

RWTH Aachen University

Fraunhofer FIT

Master's Thesis in Media Informatics

A Blockchain-Based Concept and Implementation for Machine Identity and Machine-to-Machine Communication

Author:	Mohammad Ghanem
Supervisor and Examiner:	Prof. Dr. Wolfgang Prinz
Co-Examiner:	Prof. Dr. Thomas Rose

January 2021

Eidesstattliche Versicherung

Statutory Declaration in Lieu of an Oath

Ghanem, Mohammad

391371

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)

Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

A Blockchain-Based Concept and Implementation for Machine Identity
and Machine-to-Machine Communication

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Bonn, 19/01/2021

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:

Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations

Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence

(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.

(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

I have read and understood the above official notification:

Bonn, 19/01/2021

Ort, Datum/City, Date

Unterschrift/Signature

Acknowledgments

Foremost I would like to thank my supervisor, Prof. Wolfgang Prinz, for giving me this opportunity to work on such an exciting topic. I am thankful for his continued support and guidance throughout the entire thesis.

I would also like to thank my brother Khaldoon for his valuable feedback on this work. I am deeply grateful to him for being my anchor in Germany and supporting me in all aspects of my life. Without him, I would not be able to pursue my studies.

Finally, I want to thank my family for their encouragement throughout my studies. Kind acknowledgments to all my friends and everyone who assisted me during the difficult times of 2020. I would like to especially thank Bushra, who is always lending an ear and standing beside me despite the distance between us.

Abstract

Over the last two decades, the rapid advances in digitalization methods put us on the fourth industrial era's cusp. It is an era of connectivity and interactivity between various industrial processes that involve various humans, artificial agents, systems, machines, and services. These parties need a new trusted environment to exchange and share information and data without relying on third parties. Blockchain technologies with features like security, immutability, and decentralization can provide such a trusted environment.

This thesis investigates the applicability of blockchain in the manufacturing industry. The focus is to utilize the blockchain with its characteristics to build a machine to machine (M2M) communication and digital twin solutions. The thesis proposes a generic conceptual design for a distributed system that uses smart contracts to construct digital twins for machines and products, give a self-sovereign digital identity for twins, and execute manufacturing processes over the blockchain. Moreover, a functional prototype of the proposed solution is implemented for a factory simulation case study consisting of four machines and two manufacturing processes. The prototype includes a distributed web application (DApp) that allows different actors to interact with the machines, products, and processes through the blockchain. It provides dashboards to monitor and control manufacturing processes that run autonomously by the smart contracts. Developing a prototype allows evaluating the design and the modeling to find loopholes and limitations. This work demonstrates the value of executing the industrial processes as a smart contract code residing on a decentralized network.

Keywords: Industry 4.0, Digital Twin, Blockchain, Smart Contracts, M2M, DID.

Contents

Acknowledgments	iii
Abstract	v
1. Introduction	1
1.1. Motivation	1
1.2. Research Questions	2
1.3. Outline	3
2. Related Work	5
2.1. Background	5
2.1.1. Industry 4.0	5
2.1.2. Digital Twin	5
2.1.3. Blockchain	6
2.1.4. Self-Sovereign Identity (SSI)	7
2.2. Literature Review	9
2.3. Summary	12
3. Conceptual Design	15
3.1. Application Scenario	15
3.2. General Overview	16
3.2.1. System Objectives	16
3.2.2. System Components	16
3.2.3. System Actors	17
3.3. Machine Modeling	18
3.3.1. Machine Digital Twin	18
3.3.2. Machine Identity	19
3.3.3. Twins Interaction	20
3.4. Product Modeling	22
3.4.1. Product Digital Twin	22
3.4.2. Product Identity	22
3.4.3. Product Credentials	22
3.5. Manufacturing Process Modeling	23
3.5.1. Process Structure	24
3.5.2. Process Execution	24
3.6. Registry Modeling	25
3.7. Use Cases	27

3.8. Summary	28
4. System Implementation	29
4.1. Factory Infrastructure Implementation	29
4.1.1. Fischertechnik Factory Model	29
4.1.2. Fischertechnik Factory Model Customization	30
4.2. Architecture	33
4.3. Development Tools and Frameworks	34
4.3.1. Blockchain and Smart Contracts	34
4.3.2. Gateway and Web Application	34
4.4. Smart Contracts Implementation	35
4.4.1. Machines	35
4.4.2. Products	37
4.4.3. Processes	37
4.5. Identity Implementation	38
4.5.1. Distributed Identifiers	39
4.5.2. Verifiable Credentials	39
4.5.3. Registry	40
4.6. Results	40
4.6.1. User Interfaces	40
4.6.2. Source Code	50
4.7. Summary	50
5. Evaluation	51
5.1. Solution Evaluation	51
5.1.1. Digital Twin Limitations	52
5.1.2. M2M Communication Limitations	52
5.1.3. Technical Limitations	52
5.2. Prototype Evaluation	53
5.2.1. Software Testing	53
5.2.2. Quantitative Evaluation	54
5.3. Summary	56
6. Conclusion and Future Work	57
6.1. Conclusion	57
6.2. Answer to Research Questions	58
6.3. Future Work	59
A. Use Cases Flows	61
A.1. Start Task	61
A.2. Finish Task	62
A.3. Save Product Operation	63
A.4. Save Reading	64

A.5. Save Alert	65
A.6. Kill Task	66
A.7. Get New Reading	67
A.8. Authorize/Unauthorize Process	68
A.9. Execute Process Step	69
A.10. Create Product	70
A.11. Start Process	71
B. Fischertechnik Factory Model Machines	73
C. Smart Contracts Class Diagrams	75
C.1. Machine	75
C.2. Product	77
C.3. Process	78
C.4. Ethereum DID Registry	79
C.5. Registry	80
List of Figures	81
List of Tables	83
Bibliography	85

1. Introduction

1.1. Motivation

Until today, the industry has seen three major revolutions, and the fourth is on its way [Roj17]. The fourth industrial revolution (Industry 4.0) represents the next step in the evolution of traditional factories towards smart, automated factories. These factories are designed in such a way to reduce production costs, increase productivity, improve quality, and achieve efficient use of resources. To achieve the Industry 4.0 goals, a wide range of technologies have to be used in manufacturing like Robotics, Autonomous Systems, Internet of Things, Cloud Computing, Intelligent Data Analytics, Artificial Intelligence, and many more [FM18].

However, all these technologies rely on centralized networks and need to trust intermediaries or third-party operators [Ang+18; SHK17; LBH18; Afa+18; Gar+19]. As a result, the industry faces many challenges related to the data like transparency, security, privacy, and trustworthiness. These challenges prevent Industry 4.0 to reach its full potential. A decentralized and trusted platform is needed to facilitate the relationships among parties. Such a platform can be built with the help of blockchain technologies.

The main advantage of the blockchain is decentralization. It is a distributed shared ledger instead of the traditional centralized systems [Zhe+17]. It enables a group of entities to reach an agreement on a particular activity and register it without the need for a middleman or third party. The agreed-upon activity can be anything like payment transactions, purchase activity, or any non-financial transaction [MA]. The blockchain revolution created an immutable, secure, and transparent system to log and execute any transaction type. Blockchain can also preserve the order of events and ensure the correctness of logged transactions over time. No one can individually delete or modify any of the recorded transactions. The invention of smart contracts made the blockchain a promising platform for developing distributed and trustworthy applications [MA]. As a result, many industries and businesses started to consider blockchain technologies as part of their infrastructure. Given its key features such as immutability, traceability, and reliability, it represents a perfect candidate to be integrated into Industry 4.0 factories. It will increase its efficiency, security, and provenance concerning the related data of goods, assets, and operations [Int19]. Besides, these features enable creating new business models that were impossible to form a few years ago. With the introduction of blockchain, businesses and individuals can record and secure their conducted agreements among themselves without middle parties.

This thesis aims to shed light on the blockchain's capability to improve the manufacturing industry and understand how blockchain can work with other technologies to overcome the challenges mentioned earlier. It will investigate the feasibility of implementing blockchain solutions within the boundaries of smart factories. In particular, we will focus on two aspects of manufacturing. The first one is the communication between machines to enable the concept of machine-to-machine (M2M) communication over the blockchain, where one machine can ask another machine to perform a particular task without human involvement. The second one is tracking and tracing the machines and products while executing the manufacturing processes and building a digital representation with the blockchain's help. The thesis will also spout how this technology can be used to provide a blockchain-based digital identity for both the machines and the products. However, the identity aspect is not the main focus of this thesis. Therefore, our work in this direction was limited to using two of the emerging standards in this area; namely, the Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) [DIDa; VC-].

1.2. Research Questions

The work done in this will answer the following research questions.

RQ1: How can blockchain technologies support M2M communication and digital-twins solution in the manufacturing industry?

To answer this question, a literature review is needed to find the possibilities of using blockchain technologies to support manufacturing, particularly M2M communication and digital twin solutions.

RQ2: What are approaches for the design and development of blockchain-based solutions for M2M communication and digital identity?

In blockchain-based systems, the blockchain platform is only one component. To build the whole system, it has to be integrated and fit with other components. The innovative nature of the blockchain requires different design and development approaches than conventional centralized systems. For blockchain-based systems, there are many architectural decisions have to be made. For example, decisions about which parts of the data should be stored on-chain or kept outside the off-chain.

RQ3: How can machines, products, and manufacturing processes be modeled on the blockchain?

This question is similar to the previous question but more specific to the discussed topic. It involves deciding what kind of information is required to model the machines,

products, and processes to enable the M2M communication over the blockchain. Also, how to use blockchain tools like smart contracts to model these entities.

RQ4: How feasible is it to utilize the blockchain for M2M communication and digital twin solutions?

An evaluation is needed to validate the blockchain-based solution's feasibility and applicability for the given problem to answer this question. The evaluation should also discuss the weaknesses and the limitation of such a solution.

1.3. Outline

In this section, we present an outline of the thesis. Following the introduction chapter, we present the related work in chapter 2. The first part of chapter 2 covers the necessary concepts of this thesis, such as Industry 4.0, and blockchain, and its related technologies. The second part is the academic literature review. Chapter 3 gives a conceptual design and modeling for the solution. Then chapter 4 proceeds by explaining the conceptual design's implementation details. Besides, it provides information about the case study used in this thesis and its prototype. Chapter 5 covers the evaluation for both the conceptual design and the prototype's implementation. The last chapter 6 summarizes the finding and discusses the possible future work.

2. Related Work

This chapter presents a brief overview of all the concepts used in this thesis, followed by a section summarizing the academic literature related to blockchain applications in the manufacturing industry. With this review, the chapter contributes answers to the first research question RQ1 on how blockchain technologies can benefit M2M communication and digital twin solutions.

2.1. Background

This section will explain the background information about the terms and concepts used during the course of the thesis. We will start with the none technical ones like Industry 4.0 and digital twin, then move into the technical concepts like the blockchain, smart contracts, and self-sovereign identity.

2.1.1. Industry 4.0

Industry 4.0 as a term referred to the fourth industrial revolution and appeared for the first time during the Hannover Fair in 2011 in Germany [Roj17]. The fourth revolution can be summarized as the next step in transforming traditional factories towards smart digitalized and automated ones. In such factories, each component can compute information and communicate with other components in a peer-to-peer fashion regardless of its size. New information technologies must be used and embedded in all aspects of the industrial systems to achieve such evolution. Technologies such as Robotics, autonomous systems, Internet of Things, Cloud Computing, Intelligent Data Analytics, and Artificial Intelligence [Roj17; XXL18]. The ultimate goals for industrial revolutions are to reduce production costs, increase productivity, improve the quality, and achieve efficient use of resources [TS17]. Industry 4.0 can achieve these goals in many different ways using various technologies. Such goals can be achieved in the manufacturing industry by enabling autonomous decision-making, self-healing, self-configuration, and real-time monitoring.

2.1.2. Digital Twin

The digital twin is considered one of the critical technologies for Industry 4.0. However, the term itself goes back in time before Industry 4.0. It was introduced in 2003 by Michael Grieves as part of his product lifecycle management (PLM) research at the University of Michigan [Gri14]. Since that time, the term changed repeatedly, and many

definitions appeared in both academia and industry [Ful+12]. All definitions agreed that the digital twin is the virtual or digital representation of a real physical environment. However, they are different in some technical aspects. The [RSK19] authors presents the following definition based on an extensive literature review: "A digital twin is defined as a virtual representation of a physical asset enabled through data and simulators for real-time prediction, monitoring, control and optimization of the asset for improved decision making throughout the life cycle of the asset and beyond". The digital twin concept is being used in many fields and has many applications in health care, smart cities, and manufacturing [Ful+12; RSK19].

2.1.3. Blockchain

Blockchain technology has been evolving rapidly since 2008, when it was introduced to be the underlying technology behind cryptocurrencies. Nowadays, blockchain is suitable and useful for many use cases and applications, including the industrial ones[Mus19; Al-19]. A simple definition of the blockchain is a decentralized, distributed, and immutable ledger controlled by a consensus algorithm. No one has full control over the blockchain's data because it is stored across a distributed peer-to-peer network. The blockchain is structured internally into a linked list of blocks. Each block stores information about the ordered sets of transactions. The consensus algorithm links the blocks using cryptographic hashes to secure the data from tampering and revision. There are many approaches to implementing and building the blockchain, involving many technical and technological details. Regardless of this, blockchain, in general, has the following characteristics:

- **Immutability:** Once the data is written to the blockchain, it cannot be altered or changes. This also guaranteed non-repudiation of the data, which means the data's sender cannot deny their signature's authenticity.
- **Transparency:** Anyone who has access to the blockchain network can access the data written in the blockchain and see all transactions' history.
- **Decentralization:** The blockchain runs on a peer-to-peer network. This brings the decentralized nature to data storage. Decentralization is achieved by using consensus algorithms across a network. Which also makes the decision making and governance decentralized.
- **Distributed Trust:** The participants of a network rely on the blockchain network itself rather than relying on trusted third-parties.

Blockchain can also be seen as general computational platforms because it can store and execute computer programs called smart contracts. They are programs deployed to the blockchain as data and executed by sending transactions. The execution of the smart contract, once deployed, is deterministic and immutable. The smart contract is used to

program any business logic. For example, it can be used to administer the ownership of assets, build a voting system, or automate or monitor legal contracts' execution. Smart contracts increase the range of applications and services that can be build using the blockchain. The systems that have their logic build using smart contracts are called decentralized applications or Dapps.

There are many platforms and implementations for the blockchain. Each one of them has different features and serves different purposes. One of the most famous general-purpose blockchain platforms is Ethereum. It is a global, open-source platform for decentralized applications [Eth]. Ethereum has a decentralized replicated virtual machine, called the Ethereum Virtual Machine (EVM), which can execute Turing-complete scripts and run decentralized applications. EVM code can be written and compiled in higher-level languages like Solidity [Sol].

2.1.4. Self-Sovereign Identity (SSI)

Self-Sovereign Identity is an emerging approach to build a digital identity management system where the subjects (individuals and entities) have full control over their identity-related information. Such information includes personal information, identifiers, and credentials. The subjects have their information stored locally or remotely on a distributed network. It allows the subjects to selectively grant access to their information without the need for validation from a trusted authority or intermediary operator. The goal of this approach is to improve privacy and overcome the issues of the centralized identify approaches. SSI is also referred to as a blockchain-based identity because the blockchain, with its features like decentralization and immutability, provides the necessary infrastructure to build the SSI systems [Les+19].

Interoperability is an essential requirement for digital identity, and because digital identifiers can be build using different formats and representations, standards and specifications must be used. Two of the emerging standards to provide a specification for the identify-related information are Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) [DIDa; VC-]. Both of them are technical standards build by the World Wide Web Consortium (W3C) to specify the architecture, data model, and representations of the SSI. Each one of these specifications serves different purposes, but both must be used together to build the identity management system.

The DIDs specification defines a new type of self-sovereign globally unique identifier called DID. It is a simple text string and has the following syntax:

`"did:" + <did-method> + ":" + <method-specific-identifier>`

It consists of three parts: URI scheme identifier (did), the DID method's identifier, and method-specific identifier. For example a DID for the "test DID method" will look like "did:test:123456789abcdefghi". As the DIDs is just a specification, the actual

implementation is provided by the DID method. It defines a set of rules for creating, resolving, updating, and deleting a DID. Also, the DID method allows the DID to be resolvable. The method takes the DID as input and returns a DID document contains all the metadata about the identity. The DID document might have one or more representation, but the most common representation is the JavaScript Object Notation for Linked Data (JSON-LD) [JSO]. It includes information about the public keys used for authentication, authorization, and communication mechanisms. Currently, there are around 40 implementations of the DID specification [DIDb]. Each one of these implementations is considered a different DID method.

The DID specification only provides the data model for the identity itself, which can be used for authentication and authorization. However, it does not specify the credentials or the claims about the subject of identity. The VCs specification provides a mechanism to express digital credentials called verifiable credentials in a cryptographically secure, privacy-respecting, and machine-verifiable way. It also allows the holder of verifiable credentials to generate another information called verifiable presentations. These presentations can be shared with verifiers to prove the possession of specific credentials without revealing the actual credentials. An overview of the VCs specification components and their roles is in figure 2.1 alongside the information flows between them.

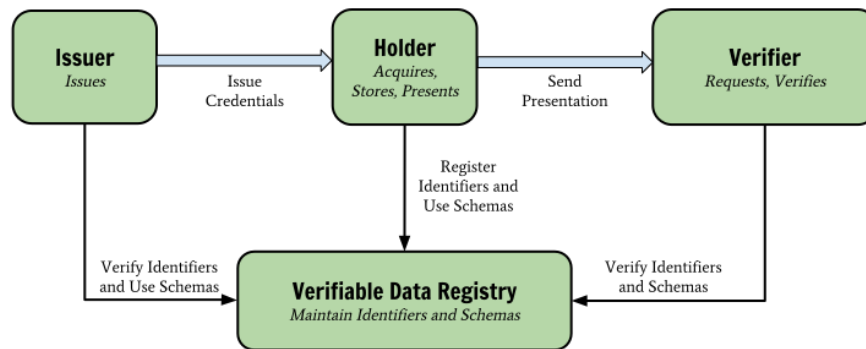


Figure 2.1.: VCs Components and Information Flows [VC-]

The holder is a role for an entity to perform by having one or more VC and generating verifiable presentations. The issuer is another role that can be performed by asserting claims about one or more subjects, creating a verifiable credential from these claims, and transmitting the holder's verifiable credential. The subject is an entity about which claims are made. In most cases, the holder and the subjects are the same entity, but it is not always the case. For example, a parent (the holder) might hold the verifiable credentials of his child (the subject). The verifier is a role performed when receiving verifiable credentials for processing and verification. The verifiable data registry is a role for a system by mediating the creation and verification of all the related information of verifiable credentials.

