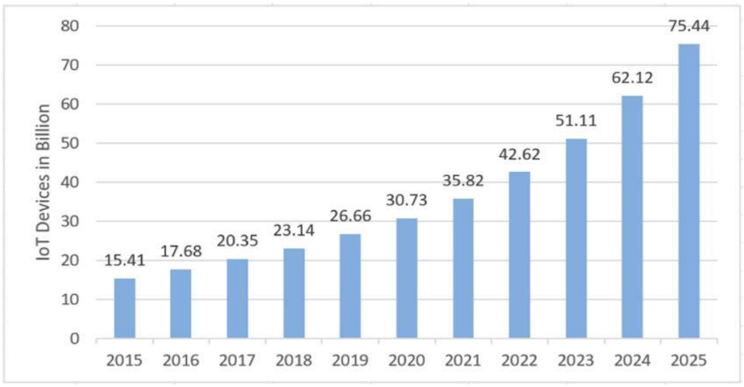


Figure 17: The utilization of IoT devices in the environment (Al Hayajneh, Bhuiyan, and McAndrew 2020).

On the one hand, the results obtained from analysing these data can generate more valuable insights if the analytic system attempts to execute its tasks as soon as possible with minimum delay, also called real-time analysis. To be more precise, the data should be analysed in less than seconds to make the insights preventive or productive. Otherwise, the derived outcome can be actionable or reactive. If we store the data in databases and plan to analyse them after some hours, the only option could be the historical analysis. Figure 18 presents the value of data based on the time spent to be analysed.

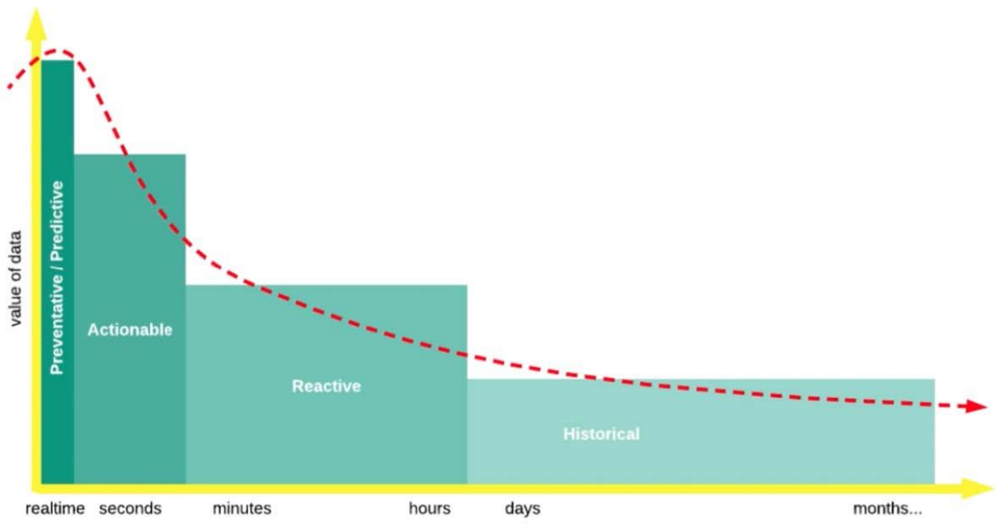


Figure 18: The data value based on the analysis time (Nemer 2022).

One of the prominent paradigms that provide sufficient means to analyse data in real time is Complex Event Processing (CEP). In this concept, the raw data that is more important for the application is transformed into the primary event (e.g., a high temperature in readings of a temperature sensor). In this way, a meaningful reduction is happening in the generated data. This also reduces the number of computing resources required to analyse the data. The produced events form primary event streams that are sent to CEP engines to be analysed in real-time. A situation of interest (e.g., detecting shoplifting) can be submitted to the system as a continuous query which is interpreted as a complex event. The CEP system presents each complex event by a pattern of primary events (e.g., a sequence of entering the shop, picking a product, not paying the fee, exiting the shop) and its main goal is to apply the query pattern over streams of primary events that comes from the sensing deployment to detect the occurrence of those situations. Once a pattern match is detected over the stream, a complex event is generated.

The CEP paradigm is extended to satisfy the requirements of distributed systems by introducing Distributed Complex Event Processing (DCEP) in which a controller is logically centralised but physically distributed as shown in Figure 19. The controller is able to perform adaptation in three places: 1) by rewriting the user queries; 2) by adjusting the placement of operators on the available computing resources; and 3) by reconfiguring the sensing deployment.

