FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Blockchain-Based Management System for IIoT

Breno Accioly de Barros Pimentel

Master in Informatics and Computing Engineering

Supervisor: Rui Pinto

July 28, 2023

Blockchain-Based Management System for IIoT

Breno Accioly de Barros Pimentel

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

President: José Manuel de Magalhães Cruz
Referee: Rui Pedro Ferreira Pinto
Referee: João Paulo da Conceição Soares

July 28, 2023

Resumo

A 4ª Revolução Industrial aproveita os benefícios das tecnologias exponenciais para aprimorar os processos industriais, unindo a Tecnologia da Informação (TI) com a Tecnologia Operacional (TO). Uma das tecnologias mais notáveis da Indústria 4.0 é a Internet Industrial das Coisas (IIoT). A IIoT depende de dispositivos Edge interconectados que compartilham uma grande quantidade de dados usando arquiteturas convencionais de rede de Protocolo de Internet (IP). Além disso, o padrão IEC 61499 pode ser usado para projetar e implantar sistemas IIoT, garantindo a interoperabilidade dos seus componentes e comunicações.

Apesar das suas muitas vantagens, a IIoT também traz novos desafios a serem superados relacionados a arquiteturas descentralizadas. Uma arquitetura de servidor central, onde nós de *Fog* ou *Cloud* atuam como tomadores de decisão para todo o sistema, é suscetível a um único ponto de falha e carece de transparência para os usuários do sistema. Para superar esses cenários, as tecnologias Blockchain estão se tornando muito populares, pois permitem sistemas descentralizados e seguros por design por meio de algoritmos criptográficos de hashing.

No entanto, a fusão de Blockchain com a IIoT apresenta desafios adicionais. Uma rede blockchain tradicional requer muito armazenamento, pois um peer pode armazenar uma cópia completa do blockchain para fins de validação e consistência de dados distribuídos. Da mesma forma, requer uma potência computacional extensiva, necessária para executar protocolos de consenso bem estabelecidos e confiáveis, como o *Proof of Work*, o que limita a sua operação em dispositivos IIoT (conhecidos por serem restritos computacionalmente).

Este trabalho concentra-se no estudo de diferentes atributos, técnicas e arquiteturas para derivar um sistema de gerenciamento baseado em blockchain leve e adequado para a IIoT. Fornecendo uma exploração aprofundada de diferentes tecnologias, a pesquisa fornece insights valiosos sobre os benefícios e desvantagens associados ao seu uso, permitindo assim a tomada de decisões informadas em relação à adoção do blockchain em cenários reais de IIoT.

Além disso, uma arquitetura baseada em Blockchain foi implementada para lidar com os desafios específicos de gerenciar um sistema IIoT em conformidade com o padrão IEC 61499. Essa arquitetura elimina a necessidade de processar um volume significativo de tráfego num servidor centralizado, resultando em custos de manutenção reduzidos e fornecendo uma plataforma segura adequada para implantação em ambientes industriais. A utilização do padrão IEC 61499 permite ainda o tratamento da heterogeneidade e das características em tempo real inerentes aos sistemas IIoT.

Abstract

The 4th Industrial Revolution exploits the benefits of exponential technologies to enhance industrial processes by merging Information Technology (IT) with Operations Technology (OT). One of the most notorious Industry 4.0 disruption technology is the Industrial Internet of Things (IIoT). IIoT relies on interconnected Edge devices that share a sheer volume of data using conventional Internet Protocol (IP) network architectures. Moreover, the IEC 61499 standard can be used to design and deploy IIoT systems while assuring its components and communications interoperability.

Despite its many advantages, IIoT also brings new challenges to overcome related to decentralized architectures. A central server architecture, where Fog or Cloud nodes act as decision-makers for the entire system, is susceptible to a single point of failure and lacks transparency for the system users. To overcome such scenarios, Blockchain technologies are becoming very popular since they enable decentralized and secure-by-design systems through cryptographic hashing algorithms.

However, merging Blockchain with IIoT presents additional challenges. A traditional blockchain network requires too much storage, as a peer may store a full copy of the blockchain for validation and distributed data consistency purposes. Similarly, it requires extensive computational power, which is needed to perform well-established and reliable consensus protocols as the Proof of Work, which limits its operation on IIoT devices (known to be computationally restricted).

This work focuses on the study of different attributes, techniques, and architecture to derive a lightweight blockchain-based management system suitable for IIoT. Providing an in-depth exploration of different technologies, the research provides valuable insights into the benefits and drawbacks associated with their use, thereby enabling informed decision-making regarding the adoption of blockchain in real-world IIoT scenarios.

Furthermore, a Blockchain-based architecture was implemented to address the specific challenges of managing an IIoT system in compliance with the IEC 61499 standard. This architecture eliminates the need for processing a significant volume of traffic in a centralized server, resulting in reduced maintenance costs and providing a secure platform suitable for deployment in industrial settings. The utilization of the IEC 61499 standard further enables the handling of heterogeneity and real-time characteristics inherent in IIoT systems.

Acknowledgements

I would like to express my sincere gratitude and appreciation to those who supported me complete this work.

First and foremost, to *FEUP* for their support, resources, and facilities that supported me in my learning journey and the writing of this manuscript which I am deeply grateful for.

To my supervisor, *Rui Pinto*, whose insightful feedback, and continuous support have greatly contributed to the overall quality of this study.

To my *Family*, whose constant belief in my abilities and their unconditional love has been a constant source of motivation.

To my friends *Allan Sousa* and *Carolina Rosemback*, whose support and friendship were invaluable.

To *Pascal* and *Mobby*, my faithful companions that even from afar their memories brought comfort and joy. Love you little ones.

"Start by doing what is necessary,
then what is possible,
and suddenly you are doing the impossible."

St. Francis Of Assisi

Contents

List of Figures

List of Tables

Listings

Abbreviations

AI	Artificial Intelligence
AM	Authority Masternode
BGP	Byzantine General Problem
CA	Certificate Authority
CPPS	Cyber-Physical Production Systems
DAG	Directed Acyclic Graph
DHT	Distributed Hash Table
DPRG	Deterministic Pseudo-Random Generator
DPoS	Delegated Proof of Stake
DT	Digital Twin
ECC	Execution Control Chart
EVM	Ethereum Virtual Machine
FB	Function Block
IBFT	Istanbul Byzantine Fault Tolerance
IIoT	Industrial Internet of Things
IoT	Internet of Things
MSP	Membership Service Provider
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof of Authority
PoB	Proof of Burn
PoET	Proof of Elapsed Time
PoS	Proof of Stake
PoW	Proof of Work
PoX	Proof of X
SDK	Software Development Kit
TLS	Transport Layer Security
TPS	Transactions Per Second
WSN	Wireless Sensor Network

Chapter 1

Introduction

1.1 Context

The 4th Industrial Revolution, also known as Industry 4.0, is progressively achieving technological advances with exponential technologies such as Artificial Intelligence, Robotics, and Cloud Computing [4]. These technologies are characterized by their fast performance growth while also becoming more accessible. More than that, the fusion of these technologies can amplify even further their capabilities and enhance industrial processes [32].

One of the most important aspects of this new revolution is the extensive connectivity between the multiple company layers with the use of Cyber-Physical Production Systems (CPPS). This interconnectivity allows for fast and effective decision-making and better-optimized manufacturing processes.

In this context, interconnected Edge devices operating in conjunction and regularly updating their status or sharing information forms a network known as the Industrial Internet of Things (IIoT). This network allows the data from the machines to circulate and be analyzed. Therefore, this can lead to performance and cost-efficiency gains with the introduction of techniques like predictive maintenance, by performing the constant monitoring of wear in the physical materials [4].

Under those circumstances, blockchain technologies are widely regarded as a promising solution to enable decentralized and secure data integrity and availability. The extensive IIoT data issues can thus be benefited from traceability, the use of smart contracts to enhance the automation of the system, and robustness to resist failures.

Together with blockchain and continuously achieving notoriety on Internet of Things projects, the Digital Twins (DT), virtual representations of physical products, devices, processes, and systems, can have their audit processes more efficient and easier because of the immutability of the data and its history in the distributed ledger. Tracking can be also enhanced since the DT information and history can be transparently accessed [52].

1.2 Motivation

A blockchain architecture would fit the distributed characteristic of an IIoT system. A management system based on this technology would be benefited from attributes such as transparency, traceability, immutability, and reduced maintenance costs.

Cyber-Physical Systems perform a vast amount of networking processes, that increase the connectivity between multiple devices, thus extending the attack surface for novel cyber-security threats. Besides privacy and security concerns, it also requires extensive computational resources to handle all the operations in this network, posing scalability issues when managing it in a client-server architecture. However, blockchain technologies can offer a secure management platform for IIoT systems. Also, because the blocks and their internally stored transactions are immutable, and the nodes are only represented by their addresses, the transparency of the system can be better ensured without compromising its integrity [4].

Moreover, threats to industrial facilities and their operating systems exist, and can have financial, ideological, or even political motivations, and are the cause of major impacts. In one of the most remarkable cyber attacks on an Industrial environment, a malware later known as *Stuxnet* used multiple sophisticated techniques to infiltrate and then manipulate the control system of a Uranium Enrichment facility. Finally, it was able to destroy multiple centrifuges, which also demonstrated the possibility of a digital attack to create consequences in the physical world [24]. A Gartner report also indicates that damages to Cyber-Physical Systems could reach a financial impact of $50 billion by 2023, with costs related to compensation, litigation, and regulatory fines [14].