2.2. Literature Review

Our search was on popular academic search engines such as Google Scholar, arXiv, and Research Gate. We searched for the following terms: ("Blockchain" OR "Distributed Ledger" OR "Smart Contract") AND ("Manufacturing" OR "Production" OR "M2M" OR "Factories" OR "Plants" OR "Digital Twin"). The result of our search included research papers published in journals and conferences. In the following subsections, we will describe what we found the most relevant to the discussed topic. We describe the goal, implementation approach, and the prototype for each of these studies. At the end of the section, we will provide a comparison between the discussed works.

FabRec: A Prototype for Peer-to-Peer Network of Manufacturing Nodes [Ang+18]

In this work, the goal is to build a trustless distributed network. Where different industrial organizations can collaborate and share information about manufacturing processes. The network was built using blockchain and smart contracts technologies. They stored information about machines, manufacturers, and their capabilities. Through this network, all the information is shared transparently among participants. A participant of this network can be a human, manufacturing machine, a computing node, or an agent representing any organization. The prototype of this work was a small network built using Ethereum. It showed how the machines on a decentralized network could autonomously interact without human involvement. The network consists of 4 separate computers representing various fabrication service providers and two mining nodes. They used Arduino, and a CNC machine controlled through MachineKit [Mac] interface to simulate the machines. The setup demonstrated how a smart contract could check for specific events received by the CNC machine. Then the contract triggers a command sent to the Arduino board. The work also evaluates the prototype's performance for two types of consensus algorithms: proof of work (POW) and proof of authority (POA).

We think this work focused more on the horizontal integration of technology among many manufacturers. The network does not contain data about the manufacturing processes. Such data is accessed through oracles from other existing Enterprise Resource Planning (ERP) systems and Manufacturing Execution Systems (MES). The work assumed that these existing IT systems are trustworthy. Regarding the prototype, it did not take into consideration a full automated manufacturing scenario. However, they showed the autonomous decision-making and interaction of two physical devices on the blockchain network through the prototype.

Toward a blockchain cloud manufacturing system as a peer to peer distributed network platform [LBH18]

The authors of this work integrated cloud manufacturing technology with blockchain. The work proposes a distributed peer-to-peer network architecture to improve manufacturing cloud platforms' security and scalability. The architecture provides self-trust,

data integrity audit, data resilience, and secure data sharing. They used smart contracts to write the rules of the agreement between the end-users and the service providers. These rules contain the due date, quality measurement, and payment information. The authors showed a case study that includes five manufacturing service providers and 15 end users. Each service provider delivers three types of services. The end users can send one request to the system at a time. The scenario begins when a customer wants to produce an industrial piece manufactured with a CNC milling machine. He publishes information about the piece using the blockchain smart contract. Then, service providers send their offers to the customer (product cost and due date). When the customer accepts an offer from the service provider, it is executed as an atomic exchange. To simulate the physical machines, they used CNC simulator [CNC]. Moreover, security and performance evaluation was carried to verify the proposed architecture.

We think this work focused on building a trusted peer to peer network to enable manufacturing as a service. The aim was to enhance communication between manufacturing parties and overcome the issues that resulted from centralization. The main benefit of using the blockchain in this work is to enable secure data exchange. However, this work did not consider how architecture can be used to automate the manufacturing processes.

Blockchain-based ubiquitous manufacturing : a secure and reliable cyber-physical system [Bar+19a]

A similar approach to the previous work. The goal of this work is to improve communication between manufacturing service users and manufacturing services providers. The proposed platform is explained in detail based on its architecture, consensus mechanism, and communication protocol. The use case of this work was in the 3D printing manufacturing industry. They tested based on network performance and three practical scenarios. The results showed that blockchain technologies could help solve some existing problems found in cloud manufacturing literature.

BPIIoT: A Light-Weighted Blockchain-Based Platform for Industrial IoT [Bai+19]

This work addresses the security and trust problems in the Industrial IoT (IIoT) ecosystem. The proposed architecture consists of two parts on-chain network and an off-chain network to reduce network load and latency. The on-chain part provides data communication and transmission services. Simultaneously, the off-chain provides data storage and complex processing that cannot be done in the on-chain part. Any machine/equipment can add operation data to the blockchain in this platform, send transactions to the related smart contracts, and receive transactions from other nodes in the blockchain. Two applications are discussed in detail to demonstrate the proposed architecture.

One of the applications is smart predictive maintenance. In this application, the platform is used to manage the equipment information and the maintenance processes. The use case is a packaging platform based on the proposed BPIIoT architecture. It has

two sensors connected with two robots to collect state data, such as temperature and vibration. This state information is being recorded in the on-chain network. The smart contract that serves as an agreement between the equipment owner and the maintenance service provider monitors the state information. When the temperature or vibration frequency exceeds a given threshold for a certain number of times, a command is triggered by the smart contract. In this way, the equipment can request maintenance services, replenishment, or replacement of spare parts issued by the service provider/supplier. They did not present the technical details about the implementation of this application. Overall this work did not focus on the manufacturing processes.

Blockchain technology in the chemical industry: Machine-to-machine electricity market [SHK17]

This work is one of the most highly cited papers with more than 250 citations. It is one of the first research works which used the M2M concept with blockchain. In this paper, the authors explored the applications of blockchain with Industry 4.0. They built a proof of concept where a blockchain is used to facilitate the interaction between machines. The goal was to enable the M2M electricity market, where industrial plants are autonomously trading electricity over a blockchain. The agreement between the producer and the consumer is built using smart contracts. The information about the energy consumption (in kWh) published by the machines is stored as transactions in the blockchain. Each transaction has a fee (in USD) according to the agreement specified by the smart contract.

The scenario presented in the prototype includes two electricity producers and one electricity consumer trading over a blockchain. The physical machines were replaced with industrial simulations in Aspen Plus (AP) [Asp]. They used MultiChain [Mul] to build the private blockchain. The authors did not evaluate their prototype. The focus of this work was on energy trading using M2M. The work does not provide any framework or architecture that can be used on a large scale in manufacturing. This work was only to demonstrate that the blockchain is capable of facilitating the interaction between machines.

Industry 4.0: Smart Contract-based Industrial Internet of Things Process Management [Gar+19]

In this study, the authors focused on industrial machine to machine communication and how blockchain technologies can improve it. They introduced smart contract-based middleware for M2M communications to make it secure and decentralized. Through this middleware, IoT devices can communicate without the need of a trusted intermediary. The middleware controls and executes contracts to order tasks from field devices. Also, it monitors the field device states and executes actions based on a change in their state. It may also request a field device to perform service under a smart contract. All

information about the actions and processes are recorded in the blockchain through smart contracts. The authors present a proof of concept to simulate smart contracts in the industrial context. They used two Raspberry Pi 3 boards representing field devices. MQTT server for the communication between devices. The proof of concept works so that after the device changes its state, it publishes this change to the MQTT server. The middleware subscribes to the device publications and reviews the state according to the deployed smart contracts. Once a smart contract is satisfied, The middleware publishes messages to request a service from another field device. The other device picks this message published by the middleware and executes the operation accordingly. Two experiments were performed to evaluate the performance of the prototype.

Unlike the other works, the focus here was on vertical communication within one factory. This work emphasizes the real-time requirement for M2M communications in industrial processes. The result showed that smart contracts technology is still not mature enough to provide such an essential requirement. Similar to the other works, the proof of concept was small and limited to only two devices.

2.3. Summary

It is evident from the works we considered in this section that researchers have already discussed blockchain applications in the manufacturing industry. They all addressed the same problem with different perspectives. On the one hand, some works focused on the horizontal integration between manufacturing parties to enable manufacturing as a service either between manufacturers themselves or between manufacturers and customers. These works neglected the actual manufacturing processes and focused more on the concept of trading.

On the other hand, some works focused more on the manufacturing processes by enabling machine-to-machine communication over the blockchain. However, the use cases of these works were oversimplified and, in most cases, were limited in size (only two machines). This does not reflect the actual communications between machines on the production line. The majority of the discussed works did not mention how the machines are being modeled in their systems. This is an essential aspect of M2M-based systems because it will help build a fully autonomous manufacturing process. None of the works discussed how the products are being modeled within the blockchain.

Another essential aspect is the identity management of the machines and the products. Each machine has to know and identify other machines before establishing communication. All the works we discussed used only the public/private key pairs as identities. This approach has many limitations and problems [ZB18]. It makes the identity tightly coupled with the algorithm used to generate the key pair. None of the work spouts the issue of managing the digital identities of the machines or the products. However, this issue is well addressed in different research areas. We found some recent works which

Ref	On-Chain M2M	Machine Modeling	Product Modeling	Process Modeling	Identity Management	Prototype	Platform
[Ang+18]	✗	✓	✗	✗	✗	✓	Ethereum
[LBH18]	✗	✓	✗	✗	✗	✓	MultiChain
[Bar+19a]	✓	✗	✗	✗	✗	✓	Hyperledger
[Bai+19]	✓	✓	✗	✗	✗	✗	✗
[SHK17]	✓	✗	✗	✗	✗	✓	MultiChain
[Gar+19]	✓	✓	✗	✗	✗	✓	Ethereum

Table 2.1.: Related Work Comparison

focused entirely on identity management in the context of the Internet of things (IoT) and the industry [Bar+19b; Fed+20; ZB18]. They present various approaches to build a self-sovereign identity management system using blockchain technologies. As our focus in this thesis is not the identity problem, we did not discuss these works in the same way we discussed other works. Table 2.1 is a comparison table that summarizes the discussion of this section. This chapter showed how blockchain technologies could benefit the M2M communications in the manufacturing industry, answering the first research question RQ1.

3. Conceptual Design

In this chapter, we will present a conceptual design for an envisioned system that utilizes the blockchain to build M2M communication and digital twins solutions. It will describe and discuss the requirements, the compounds, the modeling, and design decisions. The main contribution of this chapter is answering the second and third research questions RQ2 and RQ3.

3.1. Application Scenario

Before presenting the solution, we have to describe a scenario where the system might be used. The scenario will give a better understanding of the context and help justify the design decisions made during the solution's development. The system is supposed to be used in an Industry 4.0 factory where the manufacturing processes are automated and self-organized. Imagine a factory consisting of a warehouse, the main processing station, a sorting line, and a robot arm. These components are equipped with environmental sensors that report temperature, humidity, air pressure, and air quality. The components collaborate to execute two manufacturing processes, the supplying process and the production process.

The supplying process aims to supply the unprocessed products into the factory and store them in the warehouse. The process starts when the robot arm grabs the product from the input area and moves it to the identification station. Once the product information is available, the warehouse fetches an empty container. The robot arm moves the product from the identification station and places it into the empty container. Another small device within the warehouse moves the container and store in the warehouse's right place.

The production process's goal is to perform the main processing on the unprocessed products and deliver them through the delivery station. It starts by moving one unprocessed product from the storage place into the pickup area of the warehouse. Then the robot arm picks the product and moves it into the input area of the main processing. Afterward, the product will go through a series of processing operations like milling and melting. Once the main processing unit finishes processing the product, a small robot arm will put it on the sorting line's belt. A sensor detects the product color, and according to the color of the product, it will place it in the corresponding collection area. Lastly, the main robot arm picks the product from the collection area and moves it into the delivery station. While these two processes are executing, the factory compounds

exchange and communicate information and data about the tasks and products. They also perform quality checks on the products to ensure they meet a certain standard. Additionally, the compounds monitor the sensors' readings and can raise an alert if something went wrong.

3.2. General Overview

3.2.1. System Objectives

The system's primary goal is to utilize the blockchain as infrastructure to support the manufacturing industry in M2M communication, digital twins, and digital identity. Taking into consideration the scenario from the last section, the fine-grained objectives of the system is the following:

- Build a digital twin of the machine using the blockchain. The digital twin should include information about the machine's functions and operations like the machine's tasks, sensor readings, alerts. It also should reflect the current status of the machine.
- Build a digital twin of the product using the blockchain. The product's digital twin should include information about the operations performed on the product and related information.
- Provide a blockchain-based digital identity for the digital twins.
- Model the manufacturing process and its business logic using the blockchain. In other words, make all the communication between the machines go through the blockchain.
- Build a user interface to let the stockholders interact with the system and access the blockchain's information.
- The system design should be generic and replicable to different use cases.

3.2.2. System Components

Our system is functioning alongside the existing infrastructure of the factory. It uses the blockchain to store and manage the data generated by the factory infrastructure. The data stored in the blockchain is used to build a digital representation of the machines and the products. Furthermore, all the communication between the machines goes through the blockchain. Therefore, the system consists of the following three components:

- **Factory Infrastructure:** It represents all the existing hardware and software of the factory. It includes machines, sensors, and other devices. It also includes a client application that connects the factory infrastructure with the blockchain.

- **Blockchain:** It is the data storage and computational component of the system. It is the network of all the nodes processing the transactions and running the system's smart contracts.
- **Web Application:** It is the application used by different actors to access the information stored in the blockchain. It consists of a front-end application and a blockchain client application.

The figure 3.1 shows high-level diagram of the system components and the data flows between them.

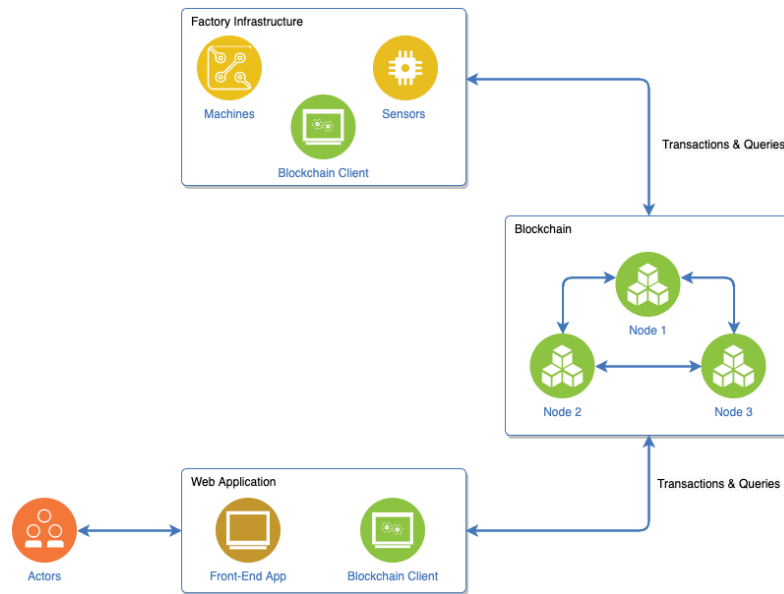


Figure 3.1.: High-Level System Overview

3.2.3. System Actors

The following is a list of all the active actors of the system.

- **Machine:** The physical machine inside the factory.
- **Process:** The manufacturing process involves two or more machines.
- **Machine Owner:** The owner of the machine.
- **Process Owner:** Any person or business who uses the machines inside a manufacturing process to produce a product.
- **Product Owner:** Any person or business who is interested in the final product.

3.3. Machine Modeling

This section presents how machines are modeled. The machine is the main and the most crucial entity in the system. Our modeling is generic and can be applied to any machine. We use the term machine to refer to any factory component, including machines and robots of all sizes. We assume that the machine already has its digital representation provided by the machine's manufacturer or third-party software. The machine has an owner, but the machine itself is an active entity. Each machine is capable of performing several industrial tasks. Several processes can use the machine. There are no restrictions on the size and the complexity of the machine or its tasks. For example, the warehouse in the application scenario can be modeled as one machine, even if it consists of several small sub-machines. In this case, we will ignore the internal interactions between the internal parts and focus on the external interactions with other entities. Another way of modeling the warehouse is to consider it a manufacturing process and model its internal parts as machines. In the implementation chapter, we will provide concrete examples of how the application scenario components are modeled as machines.

3.3.1. Machine Digital Twin

One of the system objectives is to build a digital representation of the machine using the blockchain. To build the machine's digital twin in our system, we decided to model the machine as a smart contract. Each machine will have a corresponding smart contract deployed into the blockchain. The smart contract with all the information stored within it represents the machine's digital twin created by the blockchain. Once the smart contract updates itself to include new information about the machine, it will be part of the machine's digital twin, and it can not be altered or changed. The ultimate goal when building a digital twin is to make a replica of the physical entity. However, we decided to limit the information included in the machine's digital twin. The table 3.1 shows an overview of the information being stored in the twin. All this information is managed and stored by the smart contract of the machine. Therefore, the machine's digital twin protects the information from being altered by unauthorized users or parties. However, all the information stored in the machine's smart contract will be publicly readable.

So far, we described the machine's digital twin as a registry of the information described. To go beyond this and make the digital twin of the machine an active component, we used the programmability feature of the blockchain. The smart contract of the machine can do complex programmable behaviors on the data stored within it. It could be programmed to perform conditions checks on the data stored and then do some actions once these conditions are verified. Of course, the physical machine can perform these checks by itself. However, having the digital twin to perform them will bring trust as the blockchain runs it. For example, it could be programmed to perform the following checks:

- **Product Quality Check:** This is a check performed to ensure that the product has

Information	Description
Identity	The identity of the machine. More about the identity in the following section.
Basic Information	Static information about the machine like the serial number, the model, manufacturing year, and similar info. Only the machine owner can provide such information, and once it is added to the digital twin, it can not be changed.
Processes	Information about the processes which use the machine. The authorized processes are allowed to assign tasks to the digital twin of the machine. The machine owner provides this information.
Tasks	Information about the machine's tasks. It involves information about the starting time, finishing time, the parameters, the process, and the product.
Readings	Any numeric information coming from the machine sensors like temperature or humidity. This information is sent by the physical machine and stored in the digital twin alongside the reading's timestamp.
Alerts	Information about unexpected scenarios or failures. This information could be provided by the physical machine or by the digital twin itself. The digital twin can perform some logical checks as described later in this section and create alerts based on the check result.