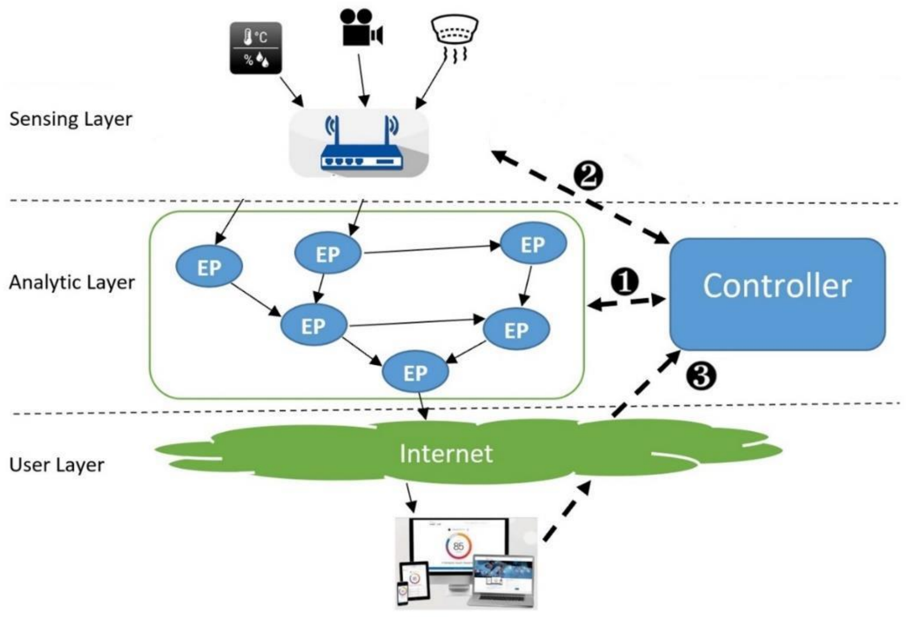


Figure 19: The layering presentation of DCEP systems.

Recently, the interoperability concept has gained more interest since it has the potential to boost the performance of systems by sharing data and models. This will eliminate the need to perform a task multiple times in different systems and consequently has tangible advantages for all systems involved in interoperation. Due to the distributed nature of DCEP systems, such benefits can also be imaginable by interconnecting these systems. For example, data sharing can impact the Quality of Service (QoS) demands, such as end-to-end latency, by reducing the time for processing data. In other words, if a DCEP system previously analysed the data of a person to achieve some specific results (e.g., performing a blood test), its outcome can be reused in another system that aims to perform a similar analysis (e.g., a blood test is required one week later in a trip). The provided reusability will reduce the required resources and time while supporting QoS-aware analysis (e.g., a faster diagnosis results in better treatment).

On the opposite side, each cooperation among systems raises privacy preservation issues. For instance, how systems can provide trustable communication links and prevent misuse of their shared data. Although DCEP systems have been working on supporting privacy concerns in recent years, some gaps still require further research. Considering both quality and privacy with the same importance and establishing a trade-off between these two aspects requires different aspects to be considered, including answers to the following questions:

- How to provide an access control mechanism or a trust management system to preserve privacy?
- What would be a concrete description of quality and privacy requirements?
- How all involved entities can benefit from interoperability?
- Which adaptation strategies can be employed to maximise the benefits?

Providing feasible solutions to these research questions makes the interoperability paradigm more practical in IoT applications when using DCEP analytic systems.

7.1 A Trade-Off between Quality and Privacy

In this section, in order to provide a trade-off between privacy and quality, we first elaborate on the definition of each of these topics separately in DCEP systems and then present the possible solutions to provide interoperability.

7.1.1 Quality

IoT applications operate on resources and streams of events that are highly dynamic due to the unstable nature of the real world and its conditions. Nevertheless, the properties of these data sources should be updated continuously to deliver more accurate and error-free data and events to corresponding applications. IoT service qualities are vulnerable to the changes happening in the environment. For example, the accuracy of a sensor could be impacted by its battery level, environmental weather conditions, and also air temperature (Gao et al. 2016). Besides, analysing event streams gathered from the IoT environment brings several problems such as the trustworthiness of data sources and their heterogeneity as well as extracting up-to-date information from real-time data streams (Kolozali et al. 2019). The literature can be categorised into four different groups as follows.

7.1.1.1 Quality of Data (QoD)

This category exhibits the research that mainly focuses on designing algorithms to evaluate and increase the level of data quality before sending them to the CEP system. When data is collected from the environment, due to cyber-physical attacks happening in the wireless medium, it could be possible that the data contain anomalies like missing data, redundant data, data failure, data outliers, touched data, etc. Data pre-processing is a way of enhancing data quality. It means data has to be validated before being analysed. In the other words, useless data such as records with missing fields, data outliers, irrelevant data, inconsistent data, and duplicate data have to be removed from the data stream since processing them is surely a waste of time.

7.1.1.2 Quality of Event (QoEv)

In an event-based system, events (both primary and complex) could be detected with various quality levels. The quality of event detection can be achieved by two factors, one is detection delay and the other is the detectability of events. Also, specifying the metrics by which such quality specification could be determined is of great importance. In fact, to evaluate the aggregated quality of an event, quality metrics such as latency, price, energy consumption, bandwidth consumption, availability, completeness, accuracy, and security should be taken into consideration (Gao et al. 2014).

7.1.1.3 Quality of Service (QoS)

In IoT applications, service qualities are vulnerable to the changes happening in the environment. For example, the accuracy of a sensor could be impacted by its battery level, environmental weather conditions, and also air temperature. In order to deal with such problems, it might be beneficial to adapt to the CEP system over the environmental changes in terms of quality measures requested by users. It could be possible by changing the CEP model adaptively when the system realizes service failures and constraint violations of user requirements. Besides, to satisfy user requirements, events and their patterns from various CEP services can be reused in another CEP system by employing an event reusability hierarchy. It is exactly the point that the interoperability paradigm can be used to benefit more the event processing systems (Sodhro et al. 2020).