1.3 Problem

The adoption of IIoT introduces new challenges. For instance, Wireless Sensor Networks (WSN) is commonly a relevant component of IIoT. This network connects multiple power-constrained nodes that, conventionally, transmit their collected data to a single *sink*, which is a controller responsible for processing it or forwarding it to a gateway that may be connected to the internet [5]. However, this architecture is vulnerable to a single point of failure that could be exploited by denial of service attacks or triggered by system crashes, which can jeopardize the system's availability. Furthermore, the maintenance is also costly. A server's outbound data transfers are usually expensive, and handling multiple requests at once can compromise the data throughput.

On the other hand, a blockchain-based architecture can offer a decentralized and secure-by-design platform for IIoT devices to operate. The distributed characteristic of a ledger grants the data stored in it an elevated replication factor reasonable for production environments where availability is a critical concern.

Nonetheless, resource-constrained devices used in IIoT may not be able to operate demanding consensus protocols to achieve agreement between nodes in a blockchain network. Traditionally,

these protocols are not energy efficient and require high computing power. For instance, the Proof-Of-Work protocol, popularized by its application on Bitcoin, involves searching for a particular nonce, that when hashed in conjunction with the block, produces a number beginning with a specific amount of zeros. For each zero added to the start of the resulting hash, the time taken to generate it is exponential [33]. Although many more consensus mechanisms exist, not all are suitable for an IIoT system, and trade-offs between security and efficiency should also be considered [44].

Another problem is the heterogeneity of the system. In an industrial environment, the sensors could use different hardware and have distinct functionalities but should be able to communicate with themselves and also with other layers of the system. The IEC 61499 standard can be used to tackle the interoperability problem of the system. This standard specifies a generic model for distributed systems and device properties to achieve portability, reusability, and interoperability.

1.4 Goals

This work will focus on the study and development of a blockchain management system suitable for devices in an IIoT network. First, it will review the benefits and drawbacks of the use of a blockchain for an IIoT environment.

Considering the problems stated at 1.3, the proposed solution should be computationally inexpensive for the devices, and an implementation of the system should be developed so that it is possible to evaluate its performance. It should also analyze and consider the applicability of the IEC 61499 standardization with a distributed ledger and its benefits of it for manufacturing automation and system modularity. This standardization could also support the integration between the multiple layers of the system.

1.4.1 Research Questions

This work intends to answer the following research question:

RQ1. What is the impact of blockchain for the management of resource-limited devices in IIoT in terms of network latency and throughput, and computational resource consumption?

RQ2. How could the IEC 61499 standard be used with a blockchain architecture to manage IIoT devices?

RQ3. How does the use of blockchain for IIoT management compared to a solution without blockchain impact complexity and maintainability?

RQ1 address the applicability of blockchain technology with IIoT, considering the possible drawbacks and limitations and also its benefits.

RQ2 deals with the integration of this system in different industrial settings and use cases.

RQ3 measures the impact that a blockchain solution has on the quality and cost of maintenance of the code.

1.5 Methodology

1.5.1 Define Use Case

The first step in this research is to carefully determine a relevant and practical use case that aligns with the objectives of the proposed management system. The chosen use case will serve as the foundation for the entire implementation and analysis. In the context of this study, special consideration will be given to utilizing the management system in conjunction with IIoT devices. The implementation of the management system will also involve the utilization of blockchain technology to provide a decentralized and transparent framework for data storage, management, and analysis.

1.5.2 System Criteria

After defining the functionalities the management system should execute and how it will interact with other systems and users, the criteria used in the architecture must be analyzed. For example, [29] provides an assessment framework with the criteria when choosing a blockchain architecture.

The selected criteria will then support the requirements and architectural decisions. This will include the relevant characteristics of the blockchain, for instance, its level of permission, consensus algorithm, and which data should be stored in the ledger.

1.5.3 Test Environment

The implemented architecture will be deployed and tested on a real testbed. The network of this testbed can be composed of Raspberry Pi[1] devices, that should work as edge devices of an IIoT system.

Depending on the architectural choices, these devices may assume different roles, such as simple transaction issuers or holders of the blockchain. Furthermore, the network might be composed of other layers to support the devices, mainly in terms of storage. There might also be a front end connected to the back end layers so users or administrators could visualize and manage the blockchain-based system.

1.5.4 Performance Indicators

To evaluate the proposed and implemented architecture, the study will use a set of performance indicators as follows:

- **Latency**: The amount of time to process and broadcast a transaction to the other nodes. It can also be used with the confirmation time for a transaction to be successfully attached to a block. A blockchain architecture with less latency can be quicker and more responsive, thus this metric can be useful to test whether the architecture is suitable for real-time applications.

[1] https://www.raspberrypi.com/

- **Throughput**: The volume of transactions that can be handled per second. A more effective and scalable architecture has a higher throughput.
- **Computational Resources**: The system should consider CPU and memory usage. This is important as the devices running in the network possess very limited resources.

The work should also consider the size of the network and how it affects the performance indicators.

1.5.5 IEC 61499 Applicability

Besides the performance indicators, the system will also be evaluated regarding the application of the IEC 61499. [36] describes and uses a set of metrics to measure the complexity and maintainability of the system based on the standard. Some of them are presented below:

- **Structural Complexity**: Measures the coupling of an FB and the rest of the system.
- **Data Complexity**: Uses the data inputs and outputs and the related events to measure the data utilization.
- **System Complexity**: Combines the structural and data complexity in a single metric to reflect the level of modularity in the system.
- **Maintainability Index**: Measures the degree in which the FBs are open to changes. The metric considers the lines of code, operators and operands and the number of paths in the control flow graph.

1.6 Document Structure

This document is structured as follows:

In the following chapter, Chapter 2, an introduction to the Industry 4.0 is presented, followed by important concepts and technologies regarding the use of the Digital Twins, the IEC 61499, and blockchain.

Chapter 3, the state of the art, the related works, and a summary of the findings and similar problems studied are detailed.

Chapter 4, the use case on which the architecture will be based is discussed, and the design choices regarding the blockchain features and attributes and technologies to be used are analyzed.

Chapter 5, the implementation details are shown, including a high-level view of the multiple layers present in the system and its specifics.

Chapter 6, the methodology and tests are described, and its results are analyzed. Chapter 7, concludes the results gathered from the study and performed tests, as well as introducing possible ideas for future work.

Image 1.1, shows the flow of the content in this work.

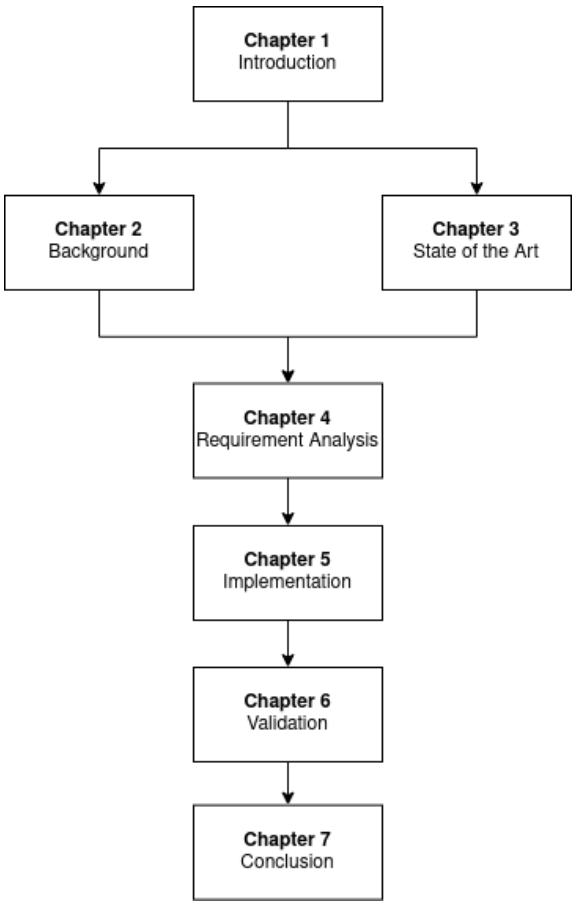


Figure 1.1: Document Structure.

Chapter 2

Background

In this chapter, the main concepts related to Industry 4.0 are presented, including its integration with blockchain technology.

2.1 Industry 4.0

The industry 4.0 is an evolution of automated industrial processes by including CPPSs and the use of exponential technologies, characterized by its exponential growth and progress. Figure 2.1 shows the industry evolution and its increase in complexity. The smart industry contributes to automated decision-making, which is possible through the high levels of connectivity between machines.

This evolution is mainly characterized by the vertical integration between the multiple levels inside the industry, enabling the real-time flow of data through the production lines to the supervision and control [4].

This new industry is possible with the adoption of key technologies to handle the fast pace of change. The industry uses additive manufacturing methods such as 3D printing, AI algorithms to predict and optimize processes and outcomes, and also blockchain and IIoT as mentioned in sections 2.1.1 and 2.2.

Blockchain and its features can provide an important platform to revolutionize Industry 4.0 by enhancing security, transparency, and trust in various processes and transactions.

While Industry 4.0 has the potential to revolutionize the way industries operate, there are some challenges that need to be addressed, including economic, regulatory, and technical aspects [4].

2.1.1 Industrial Internet of Things

The IoT refers to the interconnection between multiple technologies from a variety of domains that are integrated into a network where they can communicate with each other or enable users to interact with them. The technologies on this network can range from sensors and actuators to other everyday devices. The IIoT refers to this network when applied to an industrial setting [49, 40].