Table 3.1.: Machine Digital Twin Information

some properties or meets a certain quality standard. The check might involve accessing the digital twin of the product.

- **Reading Values Check:** This is a check on the numerical values for the machine sensors. If a reading value, for example, the temperature exceeded a certain throughout, the digital twin can do some actions like creating an alert.

3.3.2. Machine Identity

As we illustrated in the previous section, the machine's digital twin is an active actor. It needs a digital identity to facilitate communication and interaction with other entities of the system. The industrial and manufacturing systems have very long life-cycles, and therefore the identity of the machine should be the same during its lifetime. As

our system is blockchain-based, identity management can not be based on traditional approaches. Otherwise, the objectives of the system can not be achieved. The machine's identity must be owned and controlled by the machine, rather than stored or managed by a third party. Therefore, we decided to use a blockchain-based identity management approach. One of the emerging standards that use the blockchain features is Decentralized Identifiers (DIDs). Our system is using DID as a standard to manage the identities of the machines. Each machine has a permanent identifier called DID. A simple text string e.g. `did:example:123456789abcdefghi`. Each DID is resolvable to a DID document that contains information associated with the identity of the machine. The DID document is stored within the blockchain, making the DID and its document persistent and immutable.

The DID itself is just a specification that includes the core architecture, the data model, and representations. The implementation details are specified by the DID method. Different methods use different ways to build the identity management system based on the DID standard. Our DID method is based on ETHR DID method [Metb] developed by uPort [uPoa]. This method considers any blockchain address to be a valid identifier without being registered on the blockchain. Therefore, originating a new identity for the machine is as simple as generating a public-private key pair and deriving the blockchain address. In our system, it is done by the owner of the machine. The logic of managing the identity is done through a smart contract deployed on the blockchain. Therefore, all the subjects' operations to manage the identity requires interaction with the smart contract by sending transactions. This involves operations like changing the ownership, making delegations, and authentication.

3.3.3. Twins Interaction

This section explains how the physical machine and its digital twin interact and communicate with each other. According to our modeling, the information between the physical and digital twins is being exchanged in both directions. The machine sends information to be stored in the digital twin, and vice versa; the digital twin sends information to control and change the physical machine's behavior.

Interacting with blockchain is done by participating in the network and running client software of the blockchain platform. The details of this software may be different depending on the implementation of the platform. Regardless of this, every network node needs this client to process the transactions and validate/create blocks in the chain. Such functionality requires a large amount of storage and processing power as the node needs to have a full copy of the whole chain. The manufacturing machines could not have such capabilities to be a node and run the blockchain client software. Therefore, we decided that the physical machine will not run the blockchain client by itself. Instead, the physical machine will communicate via some protocol with a gateway running the blockchain client and acting like one of the blockchain nodes. The gateway will use the

machine private and public key pair to interact with the blockchain on behalf of the machine. An assumption has to be made regarding the communication channel between the machine and the gateway. We assume the channel is secured, and no attacker can alter or modify the messages exchanged between the machine and the gateway. Through this gateway, the machine will send data to its digital twin inside the blockchain.

So far, we have explained how the physical machine can send data to its digital twin. The next type of interaction is when the digital twin wants to notify or send data to the physical machine. As the machine's digital twin is a smart contract deployed on the blockchain, it runs in a closed execution environment and can not directly interact with external systems. The only way the digital twin of the machine can communicate with the outside world is through events. The smart contract of the machine emits certain types of events to interact with the physical machine. Anyone can watch these events, including the gateway. Once the gateway receives new events, it forwards them to the machine. Table 3.2 shows the events emitted by the machine smart contract.

Event Name	Event Description
Task Assigned	An event emits when a process assigns a task for the machine. The emitted event has all the information about the assigned task.
Task Started	An event emits when the machine starts executing a task.
Task Finished	An event emits when the machine finishes executing a task.

Table 3.2.: Machine Smart Contract Events

Figure 3.2 illustrates the interaction between the machine and its twin while executing a task. The sequence starts when an authorized process assigns a task for the machine by calling a function on its smart contract. The contract emits a task assigned event. The gateway is listening to the events generated by the smart contract of the machine. Once the gateway receives the task assigned event, it will first get the task's parameters then send the task to the physical machine. At the same time, the gateway will call the start function on the machine's smart contract. Calling this function will store the starting time of the task and emit the Task Started event. The physical machine is now performing the task. After it finishes the task, it will inform the gateway about the finished task. Then, the gateway calls the finish task function of the smart contract of the machine. This function call will store the finishing time of the task and emit the Task Finished event. The machine might send information about the product operations being performed to the gateway during the task execution. The gateway takes this information and passes it to the machine's digital twin, which stores it in the product's digital twin.

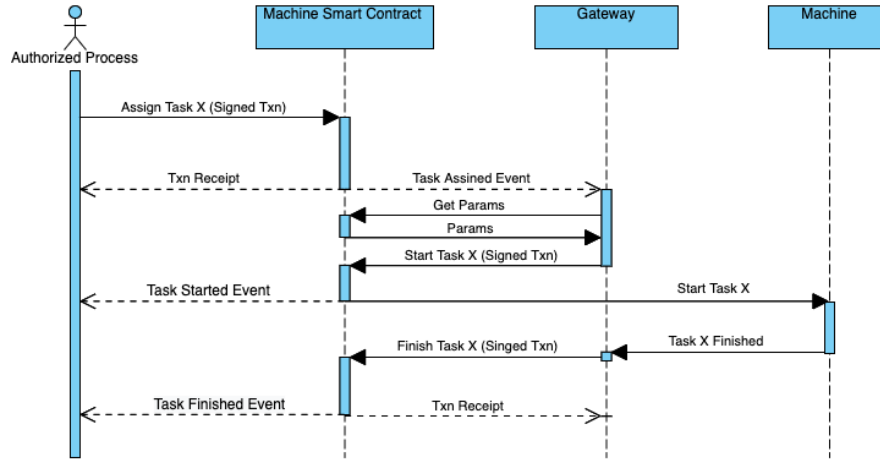


Figure 3.2.: Twins Interaction

3.4. Product Modeling

This section presents how products are modeled in our system.

3.4.1. Product Digital Twin

The product is the second primary entity in our system. The product as an entity can be modeled in many different ways. Many information can be stored throughout the product's life cycle. In our work, we focus only on what is happening during the manufacturing process inside the factory. Our modeling only takes into consideration simple non-compounded products, and the granularity is a single product. All the product information is stored in one smart contract called "Product". This information is provided autonomously by the machine's digital twins. While the machine executes one of its tasks, the machine's digital twin will add this operation to the product's digital twin if the machine operates on the product. In this way, all the operations performed on the products by different machine is stored in one place, and they form the digital representation of the manufactured product. The table 3.3 provides the information stored in the product's digital twin.

3.4.2. Product Identity

For the product identity, we are using the same identity management of the machine. Each product has a DID associated with it.

3.4.3. Product Credentials

Besides having the product information stored on-chain in the digital twin, the products have off-chain credentials. For every operation performed on the product, it receives

Information	Description
Identity	Information about the digital and physical identity of the product.
Operations	This is general information about the operations performed on the product. Each operation is stored with a name, result, timestamp, and information about the machine that did this operation.
Processes	Information about the authorized processes allowed to modify the digital twin of the product.

Table 3.3.: Product Digital Twin Information

an offline credential/claim from the machine. The credential can be verified by a third party to ensure its authenticity. To implement this, we used the emerging standard from the W3C, which is called Verifiable Credentials (VCs). This standard fits well with the DID standard we used to build the identity management. The machine is the issuer of the credential, and the product is the subject. It creates the credential that contains some information about the product, the operation, the machine's DID, and cryptographic proof. The product can claim that it underwent a particular operation or satisfied a certain standard by presenting the corresponding credential to a verifier. The verifier can check the machine's DID document (the issuer) and verify the claim's authenticity cryptographically. Having each operation as a separate credential allows the product to present the needed information to the verifier without revealing other information that might be sensitive.

3.5. Manufacturing Process Modeling

This section explains how the manufacturing processes are modeled to enable machine-to-machine communication over the blockchain. Real manufacturing processes tend to be complicated and consist of several stages or even sub-processes. Each one of them involves a lot of machines and devices. We are considering a simple manufacturing process that consists of several steps and no sub-processes. Each step is a high-level task performed by a specific machine, for example, fetch an empty container task. We also assume that the process is fixed. In other words, the steps, their order, and the corresponding task types are known in advance. However, the machine allocation is dynamic. So the machine executing a specific type of task can be replaced by any other machine which can do the same type of tasks. Executing the process requires communication and interaction between the digital twins of the machines involved in the process.

3.5.1. Process Structure

We modeled the manufacturing processes as smart contracts. Each process is a smart contract written by a manufacturer and running on his behalf on the blockchain. The smart contract is programmed to assign tasks to the digital twins of the machines. The smart contract consists of several functions. Each function represents a step in the process. The function body assigns the task to the machine and executes other business logic if necessary. The contract is responsible for starting/finishing the execution of the process instances. For each execution, the contract creates a process instance and store information about it like the starting time, the finishing time, and the execution status.

Event Name	Event Description
Process Started	An event emits when a process instance is started. The emitted event has all the information about the started instance.
Process Step Started	An event emits when a process step is started.
Process Finished	An event emits when a process instance is finished.

Table 3.4.: Process Smart Contract Events

3.5.2. Process Execution

The process execution is done with the help of a client, which also runs in the gateway. The communication between the client and the smart contract of the process is done the same way explained in the twin interaction section. The client calls functions in the smart contract by signing transactions, and the smart contract emits events to communicate with the outside world. Table 3.4 shows the events emitted by the process's smart contract. The client will be listening to these events and other events from machines' smart contracts.

The execution is carried on by the client calling the process functions. Each function represents a step in the process, and the function call assigns a task to the corresponding machine. Before executing the process, the addresses of the digital twin of the involved machines must be supplied into the smart contract. It is started when the owner of the process or an authorized actor triggers the process by calling the start function, which emits the process started event. The client then calls the first step function, which assigns the first task of the process to the corresponding machine. Now the machine will work on the task as we explained in a previous section. Once the machine finishes its task, and the task finished event is emitted from its digital twin, the client can resume the execution by calling the second step function, assigning the second task to the responsible machine.

The execution continues in the same way until the process finishes all its steps, and then the process finished event is emitted. Before assigning each task, the process's smart contract needs to authorize the machine if the task involves performing product operations. During the execution of the process, the smart contract of the process can access the digital twin of the product or the machines' digital twins and get data from them. In such a way, the process's smart contract will be enforcing the rules and requirements of the manufacturing process. With this modeling, the machines act as separate entities and can be used by several processes.

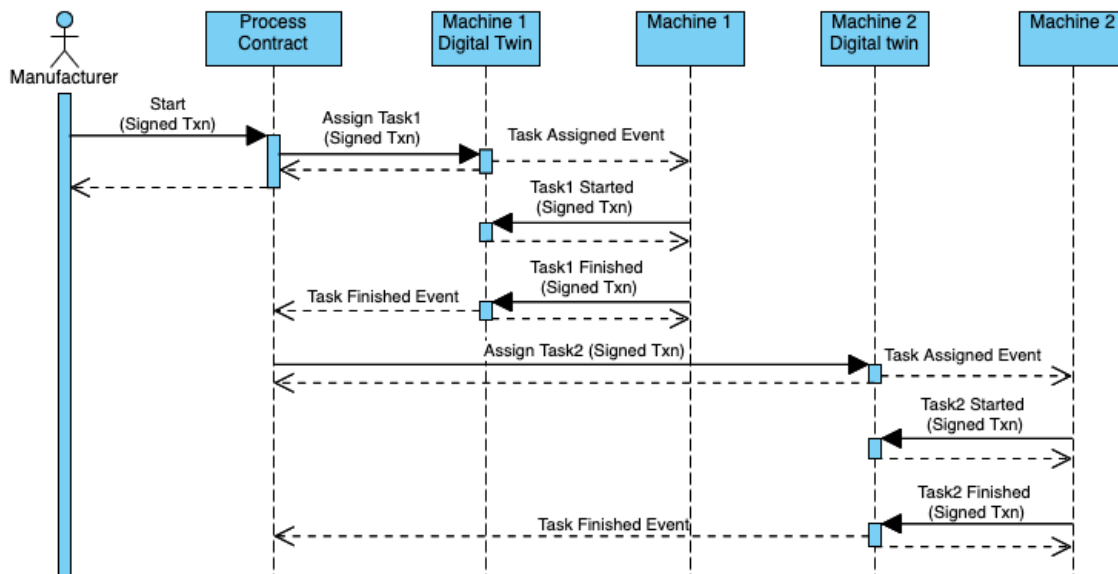


Figure 3.3.: Process Example

To illustrate the execution, let us assume we have a process that involves two machines. Each machine has its digital twin as a smart contract deployed into the blockchain. These two machines belong to different owners and might be located in different locations. The owner of the process (the manufacturer) decided that his process needs two machines, a machine capable of doing task 1 and another capable of doing task 2. The manufacturer writes and deploys the smart contract of the process, including all the business logic that implements the argument between him and the machines' owners. Figure 3.3 shows the sequence diagram for executing this example process. All the interaction between the machine and its digital twin is gone through the gateway, as explained previously. In this example, the machines and the manufacturer might be using different gateways to access the blockchain, and they are omitted from the diagram for simplicity.

3.6. Registry Modeling

The need for this compound was found at a late stage while developing the solution. This section explains why such compound is needed and how it is being modeled.

By the registry, we mean a software compound that holds information about a class of entities. A well-known example of a registry is the Domain Name Service (DNS).

Our system's registry component holds information about all the entities, including the users, machines, and processes. It maintains a mapping between custom-defined symbolic names and the blockchain addresses of these entities. For example, each machine has two addresses, the DID and the address of its smart contract. The registry will map these non-human readable addresses into human-readable names or identifiers. Another reason for using the registry is that the system needed a place where all the machines and processes smart contracts are listed. The mapping between a human-readable identifier and a blockchain address is common in blockchain-based applications. The mapping allows upgrading the smart contract by replacing the blockchain address of the old version of the contract with the address of the new version. The registry can be implemented in many different ways, with or without using blockchain technologies. However, our registry is blockchain-based, and it is modeled as smart contract. It is often called an on-chain registry contract [XWS19]. The contract stores information about the addresses and provides functions to manage this information.

3.7. Use Cases

Figure 3.4 present all the use cases of the system sorted according to the actor. More details for each use case can be found in the appendix A.

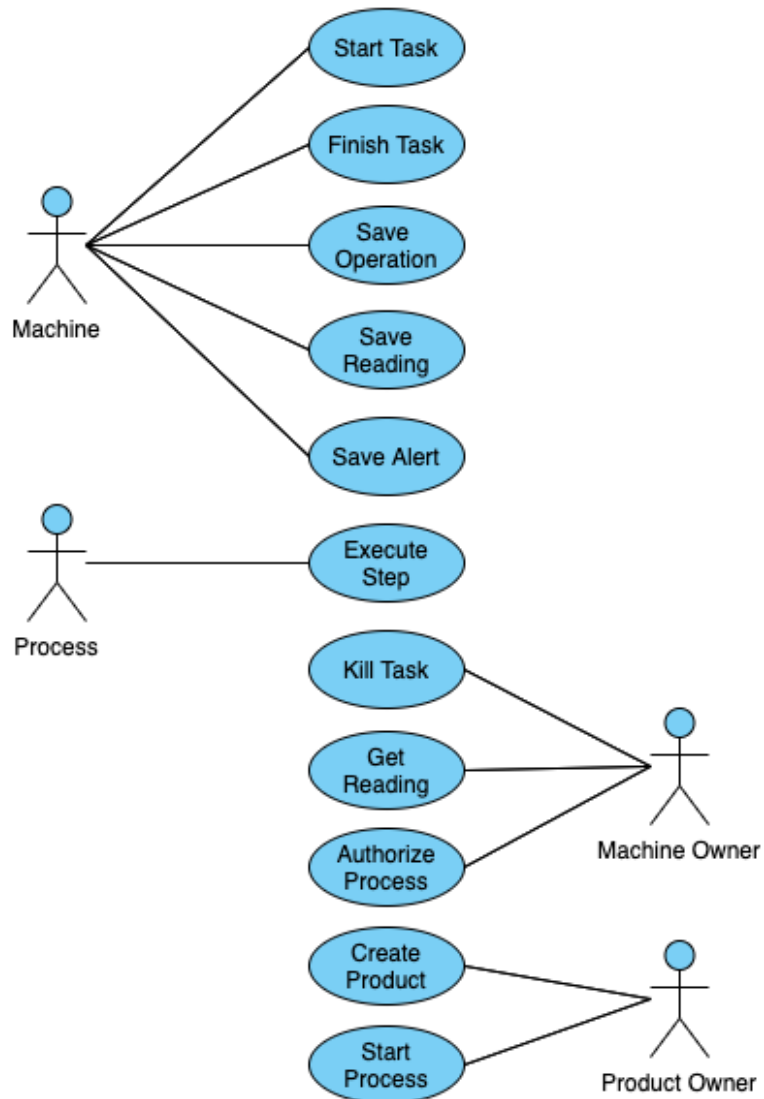


Figure 3.4.: Use Cases Diagram

3.8. Summary

This chapter provided a conceptual design for an envisioned blockchain-based system for M2M communication, digital twin, and digital identity. It started with a description of a scenario where the system might be used. Then it showed the requirements of the system and listed the actors of the system. The main contribution of this chapter was to provide abstracted modeling of the system compounds. It explained how the machines' digital twins and the products are modeled and listed in the digital twins' information. Also, it illustrated how the manufacturing processes are modeled to enable the machine to machine communication over the blockchain. This chapter's conceptual design answered the second and third research questions RQ2 and RQ3 on an abstract level without going into the technical details. The next chapter will provide technical and implementation details.

4. System Implementation

In this chapter, we provide the technical details about the solution described in the last chapter. Even though the research questions RQ2 and RQ3 have been answered in the last chapter on an abstraction level, this chapter answers them from the technological perspective by showing the actual software frameworks and tools which can be used to build the envisioned solution.

4.1. Factory Infrastructure Implementation

In chapter 3 we presented an application scenario in which the system is used. This section presents an actual implementation for a simulation of this scenario to put our case study's foundation.