7.1.1.4 Quality of Experience (QoE)

In order to increase the degree of satisfaction in terms of meeting user requirements, Quality of Experience (QoE) has recently become a trend in various subjects of IoT networks. Although in the traditional mechanisms of QoE, filling in questionnaires was the common way of ensuring user satisfaction, current methods rely on overseeing the interactions of the users with the system by taking advantage of current advances in networking and communication technologies that provide the system with observable data (Zhou et al. 2019). Finding the correct factors for evaluating the satisfaction of users is largely domain-specific and it would be a hard problem to represent a generalized framework for the definitions and evaluation of QoE metrics. However, applying the constraints and preferences of the user to the complex event processing procedure including planning, placement, and mapping operators to available hosts would lead to appropriately meeting the quality of experience of the user.

7.1.1.5 A Quality Evaluation Summary

As we can see from the previous four categories, each of the publications in the literature has mainly focused on one or two steps of quality evaluations while it seems that for an appropriate query processing as well as proper reaction to environment dynamics, the proposed system should consider the feedback that is coming from every part of the system, ranging from sensors to users.

7.1.1.6 Quality Monitoring

Considering the different levels of quality evaluation explained in the previous sections makes it clear that the quality should be monitored at each step. The input data can be assumed to be of insufficient quality if not accurate, precise, fresh, or truthful. Events are also evaluated as inadequate quality if they do not hold a certain level of confidence, are received out of order, are wrongly detected, or are not detected at all. In addition, the insights derived from measured quality have an important role in the adaptation decisions in adjusting any of the three adaptation models described in the previous sections. To measure the quality level in each step, quality agents should be placed in each layer (i.e., sensing, analytic, and user layer).

7.1.1.7 Quality Requirement Expression

One of the important parts of a quality-aware processing system is to what extent it can satisfy the quality requirements of users. To better understand the quality expectations of users, an analytic system should provide easy-to-use solutions to express the quality demands. This means that those quality metrics that are feasible to measure should be considered in the process of quality requirement elicitation. The first step is to acquire the user needs by quality expressions which determine a threshold for each specified quality metric, e.g., an accuracy level of more than 90 per cent. It might also be possible to extend the traditional quality specification by proposing dynamic thresholds which vary based on another factor, e.g., time. This way, more complex quality requirements can be expressed by the users. In addition, if the specified quality requirements are not feasible to satisfy, the DCEP system needs to rewrite the quality requirement models or in some cases adjust the sensing deployment to meet the requirements.

7.1.1.8 *A Quality Aware DCEP system*

In this section, we demonstrate our proposed solution for quality monitoring in DCEP systems, presented in Figure 20. We utilised a publish/subscribe system for communication between different entities. Producers generate primary events and consumers submit their situations of interest as queries.

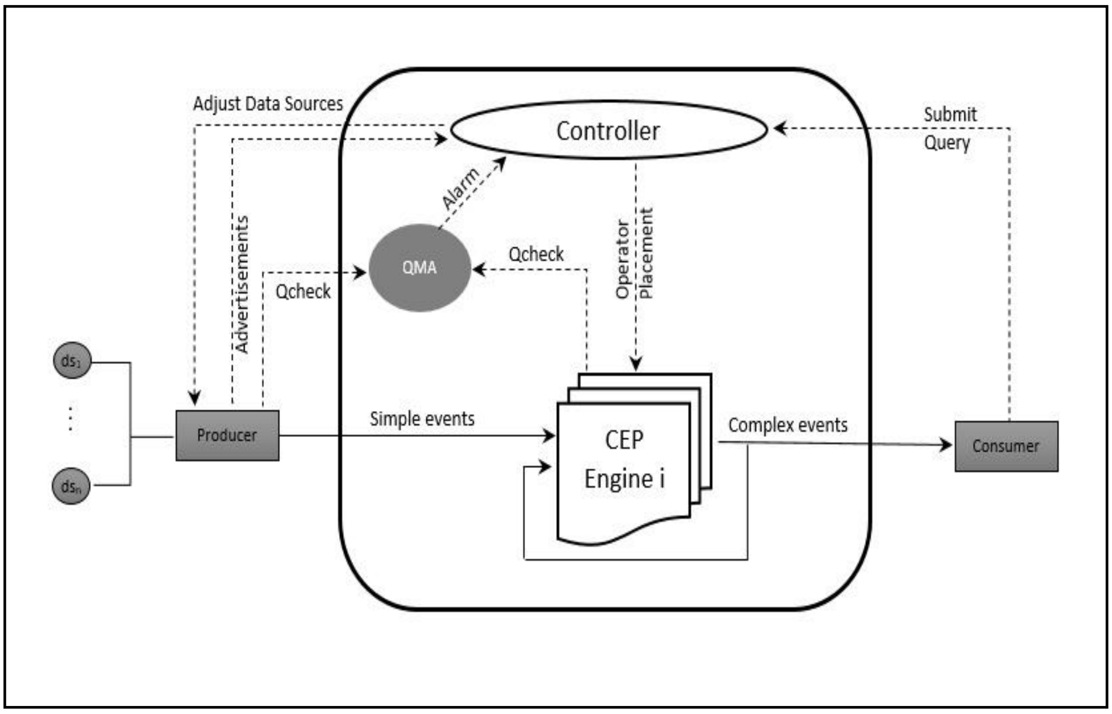


Figure 20: The Proposed Solution for Quality-Aware DCEP.

The Quality Management Agent (QMA) is responsible for evaluating the quality in different places and producing quality-related alarms that help the controller maintain the quality at a satisfactory level. In some cases, the controller will perform adjustments to the sensing deployment to provide sufficient quality for the submitted queries.