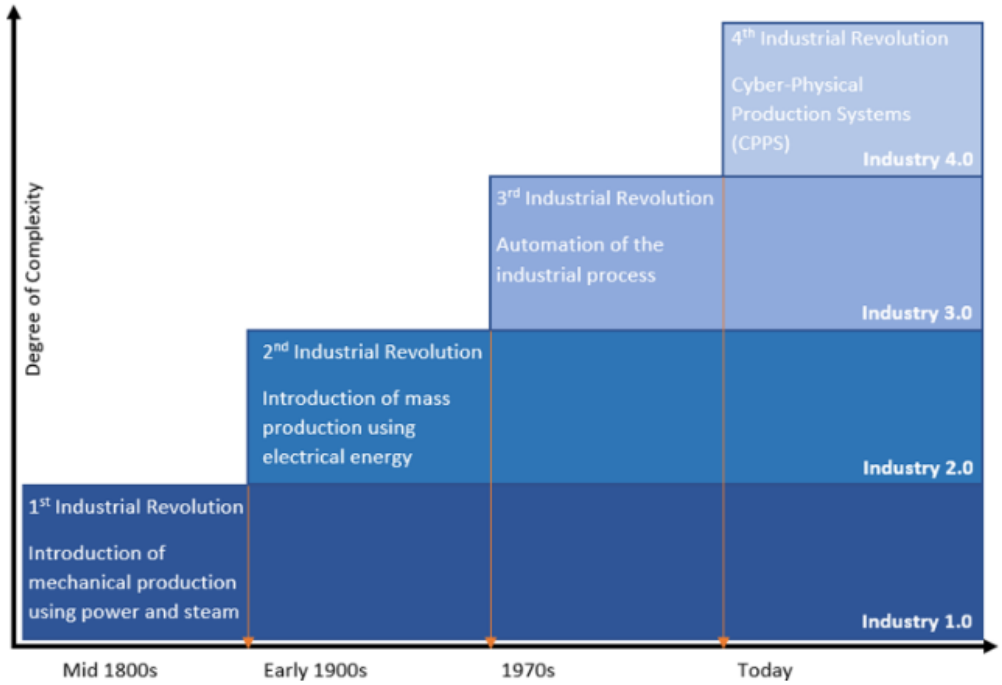


Figure 2.1: Industry evolution [4].

The IIoT has an important role in Industry 4.0, allowing the connection between industrial processes. It can be used to gather data from industrial systems and equipment to perform machine learning and big data analytics. Moreover, IIoT often uses cloud-based solutions to process and analyze all the information retrieved [41].

Multiple use cases apply to IIoT. The manufacturing sector is one of the most impacted by the technology. IIoT can change manufacturing by optimizing the supply chain and connecting machines to enable the smart factory. Another huge sector that is beneficiating from the technology is transportation, allowing routes to be optimized and infrastructure costs reduced. IIoT can also improve the processes in different and diverse areas, such as aerospace and agriculture [40].

Besides having disruptive capabilities, IIoT faces multiple challenges when being implemented [40, 41]. From the eventual problems that can arise when deploying this network, the ones that stand out are:

- **Security** and **data privacy**. Cyberattacks are able to exploit vulnerabilities in the system and leak sensitive information or compromise the system's availability and integrity.
- The **interoperability** of the system can also be a problem as multiple different devices may need to interact with each other, sometimes using different protocols, so special care must be taken to standardize the system.
- The **complex and voluminous data** flowing in the network creates the need for a more sophisticated infrastructure to handle and manage all the data.

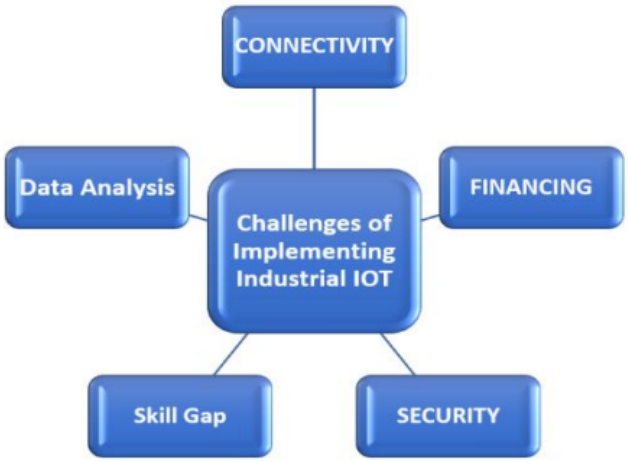


Figure 2.2: Challenges of IIoT [41].

- The **high costs** associated with the equipment, skilled workforce, and solving the aforementioned problems.

2.1.2 Digital Twins

Digital Twins (DTs) are virtual representations of a physical system and its processes or behavior. These digital models can fulfill a collection of use cases such as simulation, monitoring, and management of physical components, enabling optimized production strategies. A DT can be synchronized with the physical world, exchanging data and updating its internal state, or simply processing the information, obtaining results, but not returning this data to the real world. However, this absence of bidirectional communication in the latter case motivated some critics to disagree that these are real DTs, but rather digital shadows [26].

[13] describes three types of interactions between a digital and physical object, see Figure 2.3. First, in the digital model, the digital object has no direct exchanges of data with its physical counterpart. This model relies on manual data transfers and changes on either side do not directly impact the other. The digital shadows, as mentioned earlier, has a one-way communication where it receives the information from the physical object but does not perform any action on it. Finally, digital twins are defined as having automatic communication between the digital and physical worlds.

When building a digital twin, it is necessary to define its internal models and contents and to provide mechanisms for accessing and modifying them. These models may incorporate standards to enable modularization and data exchange. The digital twin must also feature APIs for gathering information and making it accessible to other services, and it must have mechanisms for identifying the physical device and establishing connections to it, as well as performing regular updates. Furthermore, a runtime environment is required for the execution of digital twins [28].

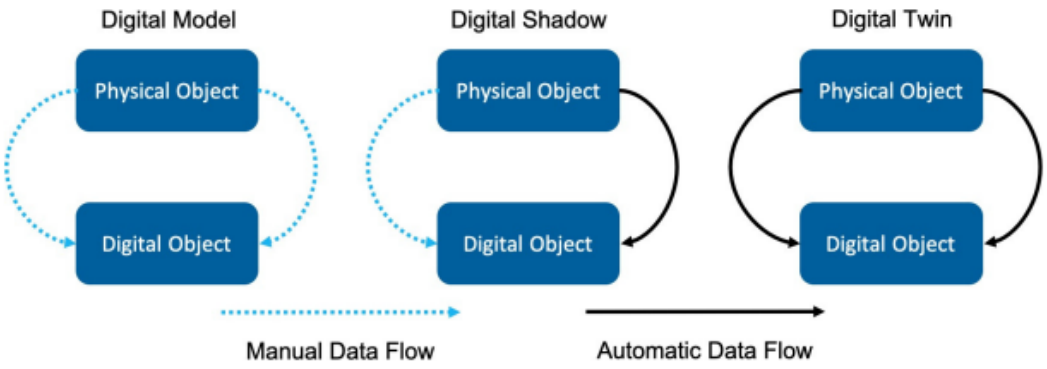


Figure 2.3: Digital model, shadow, and twin [13].

One of the potential uses of DTs is to provide valuable insights into industrial processes and systems by supporting data analysis and visualization. By creating a virtual replica of a physical asset or system, digital twins can collect real-time data about its behavior, which can then be analyzed to identify patterns and trends. These insights can be used to optimize processes, reduce costs, improve performance, and perform predictive maintenance. Additionally, digital twins can provide visual representations of complex data, such as 3D simulations, making it easier for operators and managers to interpret and analyze. By using data visualization techniques such as dashboards, charts, and graphs, digital twins can enable quick and intuitive data analysis, allowing for informed decision-making in real-time. Overall, digital twins can support data analysis and visualization in a way that enables industrial organizations to improve their operations and achieve their goals more effectively supporting the smart factory in Industry 4.0 [13, 28].

More IoT projects are using DTs, and blockchain can help potentialize this technology. The use of blockchain can benefit DTs by providing integrity and trust without the need for a central authority, ensuring the legitimacy of documents, and enabling easy access and tracking [52].

2.1.3 IEC 61499

The IEC 61499 standard specifies a generic model for distributed systems and device properties to achieve portability, reusability, and interoperability. The main component of the architecture is the function block (FB). This unit has clearly defined inputs and outputs and can have multiple variables that are protected. Thus encapsulating the functionality of each block as no global variables are allowed [48, 53].

Figure 2.4 presents the interface of an FB. It is composed of two segments, an Execution Control Chart (ECC), which receives event inputs, and a data interface that handles the encapsulated variables and functionalities. An event on the left side triggers the associated functionality and updates the internal variables with the data input. Then, it outputs an event and related data on the right side. Multiple FBs can be interconnected through their inputs and outputs allowing the creation of event-driven pipelines.

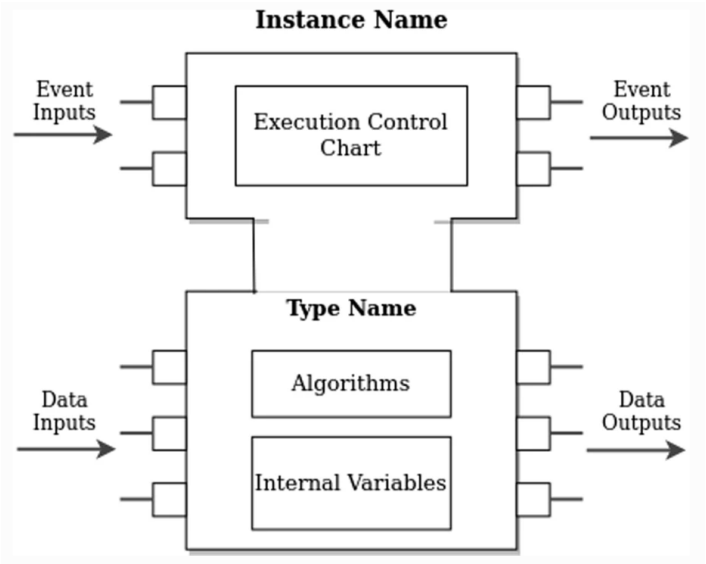


Figure 2.4: Generic Interface of a Function Block [36].