4.1.1. Fischertechnik Factory Model

We are using the Fischertechnik Learning Factory 4.0 [Fisa]. Figure 4.1 shown a picture of the model. It is a simulation of a factory used for learning and understanding industry 4.0 applications. The factory has a built-in program that depicts the ordering process, the production process, and the delivery process in digitized and networked steps. The factory model comes pre-configured and programmed to perform a set of built-in demo scenarios controlled and monitored through an online dashboard. After the order has been placed using the dashboard, the product passes through the respective factory modules, and the current status is immediately visible on the dashboard. The products are tracked using NFC (Near Field Communication): A unique identification number (ID) is assigned to each product, enabling traceability and visibility of the products' current status in the machining process. Figure 4.1 shows a picture of the factory model. Six Fischertechnik TXT controllers control it, and it is divided into the following stations:

- Vacuum Gripper Robot (VGR): A three-axis robot with a vacuum suction gripper that can position products quickly and precisely in three-dimensional space. The vacuum gripper picks up products and moves them within the working space. Figure B.1 in the appendix shows a picture of this machine.
- High-Bay Warehouse (HBW): is a storage area that allows computer-aided storage and retrieval of products. Figure B.2 in the appendix shows a picture of this machine.

- Multi-Processing Station with Oven (MPO): is a processing station that simulates different processes like melting and milling. Figure B.3 in the appendix shows a picture of this machine.
- Sorting Line with Color Detection (SLD): A sorting path with color recognition automatically separates differently colored products. Figure B.4 in the appendix shows a picture of this machine.
- Sensor Station with Camera (SSC): A environmental station with a surveillance camera is used to record measured values within the factory. It has an environmental sensor and a photoresistor that allows the air temperature, humidity, air pressure, air quality, and brightness to be measured.
- Delivery and Pickup Station (DPS): is an input and output station with a color detection and NFC reader.

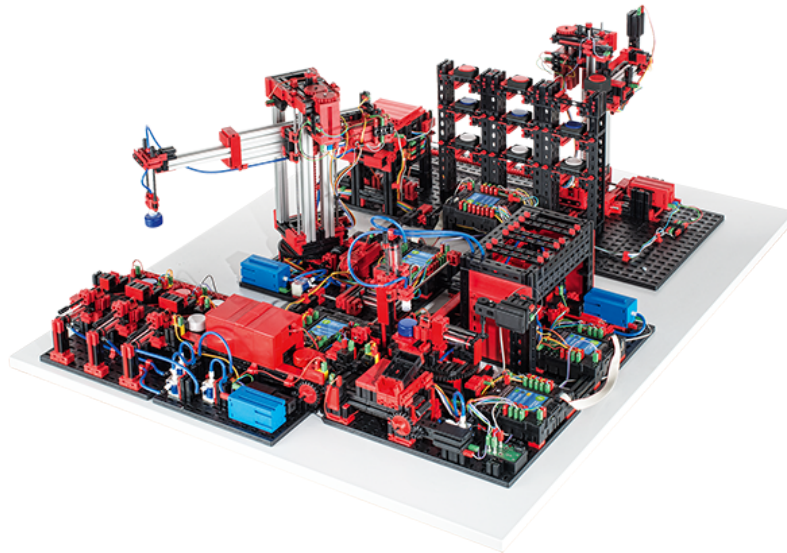


Figure 4.1.: Fischertechnik Factory Model

The controllers exchange and communicate messages to coordinate the execution of the program. All the communication is done via MQTT (Message Queuing Telemetry Transport) protocol. One of them acts as the central flow controller, and all the operations and messages are going through it. The source code of programs running by the controllers can be found on the GitHub repository of the factory model [Fisb].

4.1.2. Fischertechnik Factory Model Customization

Even though the Fischertechnik factory model is a fully functional simulation, we had to customize it to fit our case study. At first, the factory has to be split into components

and assign a specific role for each one. The factory used to have six stations mentioned in the last section. We decided to model only 4 of them as machines (VGR, HBW, MPO, and SLD). The other two stations are the SSC and DPS. The SSC station is responsible for the sensor readings. As we want the machines to act as separate entities, we considered the SSC station to be part of every machine. Therefore, every machine has its sensor readings. The other station, DPS, is considered part of the VGR machine as it is the only machine that interacts with the DPS station. Then, the built-in demo scenario had to be modified and split into different processes and tasks performed by the four machines. The following subsections will describe the tasks performed by each machine.

VGR Tasks:

- **Get info:** This task is responsible for picking up the product from the input station and detect the color and the NFC ID of the product.
- **Drop to the container:** This task places the product into the empty container.
- **Move from HBW to MPO:** This task moves the product from the HBW machine to the input station of the MPO machine.
- **Pick sorted:** This task picks the sorted product from the SLD machine.

HBW Tasks:

- **Fetch container:** This task fetches an empty container if exists.
- **Store product:** This task stores a product in the warehouse if there is an empty place.
- **Fetch product:** This task fetches a product from the warehouse if exists.
- **Store container:** This task stores the empty container.

MPO Tasks:

- **Process:** This task applies operations like melting and milling on the product.

SLD Tasks:

- **Sort:** This task sorts the product and places it in the storage location according to its color.

The machines with these tasks types are able to collaborate to perform two processes namely the supplying process and the production process. They are the same processes mentioned in the application scenario section in the last chapter. The following is a description of the two processes using the name of the machine and the task types.

Supplying Process:

This process involves only 2 machines and consists of 4 steps:

1. Get info step performed by VGR.
2. Fetch empty container performed by HBW.
3. Drop to the empty container performed by VGR.
4. Stop the product performed by the HBW.

Production Process:

This process involves 4 machines and consists of 5 steps:

1. Fetch product performed by the HBW.
2. Move the product from the HBW to MPO performed by the VGR. At the same time, the HBW will be storing back the empty container.
3. Processing the product performed by MPO.
4. Sorting the product according to the color performed by the SLD.
5. Picking the sorted product and deliver it performed by the VGR.

4.2. Architecture

Before explaining the implementation's technical details, we present the overall architecture and show the system components' information flow. Figure 4.2 shows the architecture of the system.

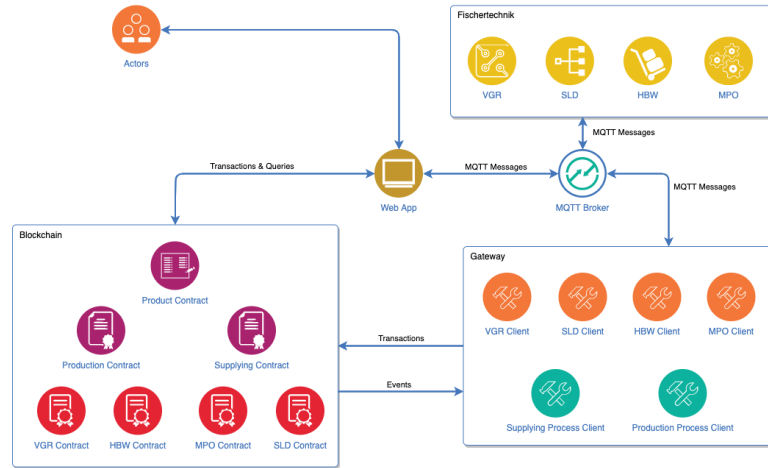


Figure 4.2.: System Architecture

The Fischertechnik component includes the four machines mentioned in the last section. Each machine runs on a different controller, and all connected to the MQTT broker. In the gateway, there are four machine client programs; each one corresponds to one physical machine. The machine client program is responsible for synchronizing the physical machine with its digital twin. Also, the gateway includes two processes client one for each process. The process client is responsible for executing the process steps one by one by monitoring the machine's status. In the blockchain, all the systems contracts are deployed and running. For each machine, there is a machine contract that represents the digital twin of the machine. For each process, there is a process contract that manages the process execution inside the blockchain. In addition to the machines and process contracts, The product's smart contract includes information about product digital twins.

The Fischertechnik and the gateway communicate with each other by sending MQTT messages. The gateway clients send transactions to smart contracts and listen to events. The Web application also sends transactions to the smart contract and uses read-only functions to display the smart contracts' information. There are contract-to-contract functions calls between the blockchain contracts, but for simplicity, they have been omitted.

4.3. Development Tools and Frameworks

4.3.1. Blockchain and Smart Contracts

In this section, we present the tools used in the development and implementation of smart contracts.

- **Truffle:** A smart contract development framework [Tru]. It is part of the truffle suit, a set of tools built to make the development of DApps easier. The framework allows compiling, debugging, deployment, and testing Solidity smart contracts in Ethereum networks. The framework uses its contract abstraction to facilitate interacting with the smart contract from Javascript.
- **Ganache:** Another tool from the truffle suit provides a lightweight blockchain for development, and test use [Qan]. Ganache has two versions, one with GUI and the other as command-line tool CLI. We used ganache CLI for testing as it provides a fast, clean instance of the Ethereum network.
- **Quorum Blockchain:** Quorum is a permissioned version of Ethereum, focused on enterprise use [Qoq]. It has several advantages over public Ethereum like privacy, high performance, and supporting multiple consensus algorithms.
- **OpenZeppelin:** Like the truffle suite, OpenZeppelin provides security products to build, automate, and operate decentralized applications [Opeb]. The main product of OpenZeppelin is OpenZeppelin Contracts, which provides standard smart contract implementation and Solidity components to be used as a library while building custom contracts. We used OpenZeppelin Contracts utility components in our contracts. We also used the OpenZeppelin Test Environment as the testing environment for all the contracts [Opea].
- **Web3.js:** The Ethereum JavaScript API [Web]. It implements the generic JSON-RPC protocol to connect and interact with any Ethereum network. We used this library in the gateway in all clients to sign transactions and interact with the deployed smart contracts.

4.3.2. Gateway and Web Application

This section presents the tools used in the development and implementation of the gateway and web application.

- **NodeJS:** An open-source, JavaScript runtime environment that executes JavaScript code outside a web browser [Nodb]. All gateway clients run inside one NodeJs server.
- **ReactJS:** A JavaScript library for building user interfaces [Rea].

- **Tabler-React:** An open source frontend template [Tab].
- **MetaMask:** A browser extension to interact with any Ethereum network [Meta]. It handles account management and connecting the user to the blockchain. With this extension, the web application users can manage their accounts and keys and send/sign transactions.

While working on the implementation, we used the following tools and framework, but eventually, we discarded them:

- **Node-RED:** It is a flow-based programming tool for wiring together hardware devices, APIs, and online services [Noda]. It provides a browser-based editor to build the application and its behavior as a network of black-boxes, or nodes called in Node-RED. Inside each node, a custom Javascript code is executed every time the node is triggered. It also offers built-in nodes to create a live data dashboard. In the early stages of the implementation, we used Node-Red to connect to the Ganache network and use the built-in user interfaces to display the blockchain's information. Then we noticed the limitations of the built-in user interfaces and the difficulty of making custom-styled interfaces. Therefore, we decided to use React instead of building the web application as it provides high-level reusability and customizability.
- **Drizzle:** Another tool provided by the truffle suite [Dri]. It is a collection of front-end libraries to keep the user interfaces synchronized with the blockchain. After we started to build the react application, we used the drizzle to keep the web application synchronized with the smart contract. However, we faced many problems due to unsolved bugs, and we found that it caused a performance issue because it causes unnecessary re-renders even if the observed smart contract did not change its state. Therefore, we decided to discard it and use the web3.js events listening functionality to synchronize the front-end with smart contracts.

4.4. Smart Contracts Implementation

This section is about implementing the blockchain compound of the system.

4.4.1. Machines

In the third chapter, we explained the abstract conceptual modeling of the machine. This section will present the technical implementation of the machines smart contracts. To make the implementation of the machine smart contract generic, we decided to use the template method pattern by making the machine contract an abstract contract. This abstract contract contains all the functionality mentioned in the machine modeling section which is the functionality shared between all machines. In this way, all digital twins of our system will have the same interface so other components can interact with

any machine as long as its smart contract extent the abstract base contract. Figures C.1 and C.2 in the appendix show a UML class diagrams for the abstract machine contract.

For a new digital twin of a machine to be created in the system, the machine's smart contract must extend the abstract machine contract. The constructor of the abstract contract takes four arguments of type address. The first one is the machine owner's address, and the second one is the DID of the machine. The third and fourth arguments are the addresses of the registry and the product smart contracts, respectively. The abstract machine's contract registers the machine in the registry with the name taken from the get name function. Once the contract is deployed, these values can not be changed. The new contract must implement the following abstract methods:

- **getNumberOfMachines:** The function takes no arguments and returns the number of tasks types.
- **getTasksTypesCount:** The function takes no arguments and returns the task name for each task type.
- **getSymbol:** The function returns the symbol of the machine, which is a string used as a human-readable short name.
- **getName:** The function returns the name of the machine.

The custom functionality can then be added to the child contract by using/overriding the parent contract's functions. The child contract can also implement and enforce custom rules or business logic by overriding the abstract contract's functions. For example, one machine can override the saveReading function and check the reading value. If the value under/above a certain threshold, an alert will be created by calling the saveAlert function.

Our case study involves the four machines mentioned in the previous section. We created a smart contract for each of them, extending the machine abstract smart contract and implementing all the abstract functions. Some custom functionality has been added to each smart contract. For example, the store product task function for the HBW machine needs to check if the product has an NFC ID and a color. Otherwise, the HBW will not accept the task, and the transaction will be reverted.

Regarding the implementation of the machine client, it is a JavaScript program run in the NodeJS server. Every machine has its client, but all of them are running on the same NodeJS server. According to the machine task types, the client might be different, but in general, all clients share the same functionality in synchronizing the machine and its digital twin. The client uses the web3.js library to interact with the machine's smart contract. It listens to the events emitted from the contract and sign transactions using the machine key pairs.

4.4.2. Products

The implementation of the product digital twin is a single smart contract called Product. The contract stores information about all products of the system. Figure C.3 in the appendix shows a UML class diagram of the product smart contract.

The product smart contract allows creating a product by calling the create product function. The function takes an Ethereum address as the DID of the product and creates a record for this product. The caller of this function will be the product owner, and it can not be changed. The product owner can add info to the product's digital twin using the web application by calling the corresponding functions and signing transactions with his keys. As we explained in the modeling section, the product smart contract authorizes processes that can authorize machines to modify the digital twin of a single product. The authorized machine can save the operations they performed on a particular product in its digital twin. Operations info and their results can be accessed later by machines' smart contracts to ensure that they meet specific requirements. Another essential info stored about the product is the physical identifier. The identifier could be an NFC UID or a barcode. It is used to access the digital twin of a product and get all the information about it. Another way of retrieving the info is by using the product's DID, which is an Ethereum address.

4.4.3. Processes

The implementation of processes uses the same approach as machines. Therefore, we created an abstract smart contract called Process to include all processes' standard functionality. Figure C.4 in the appendix shows a class diagram of the process smart contract.

For a new process to be created in the system, the smart contract process must extend the abstract process contract. The constructor takes three arguments. The first one is the owner of the process. The second and third arguments are the addresses of the registry and the product smart contracts, respectively. The new contract must implement the following abstract methods:

- **getNumberOfMachines:** The function takes no arguments and returns the number of machines participating in the process.
- **getNumberOfSteps:** The function takes no arguments and returns the number of steps of the process.
- **getMachineNumber:** The function takes the step number and returns the machine responsible for executing the step.
- **getStepTaskType:** The function takes the step number as an argument and returns the task type of this step. The task type depends on the machine assigned to execute this step.

- **getSymbol:** The function returns the symbol of the process, a string used as a human-readable short name.
- **getName:** The function returns the name of the process.

After deploying the process smart contract, the process owner must set the smart contract address for every machine involved in the process. It can be done using the `setMachineAddress`, which takes the machine's number and the address as arguments. The machine address can be changed at any time but only by the owner of the process. The process can then be started on a particular product by calling the `start process` function, which takes the Ethereum address (DID) of the product as an argument. The `start process` function calls the `authorize process` function in the product smart contract to authorize itself. Only the product owner can authorize a process; therefore, the `start process` function can only be called by the product owner.

Once the process owner calls the `start` function and the corresponding transaction is confirmed, the contract emits a `process started` event. The process client which runs in the gateway will be listening to this type of events. After the process contract emits the `starting` event, the client will begin executing the process by calling the `first step` function. All `steps` functions take the process instance ID as an argument, and inside each one of them, the corresponding task is assigned to the machine by accessing its digital twin and calling the `assign task` function. In addition to this, custom functionality can be part of the `step` function body. The process client is also listening to the `task finished` events emitted by the machine smart contracts. Every time a machine finishes a task assigned to it by the process, the process client will trigger the next step, which assigned the next task in the process. This execution continues until the process reaches its final step, and then the process client calls the `finish process` function to mark this instance of the process as finished.

Our case study involves two processes the supplying process and the production process. We created a smart contract for each one of them according to the last section's approach. For each process, a corresponding client is running in the gateway. It is also a Javascript program run inside the NodeJS server. The client's job is to execute the process by tracking the `tasks finished` events emitted by the machines involved in the process. The client uses the process owner key pairs to sign the transactions.

4.5. Identity Implementation

This section will explain the implementation of the identity management system, which involves the distributed identifiers and verifiable credentials.

4.5.1. Distributed Identifiers

As mentioned in the third chapter, we use the DID standard to assign digital identities for machines and products. The DID standard is just a specification, and the implementation details are left to the DID method. There are many DID methods available with a functional implementation. Each one of them has different functions, but all of them comply with DID specification. We used the *ethr* DID method developed by uPort. The *ethr* method uses the registry specified by the ERC 1056 [ERC]. The implementation of this ERC is available as an open-source project [uPob]. The registry itself is a smart contract called *EthereumDIDRegistry* developed by uPort to use Ethereum addresses as fully self-managed DIDs. The contract allows public key resolution for off/on-chain authentication. It also allows key rotation, delegate assignment, and revocation to allow third-party signers on the key's behalf, as well as setting and revoking off-chain attribute data. Figure C.5 in the appendix shows a UML class diagram of the *EthereumDIDRegistry* smart contract.