7.1.2 Privacy

In most IoT applications, people who share their data with the analytic system (i.e., data owner), are not aware of possible risks that threaten their data. However, if a person is kept aware of these risks, they worry about the violation of data privacy. That is why the users of the system usually refuse to deliver as much required data to the system. A possible solution to this problem is to provide a trustable system that satisfies the privacy requirements of data owners. The easiest way to establish such a trustable mechanism is to employ an access control technique that determines the access level of each entity involved in data sharing in the interoperability paradigm.

However, to provide privacy-aware interoperability, a simple Data Access Control (DAC) mechanism cannot satisfy the necessary requirements, because access to data or information derived from it should be granted once a data access request is created by the second DCEP system, e.g., in the blood test example, the access to the test results should be given to the second hospital at runtime in a dynamic way. Therefore, a DAC mechanism needs to be empowered by a dynamic authorisation component. This way, a dynamic access control technique can be applied to control the data access for all the entities involved in the DCEP systems.

7.1.2.1 Attribute-Based Access Control

Among all types of DAC mechanisms, Attribute-Based Access Control (ABAC) system is more suitable for providing interoperability between the DCEP systems due to the comprehensive information this type of DAC mechanism can provide for the system which helps to prevent privacy attacks.

In other words, an ABAC mechanism is a logical DAC technique where the permission to perform data sharing is granted based on the attributes associated with multiple entities including the data owner, the shared data, the user who requests the access, the type of action this user is supposed to do on the requested data, and in some cases, the environment condition in which the data sharing will occur. These attributes will be investigated against sharing policies, rules, or relationships that explain which operations are permitted on a given set of attributes. Figure 21 presents the methodology behind ABAC systems.

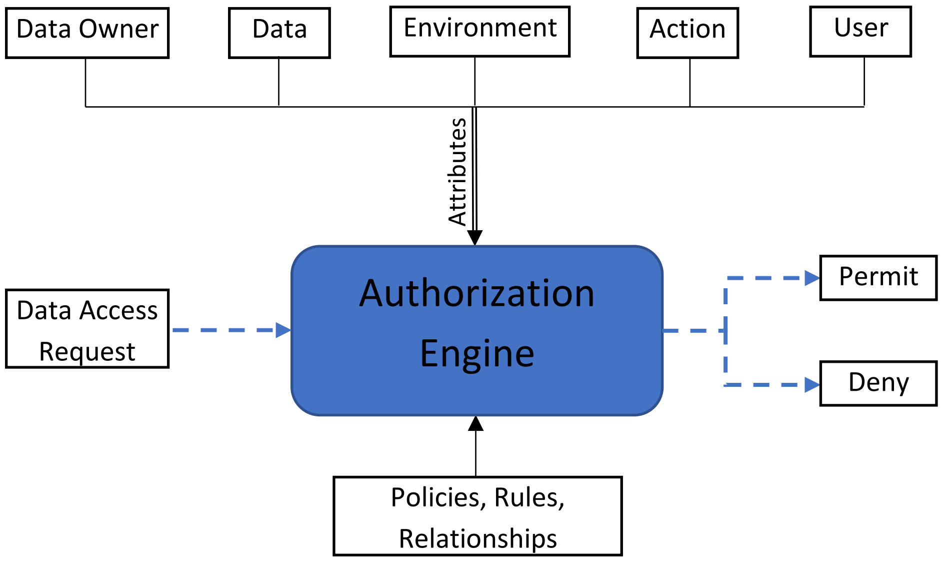


Figure 21: The Layering presentation of DCEP systems.

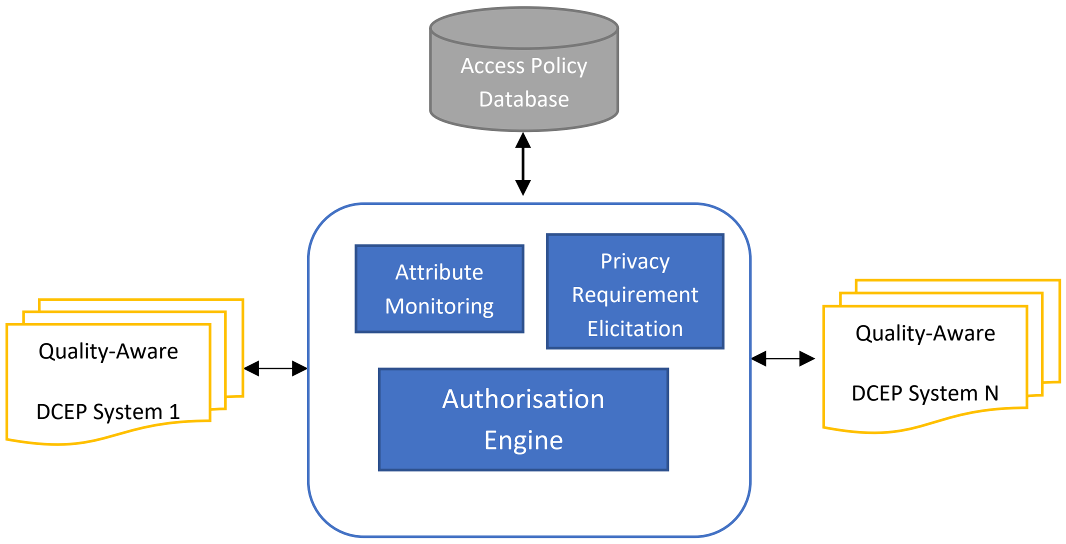
7.1.2.2 Privacy Requirement Expression and Elicitation