2.1.3.1 4DIAC

[10] offers a set of open-source applications for the design and development of industrial control systems following the IEC 61499 standard. One of the available tools is the development and testing environment 4DIAC-IDE, which allows the specification of an FBs network and its hardware interconnections. Once the system pipeline is complete, it can be mapped and executed in a device through an open port.

2.1.3.2 DINASORE

DINASORE by [35] is a framework for creating distributed industrial applications, aiming at the reconfiguration and execution of DTs in CPPS. This platform is compliant with the 4DIAC, which enables DINASORE to receive commands from the 4DIAC-IDE to create and reconfigure the FBs pipeline of the IEC 61499 industrial standard using a graphical user interface. Figure 2.5 shows the high-level architecture of the DINASORE system, its interfaces, the FB pipeline, and the resources used.

DINASORE FBs are composed of an XML file holding metadata about its structure and a Python file for the code functionality. The Python language was used with the idea of allowing the use of advanced machine-learning libraries at the edge of the system.

The platform uses Open Platform Communications - Unified Architecture (OPC UA) protocol as a communication interface with other layers of the system. The use of the OPC UA also allows the monitoring of FB variables in real time.

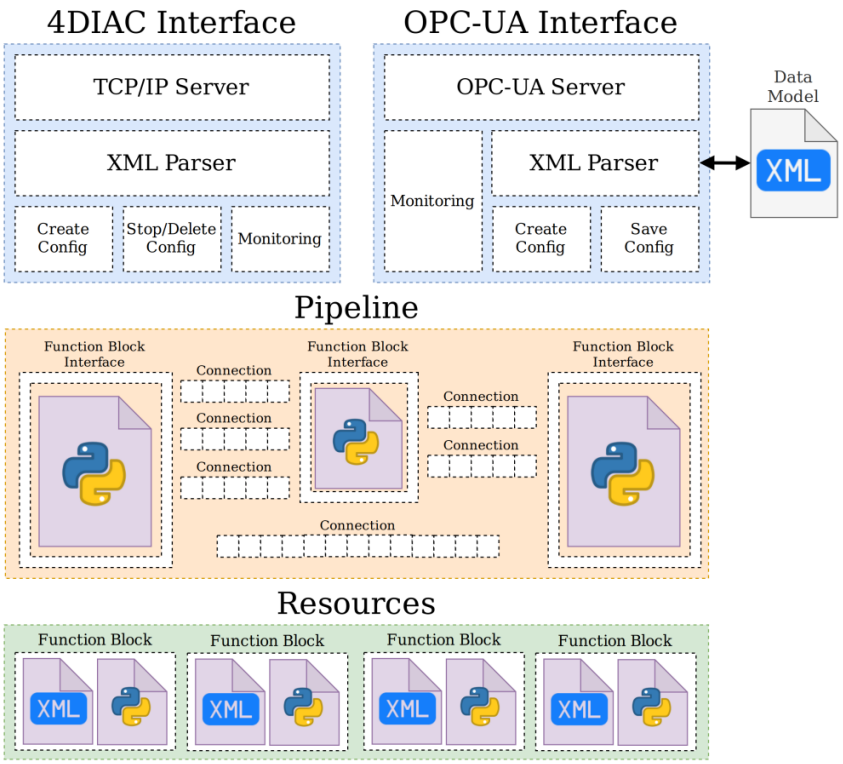


Figure 2.5: DINASORE architecture by [35].

2.2 Blockchain

A blockchain is a distributed ledger technology used to replicate and synchronize data among multiple nodes. The first application of this technology was described in [33] with the introduction of the Bitcoin cryptocurrency, a trustless, decentralized payment system that eliminates the need for a central authority.

The blockchain is formed by blocks linked by the hash value of the previous block in the chain, except for the first one, the *genesis* block. This hash value is stored in the block header, which contains other metadata such as the root of the *Merkle* tree. Each leaf in the tree is a transaction, whereas its parent value is its computed hash. The nodes in the next depth are then joined in pairs where their grouped hash is the parent value for these nodes, see Fig. 2.6. This logic continues until the depth zero, the Merkle root. Once a transaction is confirmed and added to a block, any changes to it would also affect the Merkle root and, consequently, the block hash, ensuring that it is now immutably stored.

When issuing a transaction, it is the responsibility of validator nodes to validate it before including it in the next block of the chain. However, the block generation must follow a consensus algorithm so all the nodes in the network agree on which node should attach the newly created block.

There are many consensus algorithms available, with different tradeoffs, [44] scrutinizes each consensus category with its applicability to IoT. One of the most notorious is the Proof of Work

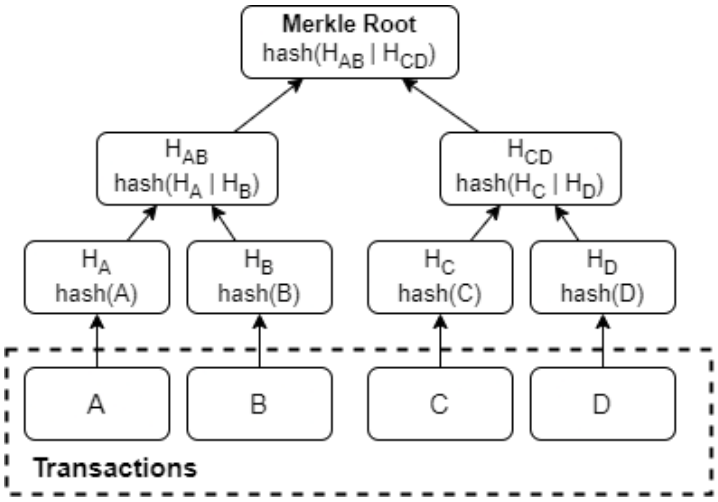


Figure 2.6: Merkle Tree.

protocol. This protocol is based on the assurance that a prover has executed a sufficient amount of computational effort within a certain time [22]. The computational load needed to operate this protocol and the high block processing time hinders its use in many applications including the IoT, where other algorithms can be more suitable. For instance, the Proof of Authority, used in permissioned blockchains, see 2.2.1, defines a set of trusted nodes acting as validators. This approach increases efficiency and scalability as only a limited number of nodes participate in the consensus, with the drawback of being less decentralized [44].

On the second generation of blockchains, the concept of *smart contracts* was included in the technology, allowing the technology to take new paths apart from crypto coins. The smart contracts are decentralized programs stored in the blockchain. These programs are executed once the preconditions for the contract are met. Its application to IoT enables automated peer-to-peer contractual procedures for device management with lower costs [34].

The smart contract results can be validated by every running peer in the network which is important when ensuring policies within supply-chain management systems and business rules. This feature can thus leverage the trust between multiple parties in an automated and secure way [2].

Finally, the third generation of blockchains aims on fixing the problems that hind its use in other industry use cases. It focuses on enhancing aspects such as scalability, interoperability, and sustainability [7]. To achieve these goals, it can use other data structure approaches, such as the Directed Acyclic Graph in the IOTA network [37].

2.2.1 Blockchain Access Type

The access criteria of a blockchain relate to the rules of who can perform the different operations in the network, such as who can read and write to the ledger or participate in the consensus protocol.

Blockchain technology is categorized into two types, permissioned and permissionless. Both permissioned and permissionless networks have their advantages and disadvantages and are suitable for different types of applications.

2.2.1.1 Permissionless

Permissionless blockchains are truly decentralized networks that allow anyone to access and participate in the protocol as they do not rely on a central authority to define roles and policies. Any user in this configuration can issue a transaction or publish a block to the ledger. Thus, the security is guaranteed by the consensus protocol, which usually rewards the peer for adding a new block with the network's cryptocurrency [51]. This type of blockchain offers full transparency for users to verify the transactions, allowing a secure, trustless, and censorship-resistant platform.

However, permissionless blockchains also have some drawbacks. The energy consumption required to operate the consensus protocols and maintain the network's security can be very high. Moreover, not all information can be safely shared in the ledger, even if the blockchain provides anonymity, the transparency and consequent lack of privacy can hinder its utilization in some use cases [30]. For instance, private keys, personally identifiable information, intellectual property, or any other sensitive information may not be suitable to be stored in this type of technology.

A permissionless network can also have more trouble when scaling as the number of participants and transactions increases. Maintaining high performance and throughput becomes increasingly difficult. For instance, Bitcoin uses the Proof of Work consensus protocol, which demands significant computational resources, with a limited block size. These attributes are due to security reasons, however, the number of transactions processed per second can be negatively impacted slowing down the network [51, 30].

2.2.1.2 Permissioned

Unlike permissionless networks, permissioned networks require some level of trust between the users. In this setting, users should first be authorized by an authority before participating in the consensus algorithm. Consequently, the organization that is maintaining the blockchain is responsible for establishing the identity of a participant to provide access. This potential can be shared by multiple organizations so that it can be also decentralized between multiple parties without fully trusting one another, employing consensus mechanisms tailored to the specific requirements of the collaborating entities. Furthermore, in the case of a peer not following the rules in the network, the responsible organization should have the means to revoke the access of this peer when needed [51].

These networks are usually highly efficient and faster when compared to permissionless ones. With controlled access and consensus mechanisms designed for the participating entities, permissioned networks can achieve higher transaction throughput and quicker validation times, enabling faster business processes and decision-making. Moreover, permissioned networks provide a certain level of privacy in transaction details. While public blockchains offer complete transparency,

permissioned networks allow organizations to have selective visibility of transaction data. Confidentiality features, such as encrypted transactions or restricted data sharing, enable participating entities to maintain the privacy of sensitive business information, ensuring confidentiality within the collaborative environment [51].