The *EthereumDIDRegistry* contract is already deployed to the Ethereum mainnet and other testnets like Ropsten and Rinkeby. Anyone can use these contracts to manage DIDs. However, we did not use any of these deployments and decided to deploy the contract to our custom network. The reason behind having our own DID registry is to have a close system where all compounds are under our control. Also, contracts can not be called from another contract unless they are in the same network. Regardless of the network, the registry can be used in the same way. Every Ethereum address is a valid identity, and the key pairs corresponding to the address maintains ultimate control over the identity represented by the DID. By default, each identity is controlled by itself. The registry allows changing the identity owner, but we did not use this functionality in our implementation.

To resolve the DID document of identity, we used a library developed by Decentralized Identity Foundation called *ethr DID Resolver* [Idea]. The library constructs the DID document using the read-only functions and contract events emitted from the registry contract. We used this library and built a DID resolver interface in our prototype to resolve any DID and construct the corresponding DID document. The result section shows an example of a DID of a machine and the corresponding DID document.

4.5.2. Verifiable Credentials

The implementation of the verifiable credentials in our system is based on the library called *did-jwt* developed by Decentralized Identity Foundation [Ideb]. It allows signing and verifying JSON Web Tokens (JWT), and all public keys are resolved using DIDs. The library support *ethr* DID method alongside many other methods.

In our case study, the signer is the machine, and the subject receiving the credential is the product. Each machine client uses the library to sign a credential using its DID.

The content of the credential could be anything as long as it is a valid JSON object. In our implementation, the credential content is information about a product operation. Anyone interested in verifying the credential can use the library or a similar library to check its validity. Under the hood, the library access the DID registry to check the credential signer's validity. We build a verifiable credential resolver to decode the credential and verify its validity. The result section shows an example of a credential created for a product by one of the machines.

4.5.3. Registry

As we described in the third chapter, the registry is an on-chain registry contract deployed into the network. The on-chain registry is a typical pattern in the blockchain-based applications, and there are already some well-established implementations for it. For example, ESN is a name service on Ethereum blockchain implemented as smart contracts [ESN]. ESN provides an extensible naming system based on the Ethereum blockchain. We had the idea of using the ESN contracts in our implementation instead of building a custom registry, but the idea was discarded. ESN would be sufficient if we only wanted the mapping between the blockchain addresses and the human-readable. We want the registry to have a list of all the machines and all processes, and therefore we decided to implement a custom registry.

Figure C.6 in the appendix shows a UML class diagram of the registry smart contract. The contract functionality is simple. It maps an Ethereum address into a string. For machines and processes, they register themselves by calling the corresponding register function inside the constructor of their smart contracts. The registry address is filled into the machines and processes smart contracts to call the registering functions. When the register machine and register process functions are called, the registry contract calls the get name method to get the name of the machine or the process to register it and link it to the contract's address.

4.6. Results

In this section, we present the results of the implementation. It is restricted to the web application as it is the only component that has user interfaces. However, the last sub-section provides details on the source code for all components' implementation.

4.6.1. User Interfaces

The following sub-sections list most of the web application user interfaces. As the web user interface is big and can not fit in one image, some images are not a full user interface but a snippet that shows a particular part of the interface.

Dashboard Interface

The dashboard interface is shown in figure 4.3 with some components hidden. This interface shows three kinds of information: machines' current status, processes' current status, and events log. The current machine's status is coming directly from the machine's digital twin for each machine. The web application is listening to the smart contract events, and based on the emitted events; it changes the status of the machine in the UI. The upper part of the interface shows the status of the Fischertechnik machines.

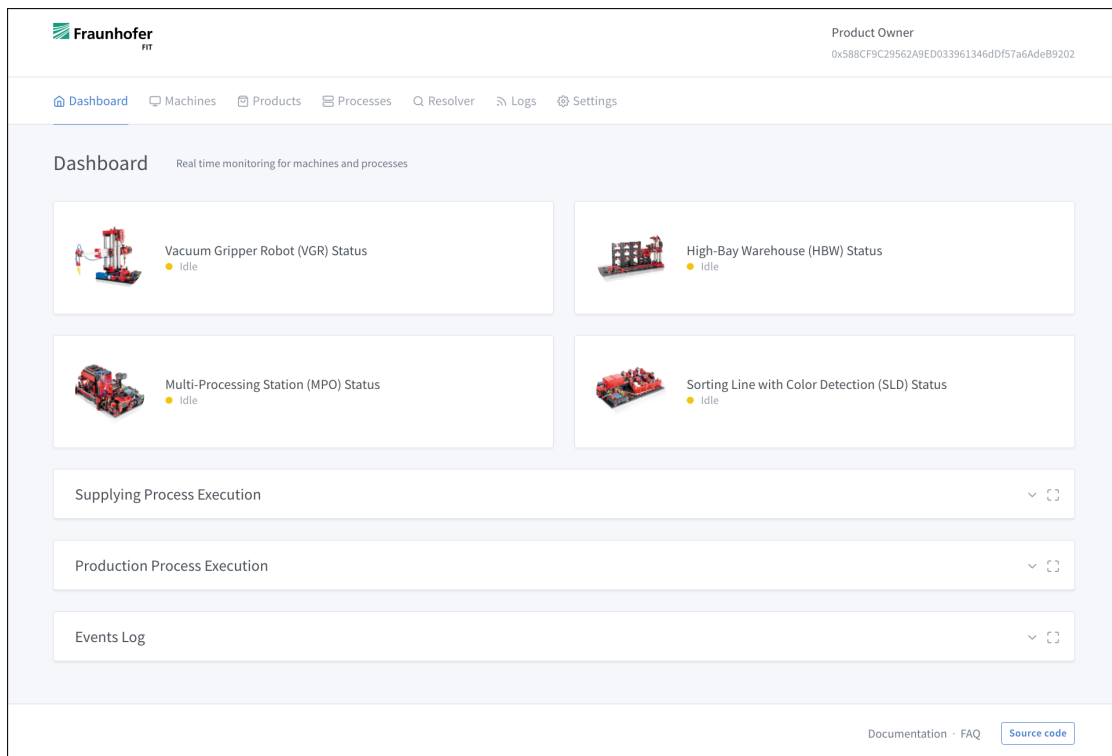


Figure 4.3.: Dashboard UI

Similar to the machine's status, the process status is coming directly from the process smart contract. Figure 4.4 shows the UI components that display the current execution status for the Fischertechnik two processes. The components show each step of the process with the machine's name and the task type. The last piece of information showing in the dashboard is the events log 4.5. The component displays details about the events emitted by the different contracts. Each row in the table corresponds to an event. The rows can be clicked to show more information about the event.

4. System Implementation

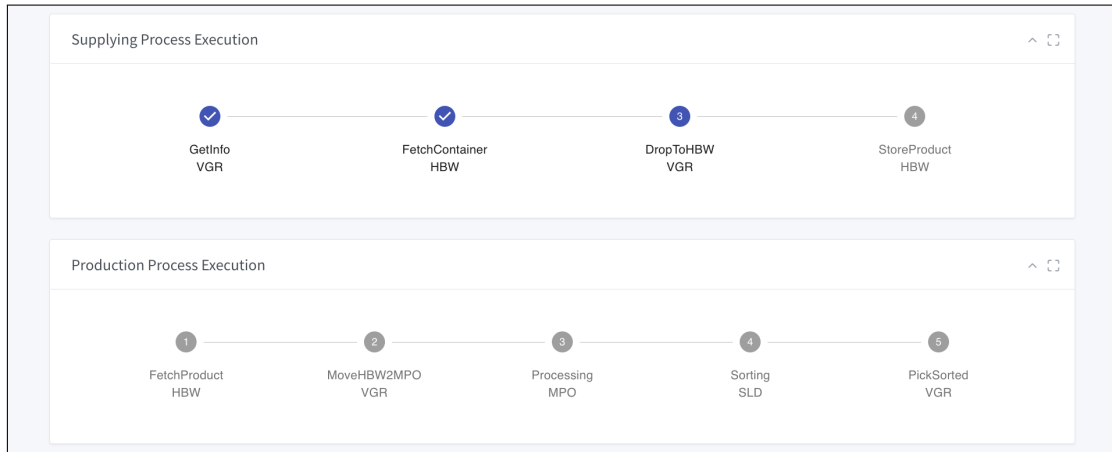


Figure 4.4.: Processes Execution UI Component

Contract Name	Event Name	Block Number	Transaction Hash
> Supplying Process	ProcessStarted	5170	0xd80e606ef8c3aa08b1ea6e37de96c3abf5e401ba2542184e56ea8b28ef8d28ea
> Vacuum Gripper Robot (VGR)	TaskAssigned	5171	0xb6f398c674f79d93d8be79d66a8ea8d0fb23b787c9428d81c2be4c21264b55f2
> Supplying Process	ProcessStepStarted	5171	0xb6f398c674f79d93d8be79d66a8ea8d0fb23b787c9428d81c2be4c21264b55f2
> Vacuum Gripper Robot (VGR)	TaskStarted	5172	0xd04062ae565ac4fd448a4081d9602976a830de2a7aa874cb763b90137597a03
> Vacuum Gripper Robot (VGR)	TaskFinished	5174	0xbd36b379bf7b6194d52199b216d5d1b36209963bf2f74d11452dda08b6b85db6

Rows per page: 5 1-5 of 18

Figure 4.5.: Events Log UI Component

Machine Interface

This interface displays the information stored in the digital twin of the machine. It includes information about the numbers of tasks, readings, and alerts with links to the corresponding pages. Other information about the machine like the DID, the machine owner, the contact address is also presented. Moreover, the interface also lists the authorized processes with an option to unauthorize them. Figure 4.6 shows the interface for the SLD machine. There are three other similar interfaces for the rest of the Fischertechnik machines. This interface can display the information for any machine as long as its smart contract is inherited from the base Machine contract.

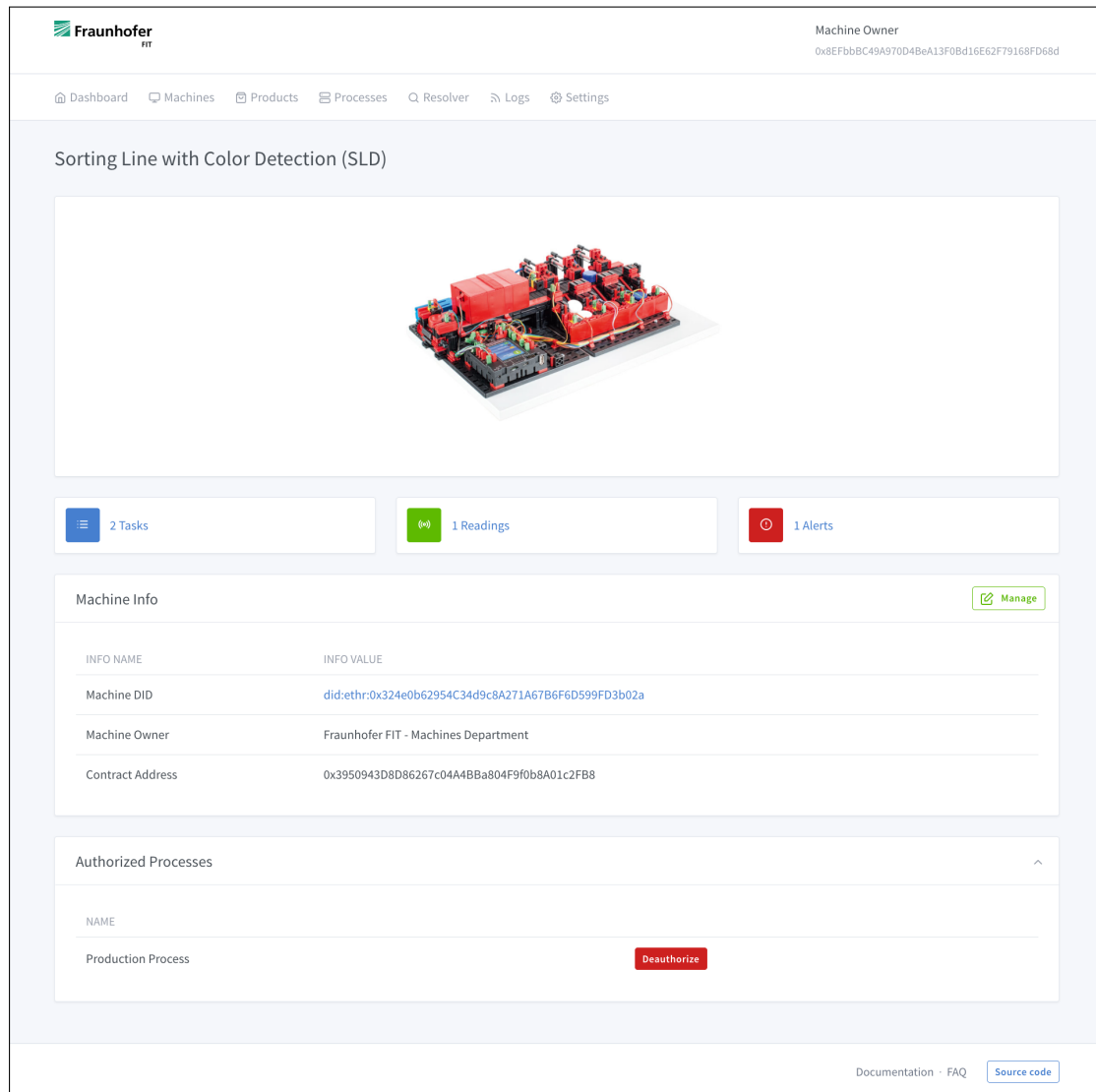
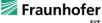


Figure 4.6.: Machine UI

Machine Tasks Interface

In this interface, information about the tasks performed is shown in the form of a table. Figure 4.7 shows the tasks list for the SLD machine. Links to the products and processes are also displayed.

4. System Implementation



Machine Owner

0x8EFbbBC49A970D4BeA13F0Bd16E62F79168FD68d

Dashboard

Machines

Products

Processes

Resolver

Logs

Settings

Sorting Line with Color Detection (SLD)

A list of all tasks performed by this machine

Machine Tasks

TASK ID

TASK NAME

TASK STATUS

STARTING TIME

FINISHING TIME

PRODUCT

PROCESS

PROCESS ID

1

Sorting

Finished Successfully

07/12/2020, 23:20:43

07/12/2020, 23:20:46

Product 1

Production Process

1

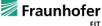
Documentation · FAQ

Source code

Figure 4.7.: Machine Tasks UI


Machine Readings Interface


This interface displays the sensor readings information, including the type of reading, the value of the reading, timestamp, and the task. Figure 4.8 shows the readings interface for the SLD machine.





Machine Owner


0x8EFbbBC49A970D4BeA13F0Bd16E62F79168FD68d


 Dashboard


 Machines

 Products

 Processes

 Resolver

 Logs

 Settings

Sorting Line with Color Detection (SLD)

A list of all sensors readings saved in the smart contract

Machine Readings

Request Reading

ID	TYPE	VALUE	TIME OF MEASURE	TASK ID
1	Brightness	60 %	08/12/2020, 00:41:42	1

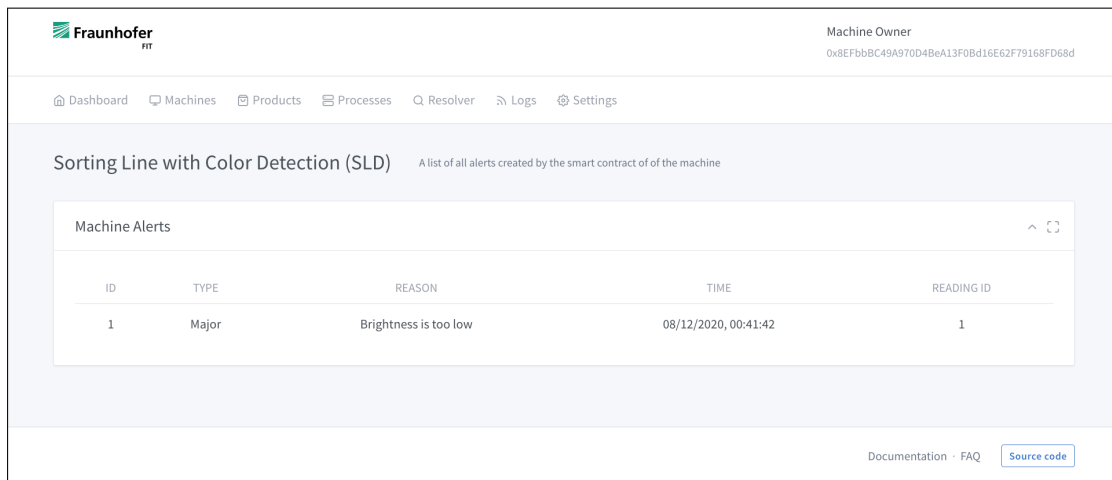
Documentation · FAQ

Source code

Figure 4.8.: Machine Readings UI

Machine Alerts Interface

Like the previous two interfaces, this interface displays the information about the alerts created by the machine's smart contract. For each alert, the reason of the alert, the timestamp, and the reading is shown. Figure 4.9 shows the alerts interface for the SLD machine.



The screenshot displays the 'Machine Alerts' interface for the 'Sorting Line with Color Detection (SLD)' machine. The interface includes a header with the Fraunhofer logo and navigation links (Dashboard, Machines, Products, Processes, Resolver, Logs, Settings). The main content area shows a table of alerts. The table has columns for ID, TYPE, REASON, TIME, and READING ID. A single alert is listed with ID 1, Type Major, Reason Brightness is too low, Time 08/12/2020, 00:41:42, and Reading ID 1. The footer contains links for Documentation, FAQ, and Source code.

ID	TYPE	REASON	TIME	READING ID
1	Major	Brightness is too low	08/12/2020, 00:41:42	1

Figure 4.9.: Machine Alerts UI

Product Interface

This interface shows information about the digital twin of the product. Information about the product like the DID, owner name, owner address, and the creation time is displayed. Furthermore, it shows all the operations performed on this product by different machines. Figure 4.10 shows the information for a product that went through three operations by two machines. For each operation, a link to the corresponding verifiable credential is provided. The verifiable credential resolver interface is used to display the credential's details by clicking on the link.