Although the GDPR regulates IoT services in acquiring privacy consent from data owners, it is still difficult to achieve consent for data sharing between the data processing system and the data owners. Therefore, for the involvement of the users and data owners, considering the following requirements will improve the privacy demands elicitation (Stach and Steimle 2019).

a) Simplicity: The privacy requirement expression should be as simple as possible for both users and data owners.
b) Awareness: Each data owner should be made aware of potential privacy risks for their shared data.
c) Customization: Due to the different privacy demands of individuals, the privacy requirements of data owners should be customised based on their perspective on privacy.
d) Categorisation: This will support the efficient management of privacy requirements in the elicitation procedure.
e) No third parties: Due to the interests of other parties, the elicitation process might be influenced by involving third parties.

7.1.3 A Quality-Privacy Trade-off

In this section, we explain our proposed solution to establish a trade-off between quality and privacy. In Figure 22, we demonstrate the components involved in establishing the quality-privacy trade-off.



Access Policy Database
Attribute Monitoring
Privacy Requirement Elicitation
Authorisation Engine
Quality-Aware DCEP System 1
Quality-Aware DCEP System N

Figure 22: The Proposed Architecture for Quality-Privacy Trade-off.

Considering the previous sections, our proposed architecture includes solutions to support both quality and privacy. For the former, the DCEP system evaluates the quality of the sensed data and generated primary and complex events, as well as monitors the status of the sensing deployment. For the latter, an ABAC mechanism is employed to support a privacy-aware communication and data-sharing scheme between the quality-aware DCEP systems. In the proposed ABAC approach, not only are the privacy demands of the data owner considered using the Privacy Requirement Elicitation component, but also the acquired attributes from different entities are continuously monitored to perform the authorisation process always according to the up-to-date information. It should be noted that the Access Policy Database is also kept up-to-date since it has a key role in the authorisation decisions.

7.2 Conclusion

In this section, we provided an overview of the potential options for applying interoperability on DCEP systems. We discussed the advantages and disadvantages of employing this paradigm while taking into account a trade-off between quality and privacy. We detailed the fact that although by providing interoperability, DCEP systems might benefit from improving the QoS-related metrics, the potential risks might compromise the privacy of shared data. In the end, by employing an Attributed Based Access Control mechanism among Quality-Aware DCEP Systems, we proposed an architecture and discussed its potential that has the capabilities to overcome the disadvantages of applying interoperability on DCEP systems.

8 Conclusion

Virtual factories are abstractions of real factories that allow for the simulation and analysis of various production processes and systems and can be used for a variety of purposes, including production planning, process optimisation, training, and prototyping. Virtual factories rely on a combination of advanced digital technologies, such as Artificial Intelligence, the Internet of Things, and big data analytics, to optimise manufacturing processes in order to increase productivity and quality. The realisation of virtual factories requires interoperability at many different levels, including data, models, services, assets and processes, to ensure a straightforward compatibility between the machines, products, processes, related products and services, as well as any descriptions of those that comprise these virtual environments. This deliverable discussed several challenges related to interoperability within the context of virtual factories and smart manufacturing.

Section 2 explored Manufacturing Execution Systems for building an industrial software layer for virtual factories. These systems can effectively manage the distributed manufacturing resources of the virtual factory by integration with the virtual manufacturing assets discovery, combination, and management services. Moreover, by combining the production plan information of ERP/MES, it can be used to evaluate the performance of the VF production lines, to continuously optimise and improve the performance of VF production lines, and to realise manufacturing business innovation. Another important function of MES is to provide related management functions for quality management, for which CMM is generally one of the main methods. The CMM measurement report (format and content) provided by the CMM software is usually fixed. If the required data used for quality analysis cannot be obtained directly from the report, it should be solved in other ways, e.g., manual calculation, which may introduce calculation errors. By collecting real-time measurement raw data from the CMM software, it provides more abundant inspection data for quality analysis.

Section 3 set out a conceptual basis and a set of requirements for interoperable digital twin simulation, concluding with a strong motivation for supporting interoperable simulation of digital twin configurations through the use of federated simulation. The interface required for federated simulation is small, maps almost directly upon the commonly used Discrete Event Simulation model and allows other simulation approaches to be supported easily while providing minimal restrictions upon simulators themselves. The framework presents a common ground to enable the interoperability of simulations in the context of Industry 4.0 processes. The assumptions made and restrictions posed are minimal, and the base communication constructs used for simulation provide a sound starting point for simulation frameworks to add support for federated digital twin simulation.

Section 4 considered the use of automatic service composition in the context of smart manufacturing. Composition techniques offer many possibilities in smart manufacturing. The intuition is that, like a Web service, a digital twin, which is a fundamental concept in smart manufacturing, can be described as a stateful automaton and, as a consequence, digital twins can be combined following approaches that have been proposed to combine Web services aiming at a specific goal. Digital twins are considered key components in smart manufacturing. They bridge the virtual and real world with the goal of modelling, understanding, predicting, and optimising their corresponding real assets. Such powerful features can be exploited in order to optimise the manufacturing process. Web service composition and Markov Decision Processes (MDPs) can be combined together to automatically propose an assignment of devices to manufacturing tasks. A stochastic service composition was described, in which the services are also allowed to have stochastic behaviour and rewards on the state transitions. This assignment takes into account the uncertainty typical of the manufacturing scenario, thus overcoming the limitations of approaches based on classical planning. Obtained policies are proven to be optimal with respect to cost and quality and are continuously updated in order to adapt to an always-evolving scenario.