Despite these advantages, these networks have certain drawbacks when compared to permissionless networks. The more obvious is the higher centralization level of this network type, which limits the participation and transparency of entities outside the blockchain. Another problem is managing the access of users to the network, in this scenario the network should be able to verify the identity of the users, so managing and storing cryptographic keys is crucial for the correct operation of the blockchain. This task can become complex, as there might be different levels of privileges inside the network and the configuration of a public key infrastructure would be necessary.

2.2.2 Technologies used in Blockchains

2.2.2.1 Hyperledger Project

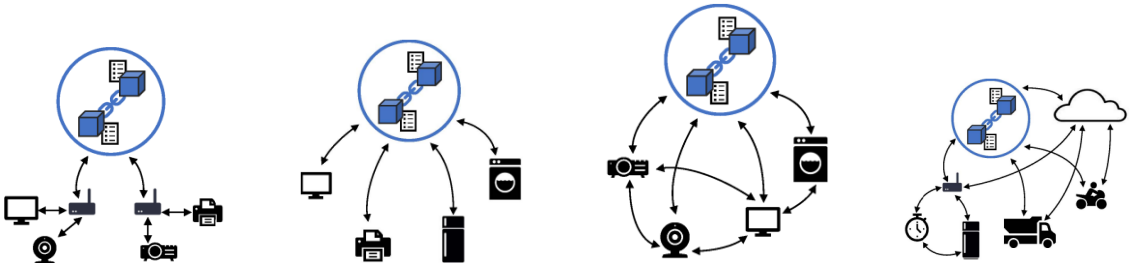
The [18] Foundation, hosted by the Linux Foundation, provides open-source frameworks, tools, and libraries focused on consortium networks for enterprise-grade blockchain deployments. The technologies offered by the project includes distributed ledger frameworks. The most relevant ledger frameworks are presented as follows:

- **Fabric**: The Hyperledger Fabric offers a modular and versatile framework. It allows pluggable consensus protocols, features such as the use of *channels* to keep the data private and accessible to part of the members, and governance of smart contracts. This architecture allows it to meet the requirements and adapt to a wide range of use cases.
- **Besu**: An Ethereum client, implementing the Ethereum protocol, that runs on the public Ethereum network or permissioned networks. It provides the Proof of Work protocol, and also the Proof of Authority with the IBFT 2.0 and Clique implementations.
- **Sawtooth**: Offers high modularity and implementations for the PBFT, Proof of Elapsed Time, and Raft consensus algorithms.

Alongside these technologies, Hyperledger also provides tools such as *Hyperledger Caliper* to measure the performance of the blockchains.

2.2.2.2 IOTA

[20] is a distributed ledger built with a focus on the IoT Industry. This system uses a directed acyclic graph, named *Tangle*, to store its transactions. The Tangle enables the parallelization of the consensus protocol and could allegedly offer a highly performant consensus solution. In the current state of the technology, it is not fully decentralized [37].



(a) Gateway devices as endpoints.

(b) Devices as transaction-issuers.

(c) Edge devices as end-points.

(d) Cloud-blockchain hybrid.

Figure 2.7: Blockchain and IoT integration paradigms from [3].

2.2.2.3 IoTeX

[21] is a blockchain platform to support large-scale, decentralized IoT systems, ensuring the authenticity of the data produced by devices on this system. It uses a randomized variation of the Delegated Proof of Stake (DPoS) blockchain consensus protocol, Roll-DPoS [12]. Roll-DPoS uses the PBFT consensus algorithm to dynamically choose a block producer group and thus achieve higher decentralization when compared with the base DPoS.

2.2.3 Blockchain in Industry 4.0

[3] defines some main paradigms when integrating edge devices networks with blockchain as follows:

- 2.7a The devices acting as a gateway, that are responsible for collecting, processing, and forwarding data, are assigned as endpoints to the blockchain. In this scenario, these devices handle the transactions of several IoT devices on their behalf.
- 2.7b The IoT devices issue transactions directly to the blockchain. This ensures that all events are immutably stored in the blockchain.
- 2.7c The edge devices, both usual IoT and gateways, issue transactions to the blockchain. However, they are also able to communicate between themselves outside of the blockchain. This could increase the throughput as data would not need to go through the blockchain first, but also not all data will be logged into the blockchain itself.
- 2.7d The edge devices can communicate with each other and issue transactions to the blockchain, but also with the possibility to use fog or cloud computing to overcome the limitations of an IoT network, by removing the computational load from the devices.

Chapter 3

State Of The Art

In this Chapter, works related to the use of blockchain, digital twins, and the application of the IEC 61499 are discussed in section 3.1. The conclusions and research gap are identified in section 3.2.

Some of the work presented is related to the Internet of Things (IoT), but because of the similarities, the solutions could also be a base for implementations in an IIoT setting.

3.1 Related Work

This section presents existing approaches to integrate blockchain technology with IIoT technology and related use cases.

3.1.1 Agri-food chain certification

[30] proposed and implemented a system to track the provenance and events of a food supply chain. They used a combination of conceptual frameworks to create an evaluation framework and perform the design choices that would better suit their architecture specifications, such as industrial applications, including supply chain management. They decided the most relevant criteria from the framework proposed by [43] and aggregated it with others.

Following this, they opted for Ethereum permissioned blockchain using the Proof-of-Authority consensus mechanism. The architecture is composed of validator nodes, holders of a copy of the blockchain, that validate transactions and generate new blocks using the consensus algorithm. Participants nodes, which also store a copy of the blockchain but do not participate in the consensus protocol. Operators, that send transactions to change the blockchain state. And also, external users who can access the system nodes in read-only mode depending on the authorization configuration.

The choice of a permissioned blockchain in an industrial setting was based on the argument that public blockchains lack performance and scalability, while also not supporting data privacy, therefore the system might not be compliant with privacy laws such as the European GDPR. However, the solution includes the use of a public blockchain to preserve the immutability of the

blockchain. For instance, the hash of the last validated block is periodically written to the main public Ethereum blockchain. Furthermore, it also uses an off-chain database to store large files.

Finally, a prototype of this architecture was implemented in a real use case for product traceability. The external users are the buyers of food and can access through a QR code the decentralized application and visualize all the relevant events of this product in the supply chain. This system could also allow agricultural professionals to certify the products.

3.1.2 Blockchain-based distributed control system

[45] presents a blockchain-based control system compliant with the IEC 61499 standard. The work focuses on applications that require increased responsiveness and processing of a large quantity of data at the edge of the network.

The system uses a three-layer network model. The blockchain operates on the top layer, securing and validating the transactions. It uses Hyperledger Fabric[1], a permissioned blockchain platform, see 2.2.2.1. This platform allows smart contracts that were implemented as Function Blocks and Service Interface Function Blocks (SIFB) interface the external environment with the function block application, enabling the smart contracts to consume outside data or transfer it to other external services. This standardization supports the application to be used in any control system.

3.1.3 Management of IoT devices using Blockchain

[23] describes a system for managing and monitoring IoT devices with a private blockchain. The architecture allows a device's configuration to be stored in the blockchain and to be retrieved when requested.

It uses the Hyperledger Composer as the Blockchain implementation. The configuration files can be stored and updated using transactions holding the encrypted device configuration along with its identification performed by the network administrators. These administrators authenticate via digital certificates managed by a blockchain-based PKI. When a transaction is successful, the devices are alerted and responsible to check whether the configuration affects its system and then loading it. Because of the possibility of large configuration files, the solution also proposes an off-chain database similar to [30]. However, it does not recommend the use of this solution as the data stored in this database would not be protected by the blockchain properties.

3.1.4 Blockchain-Based system for WSN-enabled IoTs

[17] proposed a blockchain-based system for WSN-enabled IoT with the support of cloud storage. The network sensors must first be registered by a base station, as only authenticated sensor nodes can send their sensed information. The base stations run the blockchain nodes and are responsible to collect this data and work on the block generation. Due to a large amount of data, only part of

[1] https://www.hyperledger.org/use/fabric

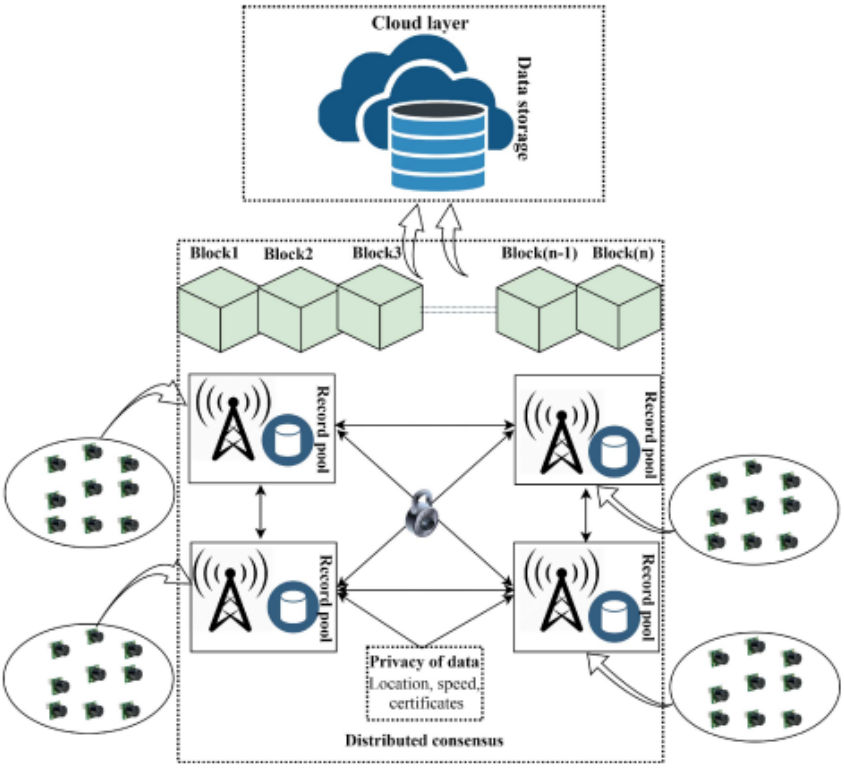


Figure 3.1: High level architecture proposed by [17].

it is saved in the ledger. The blockchain stores key parameters used in the certification of sensor nodes and their event information, such as timestamps and location. A cloud layer connected to the base stations stores the rest of the data as shown in Fig. 3.1.