The screenshot displays the 'Product Digital Twin' interface. At the top, the 'Fraunhofer FIT' logo is on the left, and the 'Product Owner' address '0x588CF9C29562A9ED033961346dDf57a6AdeB9202' is on the right. A navigation bar includes links for Dashboard, Machines, Products, Processes, Resolver, Logs, and Settings. The main content area is titled 'Product Digital Twin' and contains two sections: 'Product Info' and 'Product Operations'.

Product Info

INFO NAME	INFO VALUE
Product DID	did:ethr:did:ethr:0xd07704fD9319a8Bf1F13f4c4a148206612EE5C45
Product Owner	Fraunhofer FIT - Products Department
Product Name	Product 2
Created At	09/12/2020, 18:01:08

Product Operations

ID	OPERATION	RESULT	TIME	MACHINE	TASK ID	
2	NFTTagReading	04963f92186580	09/12/2020, 18:05:54	Vacuum Gripper Robot (VGR)	3	V.Credential
3	ColorDetection	WHITE	09/12/2020, 18:05:54	Vacuum Gripper Robot (VGR)	3	V.Credential
4	Sorting	WHITE	09/12/2020, 18:08:39	Sorting Line with Color Detection (SLD)	2	V.Credential

At the bottom right, there are links for 'Documentation · FAQ' and 'Source code'.

Figure 4.10.: Product UI

Process Interface

This interface displays the process information, including the number of instances, number of machines, and the number of steps. Also, it allows starting the process on a particular product by providing the DID product. If the process is started, the execution can be tracked with the same UI compound used in the dashboard interface. Figure 4.11 shows the interface for the Fischertechnik factory model's production process.

The screenshot displays the 'Production Process' interface within the Fraunhofer IPT system. The header includes the Fraunhofer IPT logo and the 'Product Owner' information: 0x588CF9C29562A9ED033961346dDf57a6AdeB9202. A navigation bar contains links for Dashboard, Machines, Products, Processes, Resolver, Logs, and Settings.

The main content area is titled 'Production Process' and contains three sections:

- Process Info:** A table showing process details.

INFO NAME	INFO VALUE
Process Instance Count	2 Instances
Number of Machines	4 Machines
Number of Steps	5 Steps
- Start Process:** A section for initiating the process. It includes a 'Product DID' label and a text input field with the placeholder 'did:ethr: Ethereum address 0x3f...'. A blue 'Start' button is located at the bottom right of this section.
- Process Execution:** A visual representation of the process flow. It consists of five numbered steps connected by a horizontal line:
 1. FetchProduct HBW
 2. MoveHBW2MPO VGR
 3. Processing MPO
 4. Sorting SLD
 5. PickSorted VGR

The footer of the interface contains links for 'Documentation' and 'FAQ', along with a 'Source code' button.

Figure 4.11.: Process UI

DID Resolver Interface

In this interface, the DID is resolved into the corresponding DID document. The DID document is shown as a JSON object. Figure 4.12 shows the DID document for the VGR machines' DID.

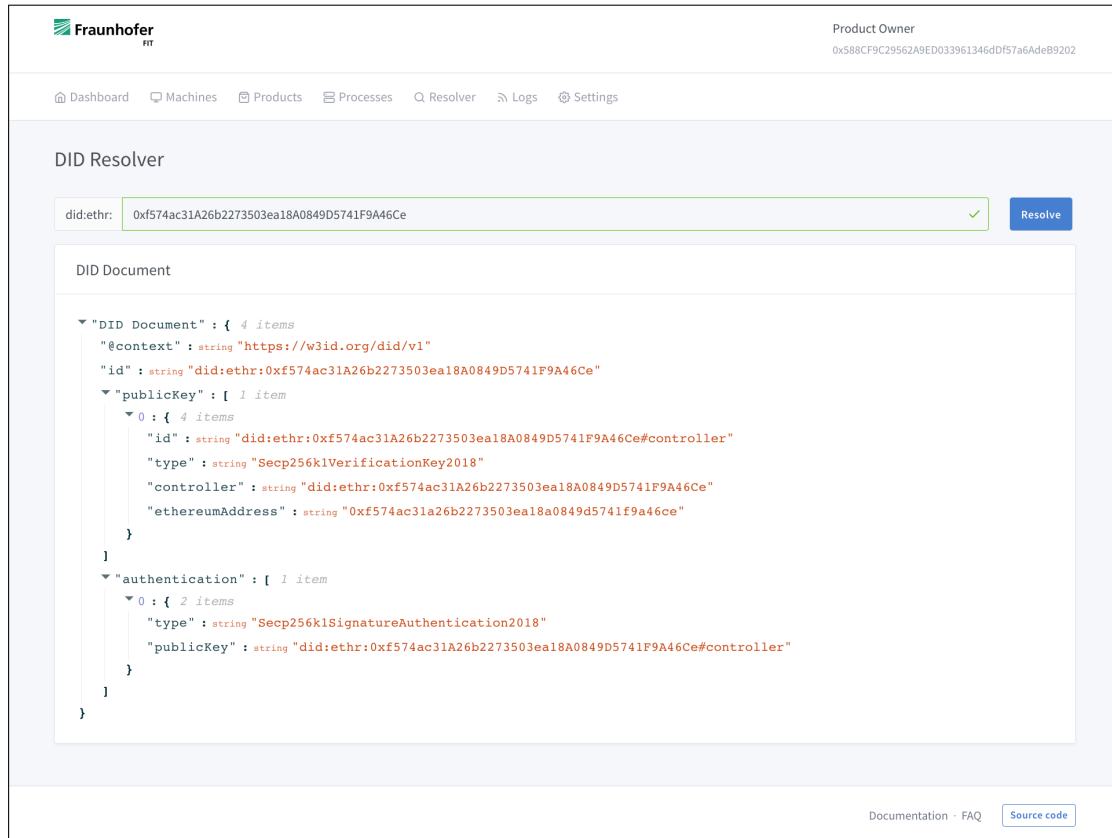


Figure 4.12.: DID Resolver UI

Verifiable Credentials Resolver Interface

This interface resolves and verifies the credentials. Both the encoded and the decoded credential is shown on this interface. Figure 4.13 shows the verifiable credential for NFC Tag Reading operation by the VGR machine on one of the products.

The screenshot displays the 'Verifiable Credentials Resolver' interface. At the top, there's a header with the 'Fraunhofer IIT' logo and 'Product Owner' information. Below the header is a navigation bar with links: Dashboard, Machines, Products, Processes, Resolver, Logs, and Settings. The main content area is titled 'Verifiable Credentials Resolver' and contains two sections: 'Encoded Verifiable Credentials' and 'Decoded Verifiable Credentials'.

The 'Encoded Verifiable Credentials' section shows a long, base64-encoded string. The 'Decoded Verifiable Credentials' section shows a JSON object representing the decoded credential. A green banner at the top of the decoded section indicates 'Signature Verified'.

```

{
  "credential": {
    "payload": {
      "iat": 1607533555,
      "sub": "did:ethr:0xd07704fd9319a8bf1f13f4c4a148206612EE5C45",
      "vc": {
        "@context": [
          "https://www.w3.org/2018/credentials/v1"
        ],
        "type": [
          "VerifiableCredential"
        ],
        "credentialSubject": {
          "productOperation": {
            "operationName": "NFC Tag Reading",
            "operationResult": "04963f92186580",
            "taskId": "3",
            "operationID": "2"
          }
        }
      }
    },
    "iss": "did:ethr:0xf574ac31a26b2273503ea18a0849d5741f9a46ce"
  },
  "doc": {
    "issuer": "did:ethr:0xf574ac31a26b2273503ea18a0849d5741f9a46ce",
    "signer": {
      "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ0eXQoIjE2MDc1MzMTU0InN1YiI..."}
  },
  "verifiableCredential": {
    ...
  }
}

```

At the bottom right, there are links for 'Documentation' and 'FAQ', and a 'Source code' button.

Figure 4.13.: Verifiable Credentials Resolver UI

4.6.2. Source Code

The source code of the implementation is divided into two parts. The first one is the source code for the Fischertechnik factory model [Ghaa]. It is a fork of the source code provided by the manufacturing company. We made the necessary changes to implement the presented scenarios. The second part is the source code of the smart contracts, the gateway, and the web application [Ghab]. Inside the source code repositories, the readme file contains technical details on how to get started with the code and run it and other implementation aspects that were not discussed inside this thesis

4.7. Summary

This chapter introduced the technical implementation of the conceptual design presented earlier in chapter 3. It started by describing the factory infrastructure's implementation and introducing the Fischertechnik factory model, which was used to build the prototype. All the Fischertechnik model aspects were explained in detail, including the machines, their task types, and processes. The chapter also introduced the actual architecture with all the compounds and the information flows between them. Furthermore, the modeling of the machines, products, and processes is discussed as concrete smart contracts with the tools used to build and operate them. The implementation results were shown in the form of the user interfaces of the web application (DApp). This chapter answered the second and third research questions RQ2 and RQ3 from the technical and technological point of view. Based on the information provided in this chapter, the following chapter presents the evaluation for the conceptual design and the prototype implementation.

5. Evaluation

In light of the results shown in the last chapter, this chapter presents an evaluation of the conceptual design and the prototype. This evaluation answers the fourth research question, RQ4, about the feasibility of the solutions. We followed Venable et al.'s evaluation framework for Design Science Research (DCR) [VPB12]. The framework guides the researchers in choosing an appropriate strategy to evaluate any artifact design or design theories in the DCR. According to this framework, three factors play a role in determining an evaluation strategy:

- The evaluation's timing: ex-ante (before artifact construction) versus ex-post evaluation (after artifact construction).
- The type of artifact: product (technologies such as tools, diagrams, or software) versus process (methods or procedures).
- The evaluation settings: artificial evaluation (laboratory setting in a simulated environment) versus naturalistic (field settings in a real environment).

In our case, the artifact is a product. It is the conceptual design and the modeling of the system proposed in the third chapter. The evaluation goal is to ensure the rigor and efficiency of the artifact. Due to the innovative nature of the blockchain-based solution, we follow an artificial ex-post strategy. Many evaluation methods work for this strategy, including simulation, controlled case studies, and theoretical arguments [VPB12]. Our evaluation is based on the software prototype developed for the Fischertechnik factory model case study described in the fourth chapter.

5.1. Solution Evaluation

This section evaluates the modeling and design of the proposed solution. As we showed in the previous two chapters, blockchain can build digital twins for machines and products and control M2M communication. The prototype we built proved that the blockchain can be an active compound in the manufacturing industry's infrastructure. Table 5.1 revisits the system objectives listed in the third chapter and specifies to which extent the objectives were fulfilled. The Table shows, all objectives were fulfilled. However, there are some limitations to some of these objectives. In the following subsection, we will explain these limitations.

Objective	Fulfillment
Build machine digital twin	✓
Build product digital twin	✓
Use Blockchain-based identities for twins	✓
Implement M2M communication through blockchain	✓
Ensure the design and the modeling is generic	✓

Table 5.1.: Objectives Fulfillments

5.1.1. Digital Twin Limitations

The main limitation in our modeling of the digital twin is the inclusiveness of the data. Our modeling of the machines and the products did not consider the entire lifecycle of the entity. We used the term digital twin even though the modeling only includes some information about the machines and products.

The machine modeling in our solution only took into consideration the operational phase of the machine lifecycle. Our machine's digital twin includes information like tasks, sensor data, and alerts. However, even for the operational phase, some information is still missing from the modeling. Information about machine maintenance activities and operations is an essential part of a full machine digital twin. Regarding the product digital twin, our modeling only considered the manufacturing phase of the product lifecycle. It includes information about the operations performed on the physical product. This information is considered a tiny fraction of the information needed to construct a full digital twin of a product.

5.1.2. M2M Communication Limitations

The processes represent M2M communication. As we mentioned earlier in the modeling section, our process model is simple. It is a sequential process consists of multiple steps performed by different machines. However, as we showed in one of the use case processes, two steps can be performed in parallel if they do not depend on each other. One of our modeling's most significant limitation is the need to hard code the process steps as functions in the smart contract. Another critical aspect of the smart contract process is that the machines used in the process have to be assigned manually. The future work section presents a better approach to overcome these limitations.

5.1.3. Technical Limitations

There are many technical limitations worth discussing, and all of them are inherited from the blockchain nature. The technical limitations include but are not limited to the

gateway, throughput, latency, and privacy.

Gateway

Due to the underlying implementation of the blockchain, the gateway is needed. The machines can not access the blockchain unless they become a node of the network. Therefore, the machines must use the gateway to run their clients, which interact with the blockchain. Having the gateway imposes some limitations as the machine is not directly connected to the blockchain. The connection between the machine and the gateway must be secure; otherwise, the system is prone to various security attacks.

Throughput and Latency

The throughput is the number of transactions being processed during a specific time. The latency is the time until the transaction is processed. These represent two of the most well-known challenges for blockchain, and many factors play a role in determining these two measures. The blockchain's underlying implementation and the consensus mechanisms can significantly increase/decrease the throughput and the latency. These measures affect our system's performance as all the machine operations like starting or finishing a task involves sending a transaction to the blockchain. While working on the prototype, we noticed a considerable difference between the latency between different configurations for the Quorum blockchain. Therefore, our system's performance depends entirely on the blockchain implementation, and it is less than any centralized system.

Privacy

Another big challenge to the blockchain-based application is privacy. The immutability and transparency nature of the blockchain makes storing private data on-chain a challenge. Also, as for the previous limitations, some blockchain implementations have overcome this challenge to some extent. However, we did not consider the privacy aspect of the data stored in the digital twin of the machines and products. All information is publicly readable. Even though we used the Quorum blockchain, which supports running a private and permissioned network, we did not use these features and decided to put privacy outside the system's scope.

5.2. Prototype Evaluation

5.2.1. Software Testing

This section explains the testing approach used in the prototype software. The smart contracts compound is the core compound, and therefore we ensured to use extensive testing while implementing it. Other compounds, such as the frontend and the gateway,

do not have the same testing level. To test the smart contracts, we used the OpenZeppelin Test Environment library [Opea]. It provides a convenient environment for testing smart contracts on both the unit and integration test levels. The library takes care of running up a local ganache-powered blockchain and other tools to be used by the test cases. For each test case, a fresh instance of the contract is deployed, and then the test code will be executed. This allows testing the functionality in insulation without introducing side effects. The test coverage for our solidity source code is 100%. Writing automated unit and integration tests benefited us the following benefits:

- Save development time as testing manually involves first deploying the contracts to blockchain and then writing additional code to interact with them.
- Validate the behaviors within smart contracts. With automated testing, it is easier to check the revert and boundary cases.
- Check and validate interactions between multiple contracts.
- Ensure that newly added functionality does not break the old functionality.

5.2.2. Quantitative Evaluation

Since our prototype is blockchain-based, the quantitative evaluation focus on blockchain-related metrics. These metrics vary depending on the blockchain technical implementation. The latency and gas consumption are two of the most common metrics used to evaluate blockchain-based applications' performance. We already talked about the latency and how it could affect our solution. As the latency depending on the underlying blockchain implementation, we did not provide a quantitative evaluation. However, gas consumption can be evaluated and calculated independently from the underlying blockchain.

The gas consumption measures the execution cost in units of gas. Each operation execution in a smart contract costs a fixed amount of gas, independent of the underlying Ethereum network. Our evaluation does not consider the exchange rate between the gas and the Ether (the currency of Ethereum), often called gas price. The gas price in the permissioned blockchain like Quorum is always zero. Nevertheless, calculating the gas consumption for the common functions calls clarifies how computationally expensive a transaction is.

The most expensive operation in operating the smart contracts is the deployment operation. Table 5.2 shows the deployment cost for each contract of the prototype in addition to the smart contract size. It is evident from the numbers that the deployment cost is proportional to the size of the contract. Our modeling relies on inheritance from base contracts making the subcontracts having a larger size. If the sub-contract is already big due to internal business logic, the maximum size might be reached, making the contract deployment impossible. It worth mentioning that on the public Ethereum

Smart Contract	Deployment Cost	Contract Size
Product	3713586 gas	16.50 KiB
Registry	1204354 gas	5.10 KiB
Supplying Process	2588280 gas	10.64 KiB
Production Process	2671109 gas	10.97 KiB
VGR Machine	5065543 gas	21.78 KiB
HBW Machine	4987150 gas	21.38 KiB
MPO Machine	4873591 gas	20.86 KiB
SLD Machine	5005507 gas	21.28 KiB

Table 5.2.: Contracts Deployment Gas Cost

networks, the contract size limit is 24 KiB. However, for the Quorum network, this limit is configurable and can be increased. Large contracts can be divided into smaller contracts with functions calls between them to overcome the size limit.

Other functionalities in the system consume less gas than deployment. Table 5.3 provides the average cost for the main smart contract functions. The calculation provided in the table is for the supplying and production processes, as explained in the implementation chapter. Start task function has a relatively consistent cost as this function is not related to the task type, and it always writes the same information into the smart contract. The cost varies in the finish task function because it writes the product's operations into the product smart contract, which means additional computation and additional cost. The process step function calls the assign task function and writes additional information depending on the task type, and therefore, the cost also varies.

Functions	Average Execution Cost
Start Task	65115 gas
Finish Task	152035 gas
Execute Process Step	311739 gas

Table 5.3.: Operations Average Cost

5.3. Summary

This chapter provided an evaluation of the work done in this thesis. The evaluation strategy for the generic conceptual design was to build a prototype for the Fischertechnik use case. Based on the work done in the prototype, we evaluate the design and the modeling of the solution. The chapter showed the result of this evaluation by discussing the limitations on both the conceptual and technical levels. Also, the evaluation considered the testing approach used in testing the smart contract compound. In the end, a quantitative evaluation of gas consumption has been presented. The evaluation presented in this chapter answered the fourth research question, RQ4, about the solution's feasibility.

6. Conclusion and Future Work

6.1. Conclusion

This thesis investigated the applicability of blockchain in the manufacturing industry. First, we introduced the relevant topics like industry 4.0, digital twins, blockchain, and self-sovereign identity. Afterward, we conducted a literature review of the industrial applications of blockchain. The review showed that researchers already used blockchain to build decentralized manufacturing solutions. We discussed and evaluated the numbers of the studies and made a comparison between them. We decided to focus our work on the M2M communication and digital twin solutions based on the literature review results.