Section 5 discussed how verification techniques help smart factories to understand their processes, verify correctness against requirements and diagnose potential problems. For smart factories to adopt these techniques, it is important to use the correct keywords to both determine the verification problem to be addressed and then match the required technical capabilities that can solve the problem. In the field, as well as in the broader research community, the keywords of compliance and conformance have often been used interchangeably. In general, compliance and conformance are two types of verification of systems, relating two artefacts. Effective methods for one verification type, however, cannot guarantee a successful verification for another. Therefore, we provided comprehensive definitions of the two notions and their related activities. We then proposed a unified terminology to enable the adoption of the techniques in smart factories. Finally, to avoid potential problems during adoption, we explored the dangers of applying specific techniques to goals for which they were not intended.

Section 6 considered interoperability at the development level. Rapid delivery cannot be accomplished without efficient procedures and automation, such as continuous integration and continuous delivery (CICD) pipelines. . A considerable benefit of having a CICD pipeline is a separation of responsibilities that will help team members to focus on their part while the CICD pipeline takes care of integration and delivery, which results in rapid releases. Git is the core component in modern software development for storing almost everything. It enables auditing and collaboration. A recent concept that uses Git in the best possible way is GitOps, which is an X as code with version control, pull requests, and CICD pipelines.

Section 7 provided an overview of the potential options for applying interoperability on DCEP systems. We discussed the advantages and disadvantages of employing this paradigm while taking into account a trade-off between quality and privacy. We detailed the fact that although by providing interoperability, DCEP systems might benefit from improving the QoS-related metrics, the potential risks might compromise the privacy of shared data. In the end, by employing an Attributed Based Access Control mechanism among Quality-Aware DCEP Systems, we proposed an architecture and discussed its potential capabilities to overcome the disadvantages of applying interoperability on DCEP systems.

9 References

Aalst, W.M.P. van der, A.H.M. ter Hofstede, and M. Weske. 2003. ‘Business Process Management: A Survey’. In *Business Process Management*, 1–12. Springer-Verlag.

Aivaliotis, P., K. Georgoulias, and G. Chryssolouris. 2019. ‘The Use of Digital Twin for Predictive Maintenance in Manufacturing’. *International Journal of Computer Integrated Manufacturing* 32 (11): 1067–80. https://doi.org/10.1080/0951192X.2019.1686173.

Arachchi, S.A.I.B.S., and Indika Perera. 2018. ‘Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management’. In *2018 Moratuwa Engineering Research Conference (MERCon)*, 156–61. https://doi.org/10.1109/MERCon.2018.8421965.

Ariola, Wayne, and C Dunlop. 2015. ‘DevOps: Are You Pushing Bugs to Your Clients Faster’. In *Thirty-Third Annual Pacific Northwest Software Quality Conference, World Trade Center Portland, Portland, Oregon*, 12–14.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, et al. 2001. ‘Manifesto for Agile Software Development’.

Beest, Nick R T P van, Heerko Groefsema, Luciano García-Bañuelos, and Marco Aiello. 2019. ‘Variability in Business Processes: Automatically Obtaining a Generic Specification’. *Information Systems* 80: 36–55.

Berardi, Daniela, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. 2005. ‘Automatic Composition of Transition-Based Semantic Web Services with Messaging’. In *Proceedings of the 31st International Conference on Very Large Data Bases*, 613–24. VLDB ’05. Trondheim, Norway: VLDB Endowment.

Berardi, Daniela, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. 2003. ‘Automatic Composition of E-Services That Export Their Behavior’. In *Service-Oriented Computing - ICSOC 2003*, edited by Maria E. Orlowska, Sanjiva Weerawarana, Michael P. Papazoglou, and Jian Yang, 43–58. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-24593-3_4.

Brafman, Ronen I., Giuseppe De Giacomo, Massimo Mecella, and Sebastian Sardina. 2017. ‘Service Composition in Stochastic Settings’. In *AI*IA 2017 Advances in Artificial Intelligence*, edited by Floriana Esposito, Roberto Basili, Stefano Ferilli, and Francesca A. Lisi, 159–71. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-70169-1_12.

Catarci, Tiziana, Donatella Firmani, Francesco Leotta, Federica Mandreoli, Massimo Mecella, and Francesco Sapio. 2019. ‘A Conceptual Architecture and Model for Smart Manufacturing Relying on Service-Based Digital Twins’. In *2019 IEEE International Conference on Web Services (ICWS)*, 229–36. https://doi.org/10.1109/ICWS.2019.00047.

Chen, Lianping. 2015. ‘Continuous Delivery: Huge Benefits, but Challenges Too’. *IEEE Software* 32 (2): 50–54. https://doi.org/10.1109/MS.2015.27.

De Giacomo, Giuseppe, Massimo Mecella, and Fabio Patrizi. 2014. ‘Automated Service Composition Based on Behaviors: The Roman Model’. In *Web Services Foundations*, edited by Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, 189–214. New York, NY: Springer. https://doi.org/10.1007/978-1-4614-7518-7_8.

Dzamashvili Fogelström, Nina, Tony Gorschek, Mikael Svahnberg, and Peo Olsson. 2010. ‘The Impact of Agile Principles on Market-Driven Software Product Development’. *Journal of Software Maintenance and Evolution: Research and Practice* 22 (1): 53–80. https://doi.org/10.1002/spip.420.