3.1.5 EtherTwin: Blockchain-based Secure Digital Twin Information Management

[38] researched the suitability of a blockchain-based DT management system and implemented a prototype of the architecture that can be used in permissioned and permissionless networks. The architecture provides an user interface, where users can access the shared DTs with confidentiality, as the data is encrypted and access control is based on the user's role.

A bi-directional communication is established between the real-world asset and its associated DT, Fig. 3.2. The DT full data is stored off-chain using the decentralized Swarm[2] storage service. It uses a *Kademlia* based distributed hash table (DHT) that interacts through smart contracts with the Ethereum network [46]. The blockchain holds the DT metadata that is linked to its off-chain counterpart.

3.1.6 Smart Grid Controller Blockchain Platform

[15] describes the implementation of a platform for virtual power plant communication based on blockchain. The virtual power plant aggregates multiple distributed energy resources in a single

[2]https://www.ethswarm.org/

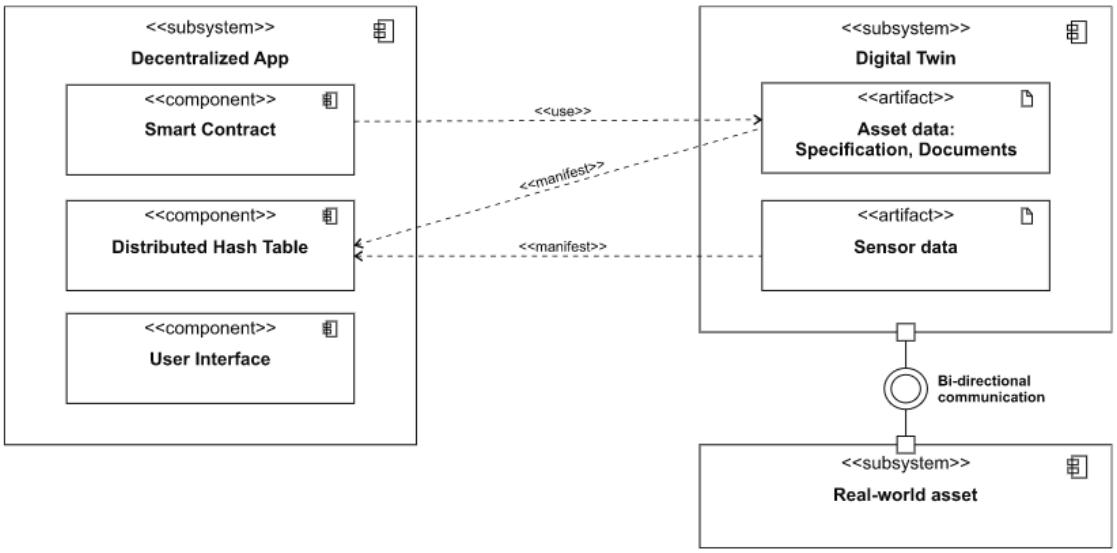


Figure 3.2: Decentralized DT sharing application subcomponents by [38].

system. This work used the Hyperledger Fabric to implement the blockchain and uses smart contracts to handle the virtual power plant operations.

3.1.7 Blockchain-based trust mechanism for digital twin empowered Industrial Internet of Things

[1] developed a smart architecture for use in DT virtual maintenance. This architecture uses a trust mechanism based on the PoA algorithm and proposes a deterministic pseudo-random generation-based (DPRG). In the model, an Authority Masternode (AM), a node that disclosed its identity and was authorized by the base station, is responsible for adding a new block, and DPRG to generate a genesis block. The resulting architecture presented reduced network delay and power consumption when compared to a PoW approach.

3.2 Conclusions

3.2.1 Blockchain Data Management

When storing data in a blockchain, there are two main pattern types. The data can be kept off-chain or on-chain [50].

Depending on the use case, the transparency offered by blockchains can be an issue such as critical data that should only be accessed by authorized users. To keep the privacy of confidential data stored on-chain, a possible solution would be to create a secret key and share it with allowed users using key management protocols. This would preserve the confidentiality of the data, however, the data is immutably stored in the chain and could be vulnerable to key compromises and brute-force attacks in the future when the current cryptography might not be enough for advanced cryptanalysis techniques associated with more computing power.

The data stored on the blockchain can also be too large, and not suitable for limited storage capacity devices. Moreover, the transactions can also have a maximum size limit. This issue could be handled by storing the data off-chain, in an external database. In this scenario, the blockchain stores the hash value of the real data and a link to it to guarantee the integrity of the files stored off-chain. However, the data could still be inappropriately modified in the external database and the hashes in the blockchain would only reveal the integrity fault, but would not have the capability of restoring the original data alone. [23] states that this solution would break the principles of blockchains. [30] mentions that the database could also be decentralized, for instance, following the IPFS protocol.

3.2.2 Blockchain Integration With IIoT

Many authors discuss the problems of integrating blockchain technology with both IoT and IIoT networks. In the proposed and implemented solutions analyzed, the restricted computing power of devices in these networks causes them to rely on cloud storage and computing and are mostly limited to performing transactions while not contributing to the consensus mechanism.

Additionally, they usually use private blockchain architectures, that impose restrictions on the access and participation of the consensus protocol. These commonly handle more transactions per second, using more energy-efficient algorithms and also possibly providing some level of privacy that may be a business need. Some work also proposes to periodically register the hash of the last generated block of the private blockchain in a public chain. [30] uses this approach to anchor a permissioned blockchain with the Ethereum main network, to improve the immutability feature of their architecture.

Finally, there is also a lack of work describing in detail the use of the IEC 61499 standard to handle the heterogeneous attribute of IIoT networks with underlying blockchain architecture. The applicability of the standard is not well defined, but there are examples of its use to communicate the blockchain with other layers of the system and with smart contracts to be used as a control mechanism [45].

Chapter 4

Requirement Analysis

In this chapter, the use case and the requirements of the system are detailed. Moreover, the details of the technologies and the characteristics chosen are clarified.

4.1 Use Case

This work will showcase the benefits of using blockchain technology to store and manage sensor-generated events from a smart factory. In this scenario, a network of diverse sensors is deployed to collect real-time data on various machinery parameters such as temperature, humidity, and noise levels. These sensors continuously emit events representing the measurements and observations they capture.

Each sensor event contains specific data points and a timestamp, which is recorded as a transaction on the blockchain. The use of blockchain technology in this monitoring use case ensures data immutability, transparency, and integrity while empowering stakeholders with real-time data access and automated actions, see Figure 4.1.

Furthermore, considerations will be given to the scalability and performance aspects of the management system, as IIoT environments often involve a large number of devices generating a significant amount of data.

4.2 Requirements

A management system should fulfill several key requirements to effectively support organizational operations and decision-making. This section describes some of the requirements needed for the use case used in this work and that the architecture will be based on.

Firstly, the system should handle data management, including operations for creating, reading, updating, and deleting data. The implemented system should also ensure the confidentiality and integrity of the collected and stored data.

Another important point is the possibility of the data collection process being automated. This will support the real-time monitoring of the information flow. Consequently, data analytics tools

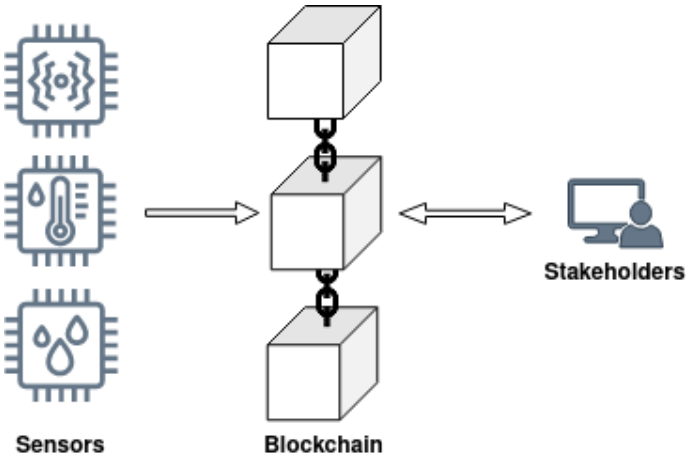


Figure 4.1: Use case model.

should be able to access this data flow and generate reports and perform data analysis to facilitate informed decision-making. As well as other data visualization applications or dashboards to enable better human interpretation and monitoring.

Additionally, reaching a consensus is essential for preserving system integrity in IIoT environments where a variety of devices and systems must function reliably and cohesively. Security safeguards should also be adopted to restrict access to consensus participation and protect the data from unauthorized users, such as user authentication through public-key cryptography and role-based access control.

4.3 Blockchain Characteristics

When building a blockchain, multiple aspects must be considered to adjust it to a particular use case. In this section, the characteristics of the blockchain are compared and analyzed so that it matches the requirements defined in 4.2.