Our main contribution was building a generic conceptual design for a system that utilizes blockchain and smart contract technologies to implement M2M communication and digital twins for machines and products. The conceptual design discussed how machines, products, and manufacturing processes are modeled as smart contracts. The modeling defined which information is stored in the digital twins of machines and products. We showed how the digital twin and the physical machine could interact and share information. We also spouted the blockchain-based identities for both the machines and the products. The design also discussed how M2M communication is implemented and executed through the blockchain.

To validate our solution, we developed a prototype that implements the conceptual design using the Ethereum network as a blockchain. We used the Fischertechnik factory model as hardware. It provided a fully functional factory simulation consisting of four machines and two manufacturing processes. The prototype included a distributed web application (DApp) that allows different actors to interact with the machines, products, and processes through the blockchain. It provided dashboards to monitor and control manufacturing processes that run autonomously by the smart contracts. We evaluated the design and the modeling based on the work done while developing the prototype. The evaluation chapter reviewed and discussed all the limitations and weaknesses. Besides, it addressed all the challenges we faced while transforming the design concepts into a functional prototype.

6.2. Answer to Research Questions

RQ1: How can blockchain technologies support M2M communication and digital-twins solution in the manufacturing industry?

As we showed in the literature review discussion, blockchain and other related technologies benefit the industry in many different ways. The use cases depend on the type of information and data stored and processed by the blockchain. However, blockchain can add the following advantages regardless of the use case: trust, security, and decentralization. The blockchain can add these advantages to the manufacturing industry because of its unique characteristics like immutability, traceability, and reliability. Our work showed how blockchain could be used in the M2M communication to automate the execution of manufacturing processes and store related information to build the digital twin representation for machines and products while providing the benefits and advantages mentioned earlier.

RQ2: What are approaches for the design and development of blockchain-based solutions for M2M communication and digital identity?

When building a blockchain-based system, there are some design decisions have to be made. In chapter 3 and 4, we have shown and justified all the decisions we made while designing the solution and developing the prototype. Our approach used the blockchain as a computational and storage component of the system. All digital identities information, M2M communication messages, other related information about the machine's tasks and products are within the smart contract as on-chain data to get the most out of the blockchain features.

RQ3: How can machines, products, and manufacturing processes be modeled on the blockchain?

The conceptual design discussed in chapter 3 showed and illustrated details on how to model these entities using smart contracts. Each machine is modeled in the blockchain as one smart contract, whereas all the products are modeled as one smart contract for simplicity. These contracts store and manage information about the machines and products and represent their digital twins. Similarly, each manufacturing process is modeled as one smart contract and responsible for executing the corresponding process by communicating messages between the machines' digital twins.

RQ4: How feasible is it to utilize the blockchain for M2M communication and digital twin solutions?

To answer this question, we evaluated both the conceptual design and the prototype in chapter 5. The evaluation showed limitations in both of them. These limitations are either caused by the blockchain nature or by the thesis's restriction in scope and time.

The digital twin modeling is limited only to a subset of the information usually available in the digital twins of machines and products. The M2M modeling only took into consideration a specific type of manufacturing processes with certain conditions. Other blockchain limitations like privacy, throughput, and latency also had influenced the solution's feasibility. Despite all the limitations and the challenges, we believe that the conceptual design and the prototype disclosed in this work proves blockchain solutions' feasibility for the manufacturing industry.

6.3. Future Work

Different aspects could be improved upon in our work, which forms a basis for future work. Each one of the limitations listed in the evaluation chapter is a groundwork for research work. Our modeling for the digital twin of the machine only included the operational aspect of the machine. However, there is much information directly related to the machine's operational conditions, like maintenance operations. Extending the digital twin information by considering other aspects of the machine will allow building more services that fit the industry 4.0 needs. M2M communication represents another area of interest. It was modeled in our design to fit particular types of manufacturing processes. Additional work needs to be done in order to make it applicable to other types of processes. Another direction to improve the process modeling is to use model-driven engineering. It allows auto-generation of the smart contract source code instead of writing it manually.

Even using our conceptual design without modification can still provide a foundation for building other services and applications. The smart contracts of the machines and the processes can be extended to enforce any custom logic. It can be a business logic to implement the payment between the machine owner and the product owner. Alternatively, it can be used to implement the warranty agreement between the machine owner and the maintenance company. Lastly, some technical aspects of the solution can be improved. For example, the interaction between the machine and its digital twin can be optimized by eliminating the gateway role and letting the machine interact directly with the blockchain. One more direction is to take the privacy of the information stored within the blockchain into consideration while building M2M communication and digital twin solutions.

A. Use Cases Flows

A.1. Start Task

Description	Mark the task as started and store the starting time of the task.
Actors	Machine and its client in the gateway.
Precondition	The task is assigned to the machine by a process.
Post Conditions	The task is marked as started, and the starting time is saved in the machine's digital twin.
Basic Flow	<ol style="list-style-type: none">1. The client receives the task assigned event.2. The client calls the start task function in the machine's smart contract by sending a transaction signed with the machine key.3. The smart contract of the machine receives the transaction and verifies it.4. The smart contract writes the current time as the starting time for the given task and changes its state.5. The smart contract emits TaskStarted event.6. The client receives the transaction hash.7. The client notifies the machine to start executing the given task.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction will be reverted.• If the task does not exist, the transaction will be reverted.

Table A.1.: Start Task Use Case

A.2. Finish Task

Description	Mark the task as finished and store the finishing time of the task.
Actors	Machine and its client in the gateway.
Precondition	The task is already started by the machine.
Post Conditions	The task is marked as finished, and the finishing time is saved in the machine's digital twin.
Basic Flow	<ol style="list-style-type: none">1. The client receives a message from the physical machine, indicating finishing one task with status.2. The client calls the finish task function in the machine's smart contract by sending a transaction signed with the machine key. The parameters are finishing status and custom information depending on the task type.3. The smart contract of the machine receives the transaction and verifies it.4. The smart contract writes the current time as the finishing time for the given task and changes its state.5. The smart contract emits TaskFinished event.6. The client receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction will be reverted.• If the task does not exist, the transaction will be reverted.• If the task does not start, the transaction will be reverted.

Table A.2.: Finish Task Use Case

A.3. Save Product Operation

Description	Save the result of an operation performed on a particular product.
Actor	Machine and its client in the gateway.
Precondition	The corresponding task is already assigned and started.
Post Conditions	The information is saved in the digital twin of the product.
Basic Flow	<ol style="list-style-type: none"> 1. The physical machine sends a message to the client contains the operation information. 2. The client calls the save operation function in the corresponding contract of the machine by sending a transaction signed with the machine key. 3. The machine's smart contract receives the transaction and verifies it. 4. The machine's smart contract calls the save operation function in the product contract. 5. The product contract verifies the transaction and checks the machine authorization. 6. The product contract saves the operation information in the digital twin of the given product. 7. The client receives the transaction hash.
Exceptions	<ul style="list-style-type: none"> • If the transaction verification fails in the machine contract of the product contract, the transaction will be reverted. • If the product does not exist, the transaction will be reverted.

Table A.3.: Save Product Operation Use Case

A.4. Save Reading

Description	Save the numerical value of sensor reading.
Actor	Machine and its client in the gateway.
Precondition	None.
Post Conditions	The value of the reading with a timestamp is saved in the digital twin of the machine.
Basic Flow	<ol style="list-style-type: none">1. The client asks the machine for a specific sensor reading.2. The physical machine sends the value of the reading with a timestamp to the client.3. The client calls the save reading function in the machine's contract by sending a transaction signed with the machine key.4. The smart contract of the machine receives the transaction and verifies it.5. The smart contract writes the information.6. The client receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails in the machine contract, the transaction will be reverted.

Table A.4.: Save Reading Use Case

A.5. Save Alert

Description	Save an alert in the digital twin of the machine.
Actor	Machine's smart contract (machine digital twin)
Precondition	None.
Post Conditions	The alert is saved with a timestamp and reason for the alert.
Basic Flow	<ol style="list-style-type: none">1. The smart contract of the machine checks the value of the reading while it is being saved.2. The value of the reading does not comply with the expected range of values.3. The smart contract calls the internal save alert function.4. The smart contract writes the information of the alert (reading ID, timestamp, reason).
Exceptions	None

Table A.5.: Save Alert Use Case

A.6. Kill Task

Description	Mark the task as finished unsuccessfully and store the finishing time of the task.
Actors	Machine Owner.
Precondition	The task is already started by the machine.
Post Conditions	The task is marked as finished unsuccessfully, and the finishing time is saved in the machine's digital twin.
Basic Flow	<ol style="list-style-type: none">1. The machine owner uses the user interface provided by the web application to kill a task.2. The web application calls the kill task function in the machine's smart contract by sending a transaction signed with the machine owner key.3. The smart contract of the machine receives the transaction and verifies it.4. The smart contract writes the current time as the finishing time for the given task and changes its state.5. The smart contract emits TaskKilled event.6. The web application receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction will be reverted.• If the task does not exist, the transaction will be reverted.

Table A.6.: Kill Task Use Case

A.7. Get New Reading

Description	Ask the machine to store a new reading from a specific type.
Actors	Machine Owner.
Precondition	None
Post Conditions	NewReading is emitted from the machine's smart contract.
Basic Flow	<ol style="list-style-type: none">1. The machine owner uses the user interface provided by the web application to request a new reading.2. The machine owner chooses the type of reading.3. The web application calls the get reading function in the machine's smart contract by sending a transaction signed with the machine owner key.4. The smart contract of the machine receives the transaction and verifies it.5. The smart contract emits NewReading event.6. The machine's client receives the event, and the save new reading use case is started.7. The web application receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction will be reverted.

Table A.7.: Get New Reading Use Case

A.8. Authorize/Unauthorize Process

Description	Allow/prevent a process assigning a task for the machine.
Actors	Machine Owner.
Precondition	None
Post Conditions	Process can/can not assign tasks to the machine.
Basic Flow	<ol style="list-style-type: none">1. The machine owner uses the user interface provided by the web application to authorize/unauthorize a process.2. The machine owner enters the address of the process.3. The web application calls the authorize/unauthorize function in the machine's smart contract by sending a transaction signed with the machine owner key.4. The smart contract of the machine receives the transaction and verifies it.5. The smart contract adds/removes the address of the process from the list of authorized processes.6. The web application receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction would be reverted.

Table A.8.: Authorize/Unauthorize Process Use Case

A.9. Execute Process Step

Description	Execute a process step that involves assigning the task for the machine.
Actors	Process with its client in the gateway.
Precondition	The corresponding process instance exists, and the step is executed in its right order.
Post Conditions	The step status is changed, and the task is assigned to the machine.
Basic Flow	<ol style="list-style-type: none"> 1. The client receives task finished events or process started events. 2. According to the event's information, the client calls the step function in the process's smart contract by sending a transaction signed with the process owner key. 3. The smart contract of the process receives the transaction and verifies it. 4. Depending on the code written in the step function, the smart contract of the process executes the code and calls the smart contract of the machine responsible for this step. 5. The smart contract of the machine creates a new task, saves information about the process in the task, and emits TaskAssigned event. 6. Once the smart contract of the process finishes executing the step function, the client receives the transaction hash. 7. The machine's client receives the TaskAssigned event, and the start task is started.
Exceptions	<ul style="list-style-type: none"> • If the transaction verification fails, the transaction will be reverted.

Table A.9.: Execute Process Step Use Case

A.10. Create Product

Description	Create virtual product (digital twin) for a physical product.
Actors	Product Owner.
Precondition	None
Post Conditions	The product is created and saved in the product's smart contract.
Basic Flow	<ol style="list-style-type: none">1. The product owner uses the user interface provided by the web application to create a product.2. The machine owner enters the DID (address) of the product.3. The web application calls the create product function in the product smart contract by sending a transaction signed with the sender key.4. The contract receives the transaction and verifies it.5. The contract saves the creation timestamp and makes the transaction's sender as the product owner.6. The web application receives the transaction hash.
Exceptions	<ul style="list-style-type: none">• If the transaction verification fails, the transaction will be reverted.

Table A.10.: Create Product Use Case

A.11. Start Process

Description	Create a new process instance and start executing it for a particular product.
Actors	Product Owner.
Precondition	The provided product exists.
Post Conditions	The process instance is created and started the execution of the process.
Basic Flow	<ol style="list-style-type: none"> 1. The product owner uses the user interface provided by the web application to start a process. 2. The machine owner enters the DID (address) of the product. 3. The web application calls the start process function in the process's smart contract by sending a transaction signed with the product owner key. 4. The contract receives the transaction and verifies it. 5. The contract creates a new process instance with the information about the product owner and the product. 6. The contract emits the ProcessStarted event. 7. The web application receives the transaction hash. 8. The process's client receives the ProcessStarted event and executes the first process step use case.
Exceptions	<ul style="list-style-type: none"> • If the transaction verification fails, the transaction will be reverted. • If the product does not exist, the transaction will be reverted.

Table A.11.: Start Process Use Case

B. Fischertechnik Factory Model Machines

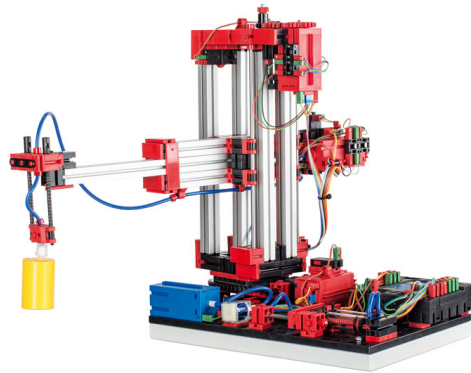


Figure B.1.: Vacuum Gripper Robot (VGR)

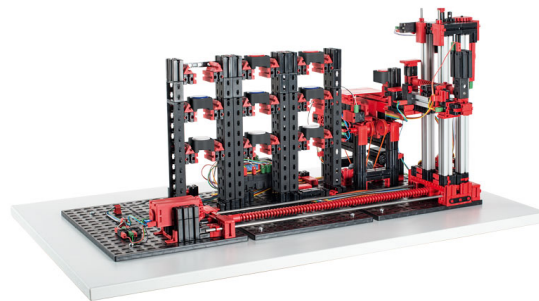


Figure B.2.: High-Bay Warehouse (HBW)

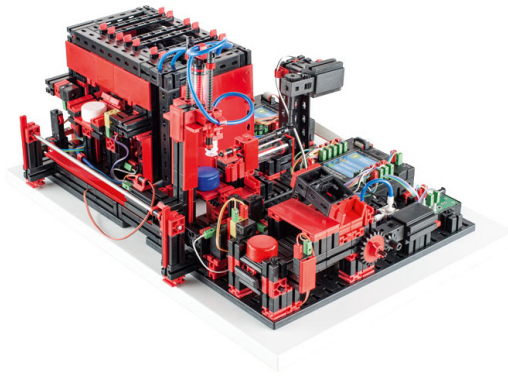


Figure B.3.: Multi-Processing Station with Oven (MPO)

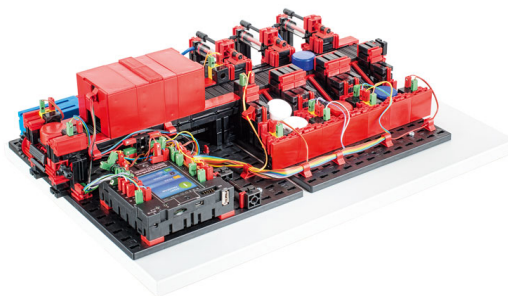


Figure B.4.: Sorting Line with Color Detection (SLD)

C. Smart Contracts Class Diagrams

C.1. Machine

The diagram of the machine smart contract is large and cannot fit one page. Therefore, we split it into two diagrams. The first one contains modifiers, events, and structs. The second one shows the state variables and functions.