Fowler, Martin. 2006. ‘Continuous Integration. Martinfowler. Com’. May.

Gao, Feng, Muhammad Intizar Ali, Edward Curry, and Alessandra Mileo. 2016. ‘QoS-Aware Adaptation for Complex Event Service’. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 1597–1604. SAC ’16. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2851613.2851806.

Gao, Feng, Edward Curry, Muhammad Intizar Ali, Sami Bhiri, and Alessandra Mileo. 2014. ‘QoS-Aware Complex Event Service Composition and Optimization Using Genetic Algorithms’. In *Service-Oriented Computing*, edited by Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, 386–

93. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-662-45391-9_28.

Gomes, Cláudio, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. 'Co-Simulation: A Survey'. *ACM Computing Surveys* 51 (3): 49:1-49:33. https://doi.org/10.1145/3179993.

Groefsema, Heerko, Nick R T P van Beest, and Marco Aiello. 2018. 'A Formal Model for Compliance Verification of Service Compositions'. *IEEE Transactions on Services Computing* 11 (3): 466–79.

Groefsema, Heerko, Nick R T P van Beest, and Abel Armas-Cervantes. 2017. 'Automated Compliance Verification of Business Processes in Apromore'. In *BPM Demo Track (CEUR), Volume 1920)*, 1–5.

Groefsema, Heerko, Nick R. T. P. van Beest, and Guido Governatori. 2022. 'On the Use of the Conformance and Compliance Keywords During Verification of Business Processes'. In *Business Process Management Forum*, edited by Claudio Di Ciccio, Remco Dijkman, Adela del Río Ortega, and Stefanie Rinderle-Ma, 21–37. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-16171-1_2.

Hull, Richard. 2008. 'Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges'. In *On the Move to Meaningful Internet Systems: OTM 2008*, edited by Robert Meersman and Zahir Tari, 1152–63. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-88873-4_17.

Humble, Jez, and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ: Addison-Wesley.

'IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules'. 2010. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, August, 1–38. https://doi.org/10.1109/IEEESTD.2010.5553440.

International Electrotechnical Commission. 2019. 'ISO/IEC 21823-1:2019 Internet of Things (IoT) - Interoperability for IoT Systems - Part 1: Framework'. International Standards Organization. https://www.iso.org/standard/71885.html.

International Organization for Standardization. 1998. 'Information Technology — Open Distributed Processing, Reference Model: Overview Part 1'. Standard ISO/IEC 10746-1:1998. Geneva, CH: International Organization for Standardization. https://www.iso.org/standard/20696.html.

———. 2017. 'Systems and Software Engineering — Vocabulary'. Standard ISO/IEC/IEEE 24765:2017(E). Geneva, CH: International Organization for Standardization. https://www.iso.org/standard/71952.html.

Jones, David, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. 2020. 'Characterising the Digital Twin: A Systematic Literature Review'. *CIRP Journal of Manufacturing Science and Technology* 29 (May): 36–52. https://doi.org/10.1016/j.cirpj.2020.02.002.

Kitain, Lior. 2018. 'Digital Twin — The New Age of Manufacturing'. Medium. 5 November 2018. https://medium.datadriveninvestor.com/digital-twin-the-new-age-of-manufacturing-d964eeba3313.

Kolozali, Şefki, Maria Bermudez-Edo, Nazli Farajidavar, Payam Barnaghi, Feng Gao, Muhammad Intizar Ali, Alessandra Mileo, et al. 2019. 'Observing the Pulse of a City: A Smart City Framework for Real-Time Discovery, Federation, and Aggregation of Data Streams'. *IEEE Internet of Things Journal* 6 (2): 2651–68. https://doi.org/10.1109/JIOT.2018.2872606.

Krusche, Stephan, and Lukas Alperowitz. 2014. 'Introduction of Continuous Delivery in Multi-Customer Project Courses'. In . https://doi.org/10.1145/2591062.2591163.

Kumbhar, Amruta, Madhavi Shailaja, and Shankar Anupindi. 2018. 'Getting Started with Continuous Integration in Software Development'.

Machado, Michael, João Silva, João Sousa, and André Filipe Pinto Vale. 2020. 'The Evolution of Tridimensional Metrology: The Era of Computer Aided Metrology'. In . American Society of Mechanical Engineers Digital Collection. https://doi.org/10.1115/IMECE2019-11600.

Martin, James. 1983. *Managing the Data-Base Environment*. Englewood Cliffs, N.J: Prentice-Hall.

Mears, Laine, John T. Roth, Dragan Djurdjanovic, Xiaoping Yang, and Thomas Kurfess. 2009. 'Quality and Inspection of Machining Operations: CMM Integration to the Machine Tool'. *Journal of Manufacturing Science and Engineering* 131 (5). https://doi.org/10.1115/1.3184085.

Medjahed, Brahim, and Athman Bouguettaya. 2011. *Service Composition for the Semantic Web*. New York, NY: Springer. https://doi.org/10.1007/978-1-4419-8465-4.

Melesse, Tsega Y., Valentina Di Pasquale, and Stefano Riemma. 2020. ‘Digital Twin Models in Industrial Operations: A Systematic Literature Review’. *Procedia Manufacturing*, International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019), 42 (January): 267–72. https://doi.org/10.1016/j.promfg.2020.02.084.