4.3.1 Access Criteria Choice

As previously mentioned in Chapter 2.2.1, a permissioned network offers controlled access, ensuring only trusted entities can join the network and actively participate in the consensus process. This level of control and governance is crucial in an IIoT environment where sensitive and critical operations are involved. Moreover, it can offer higher transaction throughput and lower latency. In this work, the architecture will thus follow this configuration, as it will supposedly fit better to the defined requirements.

4.3.2 Consensus Mechanisms

Ensuring trust, security, and consensus among IIoT devices is crucial for the reliable operation of the system, and through consensus protocols, the IIoT ecosystem can achieve decentralized,

secure, and efficient operation, enabling a wide range of applications and services that rely on trustworthy interactions among IIoT devices. There are multiple consensus protocols available that are suitable for different purposes. However, to enable its use on IIoT devices, they must address challenges such as limited resources, network constraints, and low-latency communication.

In this section, the characteristics and benefits of these protocols are analyzed in the context of an IIoT network. This analysis is mainly based on the works presented by [44] and [42].

4.3.2.1 Proof-of-X Consensus

The Proof-of-X (PoX) mechanism consists in letting participants demonstrate their reliability by performing some kind of proof. The proof criteria are related to the resource, influence, or commitment allocation in the network.

There are multiple PoX protocols available. The following list describes some of the most relevant or that would potentially fit in an IIoT setting.

- **Proof of Work**: The PoW protocol relies on participants called miners to perform a computationally expensive task before being allowed to introduce a new block in the network. This makes it economically and computationally expensive for any individual or group to manipulate the blockchain's history. The consensus is achieved by the majority of miners agreeing on the validity of the longest chain. However, the mining dependency of the protocol requires a considerable energy consumption that is not suitable for IIoT.

- **Proof of Authority**: PoA works in a permissioned network where the participants must first be approved and trusted to join the consensus. This system uses the concept of reputation to operate, where misconducting participants could have their access revoked to maintain the network's integrity. This protocol offers faster transaction processing times, as the validators are known, and no intensive computational work is needed. The main drawback of this mechanism is the level of centralization created by authorities governing access to the network.

- **Proof of Stake**: In PoS, validators to introduce new blocks are chosen randomly but the probability of one being selected is based on its stake, which is the amount of cryptocurrency the participant holds. Compared to PoW, this approach is more energy efficient and allows faster block creation, with the drawback of favoring those with higher stakes.

- **Proof of Burn**: PoB uses a unique concept where network users burn their coins to demonstrate their commitment, thus allowing them to participate in the consensus. The burning process works by sending the coins to a burn address, where they cannot be redeemed. The more tokens a participant burns, the higher their chances of being selected to mine the next block or validate transactions. The positive points of this method are similar to the ones of PoS but with the drawback of the participants not being able to retrieve their coins once it decides to leave the network.

- **Proof of Elapsed Time**: The PoET mechanism works in a trusted execution environment, where a random leader election is performed. The elected leader is responsible for creating and adding a new block to the chain. The leader then waits for a randomly generated period of time from a trusted function, and the first validator node is selected as the next leader. Because of the uniform distribution of waiting times, this process gives other participants a fair chance of being selected to add a new block. The protocol is more energy efficient than other approaches, such as the PoW, as it does not require mining operations.

4.3.2.2 Paxos-based Consensus

A Paxos-based consensus protocol consists of three phases. In the Prepare phase, a proposer sends a proposal number to the acceptors, requesting their promise not to accept any proposals with a number less or equal to the proposal number used in the previous round. In the Accept phase, if the proposer receives promises from a majority of acceptors, it sends an accept message with a proposed value. Acceptors check if they have made promises and, if so, accept the proposal. In the Learn phase, when the learner nodes receive accepted messages from a majority of acceptors, they learn the accepted value, which becomes the agreed-upon value.

The Paxos protocol family has many variations such as Multi-Paxos, Fast-Paxos, and Byzantine-Paxos, each with its specific optimizations and considerations. These protocols find applications in various distributed systems, including Apache Cassandra, a widely used distributed NoSQL database system, that uses Paxos to ensure data consistency and fault tolerance.

Although successful in other use cases, on a low-power network with devices with limited resources, Paxos is not suitable as it is complex and requires significant communication overhead.

4.3.2.3 Raft consensus

The Raft consensus is designed to manage replicated logs and achieve agreement in a distributed system. It provides a straightforward and understandable approach to consensus compared to more complex algorithms like Paxos. The Raft algorithm emphasizes robust consistency, fault tolerance, and simplicity. It assigns a leader to each term, divides time into terms, and makes sure that all nodes replicate and maintain consistency in the logs. Raft has grown in popularity because of its simple design and systems such as the CockroachDB[1] distributed database applies the mechanism.

The algorithm process starts with a leadership election, where nodes exchange messages to determine the most up-to-date node as the leader for the current term. The leader is responsible for coordinating client requests and appending them to its log. It then replicates the log entries to other nodes, ensuring that a majority of nodes confirm the replication before committing the entries. In the meantime, the nodes periodically receive heartbeats, messages to indicate the normal operation of a node, from the leader. This approach guarantees strong consistency across the system and prevents inconsistencies that may arise from concurrent operations.

[1] https://www.cockroachlabs.com/

Raft's simplicity and fault tolerance make it an attractive choice for IIoT applications in permissioned networks. Its leader election process, log replication mechanism, and ability to handle leader failures align well with the requirements of IIoT systems.

4.3.2.4 Byzantine Consensus

This Byzantine family of protocols is based on the Byzantine General Problem (BGP). In this problem, a group of Byzantine generals surrounds an enemy city and must collectively decide whether to attack or retreat. The challenge lies in the presence of traitorous generals who may send conflicting messages to deceive the loyal ones, where the goal is to reach a consensus among the loyal generals despite the presence of Byzantine faults.

The main variations of these protocols are the Practical Byzantine Fault Tolerance (PBFT), which works by using multiple rounds of voting among known and trusted nodes to achieve consensus on a proposed block, and the Istanbul Byzantine Fault Tolerance (IBFT). Both ensure that the network is reliable up to m nodes with byzantine behavior, whereas, in a system with $3m + 1$ nodes, there is no solution.

This consensus family does not require high computing resources and is simpler to implement, for instance, when compared to Paxos, which makes it a good option for an IIoT network.

4.3.2.5 DAG-based Consensus

This consensus family relies on the Directed Acyclic Graph (DAG) structure in an attempt to solve the common problems in conventional blockchains, scalability, and throughput. Consequently, distributed ledgers with this technology are not exactly considered blockchains although their decentralization and security goals are similar. In DAG each transaction references one or more previous transactions, forming a graph structure, instead of the linear chain approach of a traditional blockchain.

DAG technology is still in a very recent stage. With IOTA being one of the prominent pioneers in this field, it has gained attention for its implementation of the Tangle, a DAG-based ledger architecture. The Tangle aims to provide a scalable and feeless transaction system, particularly suitable for IoT applications [37].

4.3.2.6 Consensus Applicability to IIoT

As described in the previous sections, not all consensus fit in every application. Figure 4.2, shows a comparison between protocols, some of them described in this section, in relation to their applicability to IoT. White circles denote a non-suitable protocol, black-white circles are partially suitable, and black circles are suitable for employment in IoT. From this table, only PoET, PBFT, and Tangle, are considered genuinely suitable for IoT. Others such as PoS, DPoS, and Raft are only considered partially suitable.

Consensus method	Access[1]	Decentral-ization	Scalability	Latency[2]	Through-put[3]	Adversary tolerance	Computing overhead	Network overhead	Storage overhead	IoT suitability
PoW	Public, PL.	High	High	High	Low	<25% Computing Power	High	Low	High	○
PoC	Public, PL.	High	High	High	Low	<50% Storage Space	Low	Low	Very High	○
PoET	Private, P. or PL.	Medium	High	Low	High	N/A	Low	Low	High	●
PoS	Public P. or PL.	High	High	Medium	Low	<51% Stakes	Medium	Low	High	◐
DPoS	Public, PL.	Medium	High	Medium	High	<51% Validators	Medium	N/A	High	◐
LPoS	Public PL.	High	High	Medium	Low	<51% Stakes	Medium	Low	High	○
PoI	Public PL.	High	High	Medium	High	<51% Importance	Low	Low	High	◐
PoA	Public, PL.	High	High	Medium	Low	<51% Online Stakes	High	Low	High	○
Casper	Public, PL.	High	High	Medium	Medium	<51% Validators	Medium	Low	High	○
PoB	Public, PL.	High	High	High	Low	<25% Computing Power	Medium	Low	High	○
PBFT	Private, P.	Medium	Low	Low	High	<33% Faulty Replicas	Low	High	High	●
dPBFT	Private, P.	Medium	High	Medium	High	<33% Faulty Replicas	Low	High	High	◐
Stellar	Public, PL.	High	High	Medium	High	Variable	Low	Medium	High	◐
Ripple	Public P.	High	High	Medium	High	<20% Faulty UNL nodes	Low	Medium	High	◐
Tendermint	Private, P.	Medium	High	Low	High	<33% Voting power	Low	High	High	◐
ByzCoin	Public, PL.	High	High	Medium	High	<33% Faulty Replicas	High	Medium	High	○
Algorand	Public, PL.	High	High	Medium	Medium	<33% Weighted Users	Low	High	High	○
Dfinity	Public, P. or PL.	High	High	Medium	N/A	N/A	Low	N/A	N/A	○
RSCoin	Private, P.	Low	High	Low	High	N/A	Low	Medium	High	○
Elastico	Public, PL.	High	High	High	Low	<25% Faulty VAlidators	Medium	High	High	○
OmniLedger	Public, PL.	High	High	Medium	High	<25% Faulty Validators	Medium	Medium	Low	◐
RapidChain	Public, PL.	High	High	Medium	High	<33% Faulty Validators	Medium	Low	Low	◐
Raft	Private, P.	Medium	High	Low	High	<50% Crash Fault	Low	N/A	High	◐
Tangle	Public, PL.	Medium	High	Low	High	<33% Computing Power	Low	Low	Low	●

Figure 4.2: Consensus protocols compared by [42].