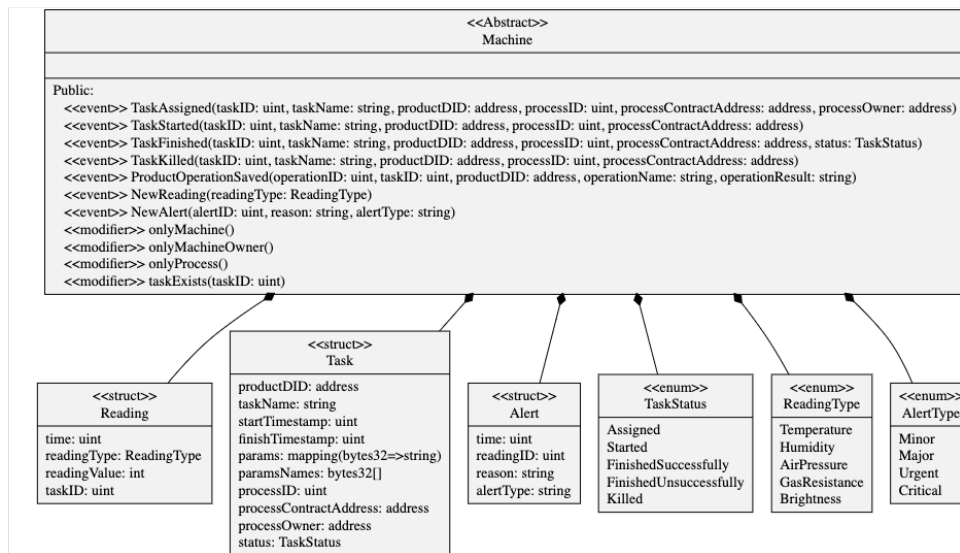


Figure C.1.: Machine Modifiers, Events, and Structs



Figure C.2.: Machine State Variables and Functions

C.2. Product



Figure C.3.: Product Smart Contract

C.3. Process

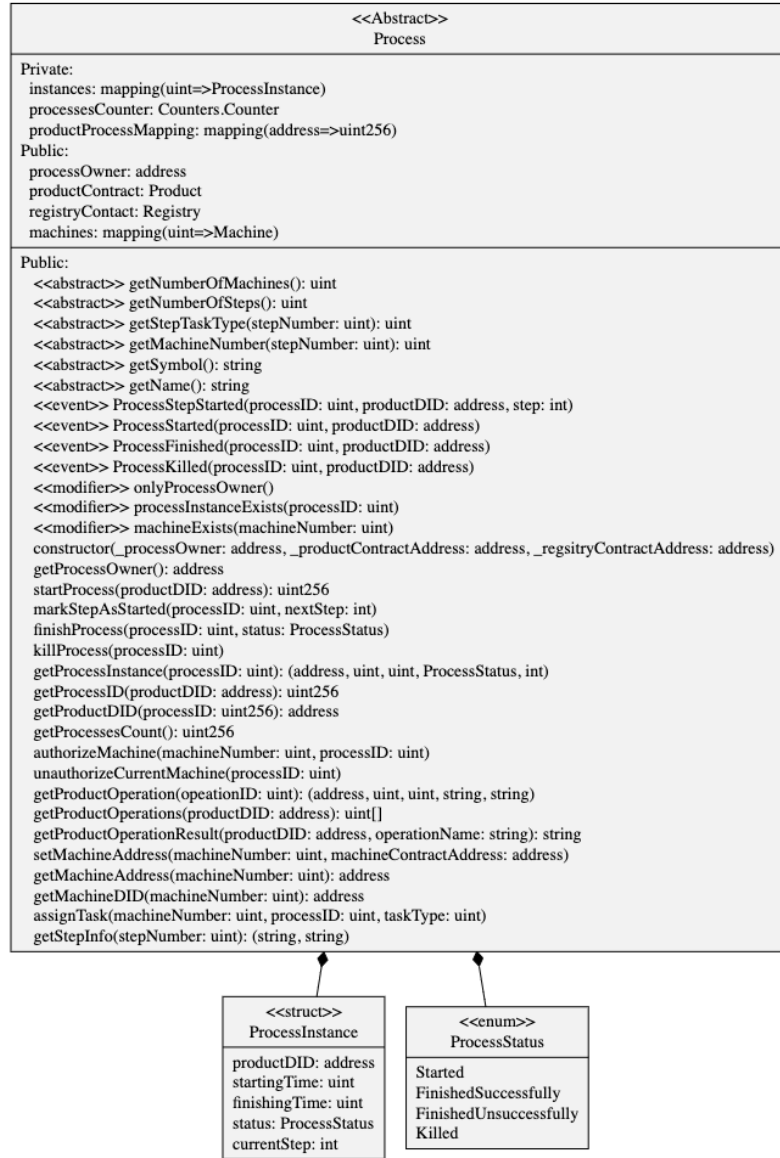


Figure C.4.: Process Smart Contract

C.4. Ethereum DID Registry

EthereumDIDRegistry
Public: owners: mapping(address=>address) delegates: mapping(address=>mapping(bytes32=>mapping(address=>uint))) changed: mapping(address=>uint) nonce: mapping(address=>uint)
Internal: checkSignature(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, hash: bytes32): address changeOwner(identity: address, actor: address, newOwner: address) addDelegate(identity: address, actor: address, delegateType: bytes32, delegate: address, validity: uint) revokeDelegate(identity: address, actor: address, delegateType: bytes32, delegate: address) setAttribute(identity: address, actor: address, name: bytes32, value: bytes, validity: uint) revokeAttribute(identity: address, actor: address, name: bytes32, value: bytes)
Public: <<event>> DIDOwnerChanged(identity: address, owner: address, previousChange: uint) <<event>> DIDDelegateChanged(identity: address, delegateType: bytes32, delegate: address, validTo: uint, previousChange: uint) <<event>> DIDAttributeChanged(identity: address, name: bytes32, value: bytes, validTo: uint, previousChange: uint) <<modifier>> onlyOwner(identity: address, actor: address) identityOwner(identity: address): address validDelegate(identity: address, delegateType: bytes32, delegate: address): bool changeOwner(identity: address, newOwner: address) changeOwnerSigned(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, newOwner: address) addDelegate(identity: address, delegateType: bytes32, delegate: address, validity: uint) addDelegateSigned(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, delegateType: bytes32, delegate: address, validity: uint) revokeDelegate(identity: address, delegateType: bytes32, delegate: address) revokeDelegateSigned(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, delegateType: bytes32, delegate: address) setAttribute(identity: address, name: bytes32, value: bytes, validity: uint) setAttributeSigned(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, name: bytes32, value: bytes, validity: uint) revokeAttribute(identity: address, name: bytes32, value: bytes) revokeAttributeSigned(identity: address, sigV: uint8, sigR: bytes32, sigS: bytes32, name: bytes32, value: bytes)

Figure C.5.: Ethereum DID Registry Smart Contract

C.5. Registry

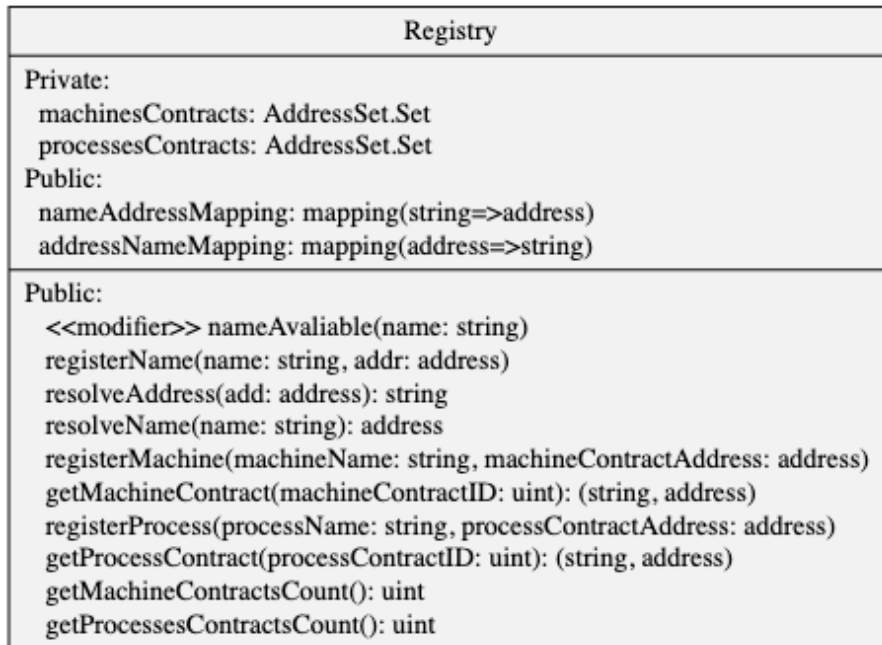


Figure C.6.: Registry Smart Contract

List of Figures

2.1. VCs Components and Information Flows [VC-]	8
3.1. High-Level System Overview	17
3.2. Twins Interaction	22
3.3. Process Example	25
3.4. Use Cases Diagram	27
4.1. Fischertechnik Factory Model	30
4.2. System Architecture	33
4.3. Dashboard UI	41
4.4. Processes Execution UI Component	42
4.5. Events Log UI Component	42
4.6. Machine UI	43
4.7. Machine Tasks UI	44
4.8. Machine Readings UI	44
4.9. Machine Alerts UI	45
4.10. Product UI	46
4.11. Process UI	47
4.12. DID Resolver UI	48
4.13. Verifiable Credentials Resolver UI	49
B.1. Vacuum Gripper Robot (VGR)	73
B.2. High-Bay Warehouse (HBW)	73
B.3. Multi-Processing Station with Oven (MPO)	74
B.4. Sorting Line with Color Detection (SLD)	74
C.1. Machine Modifiers, Events, and Structs	75
C.2. Machine State Variables and Functions	76
C.3. Product Smart Contract	77
C.4. Process Smart Contract	78
C.5. Ethereum DID Registry Smart Contract	79
C.6. Registry Smart Contract	80

List of Tables

2.1. Related Work Comparison	13
3.1. Machine Digital Twin Information	19
3.2. Machine Smart Contract Events	21
3.3. Product Digital Twin Information	23
3.4. Process Smart Contract Events	24
5.1. Objectives Fulfillments	52
5.2. Contracts Deployment Gas Cost	55
5.3. Operations Average Cost	55
A.1. Start Task Use Case	61
A.2. Finish Task Use Case	62
A.3. Save Product Operation Use Case	63
A.4. Save Reading Use Case	64
A.5. Save Alert Use Case	65
A.6. Kill Task Use Case	66
A.7. Get New Reading Use Case	67
A.8. Authorize/Unauthorize Process Use Case	68
A.9. Execute Process Step Use Case	69
A.10.Create Product Use Case	70
A.11.Start Process Use Case	71

Bibliography

- [Afa+18] M. Y. Afanasev, Y. V. Fedosov, A. A. Krylova, and S. A. Shorokhov. "An application of Blockchain and Smart Contracts for Machine-to-Machine Communications in Cyber-Physical Production Systems." In: May (2018). doi: 10.1109/ICPHYS.2018.8387630.
- [Al-19] J. Al-jaroodi. "Industrial Applications of Blockchain." In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (2019), pp. 550–555. doi: 10.1109/CCWC.2019.8666530.
- [Ang+18] A. Angrish, B. Craver, M. Hasan, and B. Starly. "A Case Study for Blockchain in Manufacturing: "fabRec": A Prototype for Peer-to-Peer Network of Manufacturing Nodes." In: *Procedia Manufacturing* 26 (2018), pp. 1180–1192. issn: 23519789. doi: 10.1016/j.promfg.2018.07.154. arXiv: 1804.01083.
- [Asp] Aspen-Plus. URL: <https://www.aspentech.com/en/products/engineering/aspen-plus> (visited on 01/15/2021).
- [Bai+19] L. Bai, M. Hu, M. Liu, and J. Wang. "BP-IIoT: A Light-Weighted Blockchain-Based Platform for Industrial IoT." In: *IEEE Access* 7 (2019), pp. 58381–58393. issn: 21693536. doi: 10.1109/ACCESS.2019.2914223.
- [Bar+19a] A. V. Barenji, Z. Li, W. M. Wang, G. Q. Huang, and A. David. "Blockchain-based ubiquitous manufacturing : a secure and reliable cyber-physical system." In: *International Journal of Production Research* 0.0 (2019), pp. 1–22. doi: 10.1080/00207543.2019.1680899.
- [Bar+19b] P. C. Bartolomeu, E. Vieira, S. M. Hosseini, and J. Ferreira. "Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT." In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2019-September* (2019), pp. 1173–1180. issn: 19460759. doi: 10.1109/ETFA.2019.8869262.
- [CNC] CNC-Simulator. URL: <https://cnccsimulator.info/> (visited on 01/15/2021).
- [DIDa] DID-Core. URL: <https://www.w3.org/TR/did-core/> (visited on 01/15/2021).
- [DIDb] DID-Method-Registry. URL: <https://w3c-ccg.github.io/did-method-registry/> (visited on 01/15/2021).
- [Dri] Drizzle. URL: <https://www.trufflesuite.com/drizzle/> (visited on 01/15/2021).
- [ERC] ERC1056. URL: <https://github.com/ethereum/EIPs/issues/1056/> (visited on 01/15/2021).

- [ESN] ESN. URL: <https://docs.ens.domains/> (visited on 01/15/2021).
- [Eth] Ethereum. URL: <https://ethereum.org/en/> (visited on 11/21/2020).
- [Fed+20] G. Fedrecheski, J. M. Rabaey, L. C. Costa, P. C. Calcina Ccori, W. T. Pereira, and M. K. Zuffo. "Self-Sovereign Identity for IoT environments: A Perspective." In: *GIoTS 2020 - Global Internet of Things Summit, Proceedings* (2020). doi: 10.1109/GIoTTS49054.2020.9119664. arXiv: 2003.05106.
- [Fisa] Fischertechnik. URL: <https://www.fischertechnik.de/en/products/teaching/training-models/> (visited on 01/15/2021).
- [Fisb] Fischertechnik. URL: https://github.com/fischertechnik/txt_training_factory/ (visited on 01/15/2021).
- [FM18] T. M. Fernández-caramés and S. Member. "A Review on the Application of Blockchain for the Next Generation of Cybersecure Industry 4.0 Smart Factories." In: (2018). arXiv: arXiv:1902.09604v1.
- [Ful+12] A. Fuller, Z. Fan, C. Day, and C. Barlow. "Digital Twin : Enabling Technology , Challenges and Open Research." In: (2012).
- [Gar+19] C. Garrocho, C. Marcio Soares Ferreira, A. Junior, C. Frederico Cavalcanti, and R. R. Oliveira. "Industry 4.0: Smart Contract-based Industrial Internet of Things Process Management." In: (2019), pp. 137–142. doi: 10.5753/sbesc_estendido.2019.8649.
- [Ghaa] M. Ghanem. URL: https://github.com/ghanem-mhd/txt_training_factory/ (visited on 01/15/2021).
- [Ghab] M. Ghanem. URL: <https://github.com/ghanem-mhd/studious-enigma/> (visited on 01/15/2021).
- [Gri14] M. Grieves. "Digital Twin : Manufacturing Excellence through Virtual Factory Replication This paper introduces the concept of a A Whitepaper by Dr . Michael Grieves." In: *White Paper* March (2014).
- [Idea] D. Identity. URL: <https://github.com/decentralized-identity/ethr-did-resolver> (visited on 01/15/2021).
- [Ideb] D. Identity. URL: <https://github.com/decentralized-identity/did-jwt/> (visited on 01/15/2021).
- [Int19] T. Internet. *Blockchain Technology for Security*. 2019. ISBN: 9783319950372.
- [JSO] JSON-LD. URL: <https://www.w3.org/TR/json-ld/> (visited on 01/15/2021).
- [LBH18] Z. Li, A. V. Barenji, and G. Q. Huang. "Toward a blockchain cloud manufacturing system as a peer to peer distributed network platform." In: *Robotics and Computer-Integrated Manufacturing* 54.January (2018), pp. 133–144. ISSN: 07365845. DOI: 10.1016/j.rcim.2018.05.011.

-
- [Les+19] L. Lesavre, P. Varin, P. Mell, M. Davidson, and J. Shook. "A Taxonomic Approach to Understanding Emerging Blockchain Identity Management Systems." In: (2019). doi: 10.6028/NIST.CSWP.07092019-draft. arXiv: 1908.00929.
- [MA] N. Mohamed and J. Al-jaroodi. "Applying Blockchain in Industry 4.0 Applications." In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (), pp. 852–858. doi: 10.1109/CCWC.2019.8666558.
- [Mac] Machinekit. URL: <https://www.machinekit.io/> (visited on 01/15/2021).
- [Meta] Metamask. URL: <https://metamask.io/> (visited on 01/15/2021).
- [Metb] E. D. Method. URL: <https://github.com/decentralized-identity/ethr-did-resolver/blob/master/doc/did-method-spec.md/> (visited on 09/10/2020).
- [Mul] Multichain. URL: <https://www.multichain.com/> (visited on 01/15/2021).
- [Mus19] A. Mushtaq. "Implications of Blockchain in Industry 4.0 IMPLICATIONS OF BLOCKCHAIN IN INDUSTRY 4.0." In: June (2019). doi: 10.1109/CEET1.2019.8711819.
- [Noda] Node-RED. URL: <https://nodered.org/> (visited on 01/15/2021).
- [Nodb] Nodejs. URL: <https://nodejs.org/en/> (visited on 01/15/2021).
- [Opea] OpenZeppelin. URL: <https://github.com/OpenZeppelin/openzeppelin-test-environment/> (visited on 01/15/2021).
- [Opeb] Openzeppelin. URL: <https://openzeppelin.com/> (visited on 01/15/2021).
- [Qan] Qanache. URL: <https://github.com/trufflesuite/ganache/> (visited on 01/15/2021).
- [Qoq] Qoquorum. URL: <https://www.goquorum.com/> (visited on 01/15/2021).
- [Rea] Reactjs. URL: <https://reactjs.org/> (visited on 01/15/2021).
- [Roj17] A. Rojko. "Industry 4.0 Concept : Background and Overview." In: 11.5 (2017), pp. 77–90.
- [RSK19] A. Rasheed, O. San, and T. Kvamsdal. "Digital Twin: Values, Challenges and Enablers." In: (2019), pp. 1–31. arXiv: 1910.01719.
- [SHK17] J. J. Sikorski, J. Haughton, and M. Kraft. "Blockchain technology in the chemical industry: Machine-to-machine electricity market." In: *Applied Energy* 195 (2017), pp. 234–246. ISSN: 03062619. DOI: 10.1016/j.apenergy.2017.03.039.
- [Sol] Solidity. URL: <https://docs.soliditylang.org/en/v0.7.4/> (visited on 01/15/2021).
- [Tab] Tabler-React. URL: <https://github.com/tabler/tabler-react/> (visited on 01/15/2021).
-

- [Tru] Truffle. URL: <https://github.com/trufflesuite/truffle/> (visited on 01/15/2021).
- [TS17] L. Thames and D. Schaefer. *Industry 4.0 : An Overview of Key Benefits , Technologies , and Challenges*. April 2019. 2017. ISBN: 9783319506609. DOI: 10.1007/978-3-319-50660-9.
- [uPoa] uPort. URL: <https://www.uport.me/> (visited on 01/15/2021).
- [uPob] uPort. URL: <https://github.com/uport-project/ethr-did-registry> (visited on 01/15/2021).
- [VC-] VC-Data-Model. URL: <https://www.w3.org/TR/vc-data-model/> (visited on 01/15/2021).
- [VPB12] J. Venable, J. Pries-Heje, and R. Baskerville. "A comprehensive framework for evaluation in design science research." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7286 LNCS.2012 (2012), pp. 423–438. ISSN: 03029743. DOI: 10.1007/978-3-642-29863-9_31.
- [Web] Web3.js. URL: <https://github.com/ethereum/web3.js/> (visited on 01/15/2021).
- [XWS19] X. Xu, I. Weber, and M. Staples. *Architecture for Blockchain Applications*. 2019. ISBN: 9783030030346. DOI: 10.1007/978-3-030-03035-3.
- [XXL18] L. D. Xu, E. L. Xu, and L. Li. "Industry 4.0: State of the art and future trends." In: *International Journal of Production Research* 56.8 (2018), pp. 2941–2962. ISSN: 1366588X. DOI: 10.1080/00207543.2018.1444806.
- [ZB18] X. Zhu and Y. Badr. "Identity management systems for the internet of things: A survey towards blockchain solutions." In: *Sensors (Switzerland)* 18.12 (2018), pp. 1–18. ISSN: 14248220. DOI: 10.3390/sxx010005.
- [Zhe+17] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. "An Overview of Blockchain Technology : Architecture , Consensus , and Future Trends." In: June (2017). DOI: 10.1109/BigDataCongress.2017.85.