Milosevic, Zoran, and Andy Bond. 2016. ‘Digital Health Interoperability Frameworks: Use of RM-ODP Standards’. In *EDOC Workshop 2016*, 1–10.

MTConnect Institute. 2018. ‘MTConnect Standard, MTConnect Part 5.0 Interfaces Version 1.4.0’.

Olsson, Helena Holmström, Hiva Alahyari, and Jan Bosch. 2012. ‘Climbing the" Stairway to Heaven"–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software’. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, 392–99. IEEE.

Pires, Flávia, Ana Cachada, José Barbosa, António Paulo Moreira, and Paulo Leitão. 2019. ‘Digital Twin in Industry 4.0: Technologies, Applications and Challenges’. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, 1:721–26. https://doi.org/10.1109/INDIN41052.2019.8972134.

Platenius-Mohr, Marie, Somayeh Malakuti, Sten Grüner, Johannes Schmitt, and Thomas Goldschmidt. 2020. ‘File- and API-Based Interoperability of Digital Twins by Model Transformation: An IIoT Case Study Using Asset Administration Shell’. *Future Generation Computer Systems* 113 (December): 94–105. https://doi.org/10.1016/j.future.2020.07.004.

Rahman, Akond Ashfaque Ur, Eric Helms, Laurie Williams, and Chris Parnin. 2015. ‘Synthesizing Continuous Deployment Practices Used in Software Development’. In *2015 Agile Conference*, 1–10. https://doi.org/10.1109/Agile.2015.12.

Ram, Karthik. 2013. ‘Git Can Facilitate Greater Reproducibility and Increased Transparency in Science’. *Source Code for Biology and Medicine* 8 (1): 7. https://doi.org/10.1186/1751-0473-8-7.

Savor, Tony, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. 2016. ‘Continuous Deployment at Facebook and OANDA’. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 21–30.

Scherfke, Stefan. 2013. ‘Machine Shop — SimPy 4.0.1 Documentation’. 2013. https://simpy.readthedocs.io/en/4.0.1/examples/machine_shop.html.

Schluse, Michael, Marc Priggemeyer, Linus Atorf, and Juergen Rossmann. 2018. ‘Experimentable Digital Twins—Streamlining Simulation-Based Systems Engineering for Industry 4.0’. *IEEE Transactions on Industrial Informatics* 14 (4): 1722–31. https://doi.org/10.1109/TII.2018.2804917.

Sodhro, Ali Hassan, Abdul Sattar Malokani, Gul Hassan Sodhro, Muhammad Muzammal, and Luo Zongwei. 2020. ‘An Adaptive QoS Computation for Medical Data Processing in Intelligent Healthcare Applications’. *Neural Computing and Applications* 32 (3): 723–34. https://doi.org/10.1007/s00521-018-3931-1.

Spinellis, Diomidis. 2012. ‘Git’. *IEEE Software* 29 (3): 100–101. https://doi.org/10.1109/MS.2012.61.

Stach, Christoph, and Frank Steimle. 2019. ‘Recommender-Based Privacy Requirements Elicitation - EPICUREAN: An Approach to Simplify Privacy Settings in IoT Applications with Respect to the GDPR’. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1500–1507. SAC ’19. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3297280.3297432.

Stacy, D, M Prudnikov, A Khan, and X Shen. 2017. ‘Practicing Continuous Integration and Continuous Delivery on Aws: Accelerating Software Delivery with Devops’. *Amazon Web Services, Tech. Rep.*

Van Der Aalst, Wil. 2012. ‘Process Mining’. *Communications of the ACM* 55 (8): 76–83. https://doi.org/10.1145/2240236.2240257.

Vrieze, Paul de, R. Arshad, and Lai Xu. 2023. ‘Interoperable Collaborative Manufacturing Process Simulation for Digital Twins (Submitted)’. *Computers in Industry*.

Wei, Shuangyu, Yuewei Bai, and Lai Xu. 2020a. ‘Study on Interoperation and Its’ Implementation of MES to Support Virtual Factory’. *Journal of Physics: Conference Series* 1633 (1): 012152. https://doi.org/10.1088/1742-6596/1633/1/012152.

———. 2020b. ‘Towards Quality Analysis of MES through CMM Data Interoperation’. *Journal of Physics: Conference Series* 1693 (1): 012049. https://doi.org/10.1088/1742-6596/1693/1/012049.

Xu, Lai, Paul De Vrieze, Hongnian Yu, Keith Phalp, and Yuewei Bai. 2020. ‘Interoperability of the Future Factory: An Overview of Concepts and Research Challenges’. *International Journal of Mechatronics and Manufacturing Systems* 13 (1): 3. https://doi.org/10.1504/IJMMS.2020.108333.

Yadav, Nitin, and Sebastian Sardina. 2011. ‘Decision Theoretic Behavior Composition’. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, 575–82. AAMAS ’11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Yemenicioğlu, Ender, and Arndt Lüder. 2014. ‘Implementation of an AutomationML-Interface in the Digital Factory Simulation’. In . https://doi.org/10.13140/2.1.3149.5681.

Zhou, Liang, Dan Wu, Xin Wei, and Zhenjiang Dong. 2019. ‘Seeing Isn’t Believing: QoE Evaluation for Privacy-Aware Users’. *IEEE Journal on Selected Areas in Communications* 37 (7): 1656–65. https://doi.org/10.1109/JSAC.2019.2916452.