4.3.3 Blockchain Framework

Section 2.2.2 highlights a range of technologies that can be employed to construct blockchains. Within this subsection, a comprehensive analysis and discussion of some of these technologies will be undertaken, facilitating the identification of an optimal choice that aligns with the proposed use case. By delving deeper into their features and capabilities, a more informed decision can be made regarding the selection of suitable blockchain technology.

- **IOTA**: IOTA is built with the IoT in mind. It distinguishes itself with its unique architecture called the Tangle, which is a Directed Acyclic Graph (DAG) that offers scalability, feeless transactions, and low resource requirements. The Tangle eliminates the need for traditional miners and enables devices to participate in the network consensus, making it highly suitable for resource-constrained IIoT environments.

- **Corda**: Corda's architecture is designed to accommodate various deployment scenarios, including public, private, and consortium networks and it is very popular in the enterprise industry. Unlike traditional blockchain platforms, Corda adopts a unique approach by focusing on privacy and selective data sharing.

- **Quorum**: Quorum is an Ethereum fork for the creation of a permissioned network. It introduces enhanced privacy features, such as the private transaction identifier used to ensure data privacy and private contracts, making it suitable for sensitive data handling in industrial settings. Additionally, Quorum supports Raft and Istanbul Byzantine Fault Tolerant (IBFT) consensus mechanisms.

- **Hyperledger Fabric**: Fabric provides a permissioned network structure, ensuring controlled access and participation, which is essential for maintaining the security and privacy of sensitive data. Its modular architecture allows for flexibility in implementing custom consensus algorithms, and privacy controls, enabling tailored solutions for multiple use cases. Fabric uses the concept of chaincodes, which are smart contracts that facilitate automation and business logic enforcement within IIoT networks. In the current version of Fabric, the Raft consensus protocol is recommended in the network. However, this consensus mechanism may introduce some latency in transaction processing, impacting real-time IIoT applications.

- **Hyperledger Sawtooth**: Sawtooth supports both public and private networks, making it suitable for a variety of use cases, including consortiums where multiple organizations collaborate. Sawtooth also employs a unique consensus algorithm called Proof of Elapsed Time (PoET), which optimizes energy consumption in a distributed network. Additionally, Sawtooth's transaction family model allows the development of custom transaction logic tailored to specific IIoT requirements, ensuring efficient and secure data exchange.

- **Hyperledger Besu**: Besu is an Ethereum-based client, which uses the Ethereum Virtual Machine (EVM), enabling smart contracts and decentralized applications to be executed on the network. Besu also supports various consensus protocols, including PoW and PoA. Moreover, Besu provides ways for monitoring node and network performance, using Prometheus metrics and tools such as Block Explorer and EthStats Network Monitor.

After analyzing these frameworks, the options that made more sense were Hyperledger Fabric and Sawtooth. Due to its modular architecture, flexible consensus mechanism, and support for private transactions, Fabric appears a good option, and this work will use this framework to develop and test the blockchain. Although Sawtooth uses an efficient protocol, PoET, it had less information on how to set up the network itself, while Fabric, by being more popular, had more resources to guide the architecture creation. Another considered factor was that Sawtooth didn't provide a compatible benchmarking tool, such as Hyperledger Caliper, to analyze the scalability of the system. Many authors also used this framework, for instance, [16] explored the utilization

and evaluation of Fabric nodes running on Raspberry Pis. The objective was to develop a smart grid solution capable of simulating Virtual Power Plants, as described in section 3.1.6. This study considers it a viable option for implementation, although it acknowledged that it should be tested in different use cases.

Another very promising technology is IOTA but was discarded because, in the current state, the technology relies on a centralized entity operated by the Foundation developing the technology. Furthermore, the network is still in development and the ecosystem is still in the early stages, limiting the available tools and libraries. Nevertheless, this technology could be a great option in the future when the technology is more robust and tested.

4.4 Monitoring & Data Visualization

As mentioned in the requirements, the real-time monitoring of the information flow is one of the aspects that should be implemented in this management system. To handle this issue, this work will use Prometheus[2] and Grafana[3], open-source tools for monitoring and data visualization respectively.

Fabric already provides an interface for gathering network-related metrics by Prometheus, which makes it suitable for this framework. Prometheus also allows the setup of custom metrics, which will be useful to monitor the status of the multiple sensors.

Moreover, the data collected by Prometheus can be seamlessly integrated with Grafana. Grafana allows the creation of custom dashboards to visualize the metrics and alerts stored by Prometheus.

4.5 Final Design Choices

To sum up, the decisions presented in this section, the network will follow a permissioned design. Consequently, the Hyperledger Fabric with the Raft protocol will be used, allowing the creation of authentication policies and ensuring that if a minority of nodes becomes faulty or crashes the system is still operational. This technology will proceed to feed the Prometheus with metrics data, that will then be used by Grafana to build the monitoring dashboards.

Figure 4.3 presents the different layers of the system. At the lower layer the sensors are located, these will periodically emit their status which will be collected by the upper layer. The application layer will use the blockchain smart contracts, invoked with the help of FBs being executed through Dinasores instances, to update the sensor states in the ledger. The blockchain layer will store this data in a LevelDB-based ledger, one of the available storage options in Fabric that will be better presented in section 5.1. Finally, the monitoring layer will use the Fabric endpoints to collect the metrics data to Prometheus and use it in Grafana to enable the data visualization.

[2] https://prometheus.io/
[3] https://grafana.com/

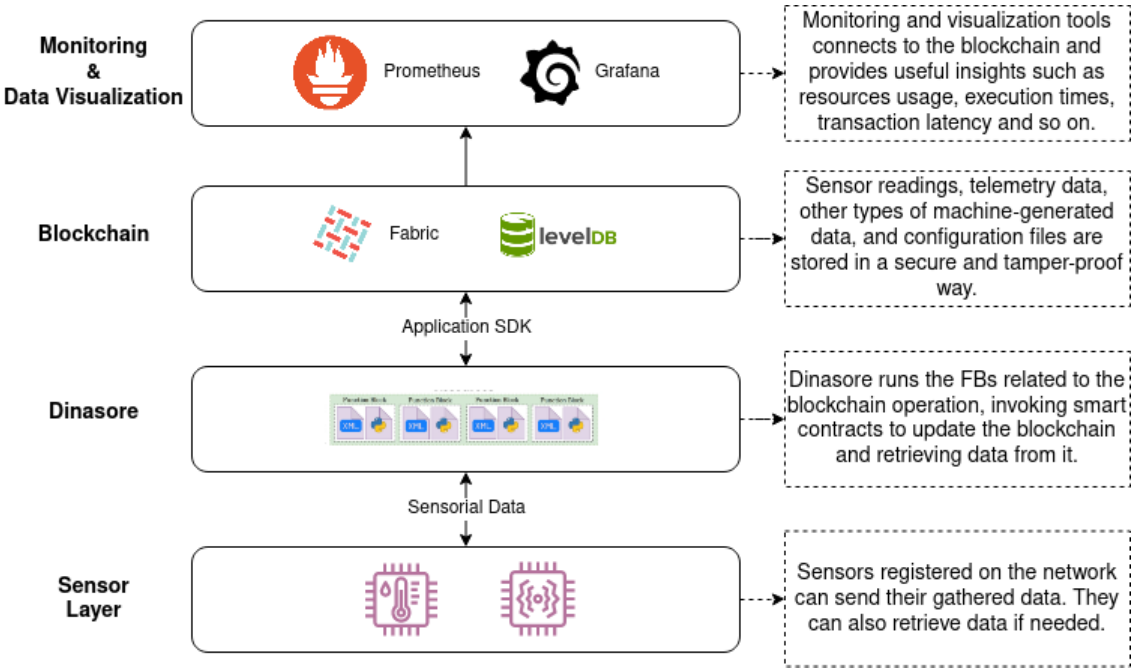


Figure 4.3: Architecture layers.

Chapter 5

Implementation

This chapter details how the system was implemented, describing more extensively the design choices and the final technologies being used, and how the devices will compose the network. The use of the IEC 61499 specification is also defined.

5.1 Blockchain Layer

The blockchain choice was the Hyperledger Fabric, as described in Chapter 4. This section explains the architecture, features, and development details of creating a blockchain with this framework.

5.1.1 Fabric Architecture

The Fabric model is composed of different types of nodes. The most fundamental nodes are the peers. These manage the ledger by invoking smart contracts, and let client applications interact with them by exposing APIs, such as the Fabric Gateway that submits transactions to the network. Fabric provides SDKs in multiple languages to enable the development of such applications.

Each peer in the network participates in at least one channel, that could have multiple peers assigned to it as the use case requires and the relying infrastructure supports. The channel holds its ledger and smart contracts, working as a private network, allowing secure communication for the participants within this network. The peers should store a copy of the ledgers and smart contracts of the channels it is in. To access the resources and services of a channel, the client applications interact with peers in the respective channel.

The network in Fabric is managed by one or more organizations. Each organization operates its own peers, contributes resources to the network, and communicates and achieves consensus with other organizations through mutual channels.

Fabric allows communication to be secure using Transport Layer Security (TLS), so peers need to have their identity assigned by digital certificates, that are issued by administrators of their organization. Moreover, the Membership Service Provider (MSP), is used to define the role and the authorized resources for a peer based on its identity and the policies configured in the channel.

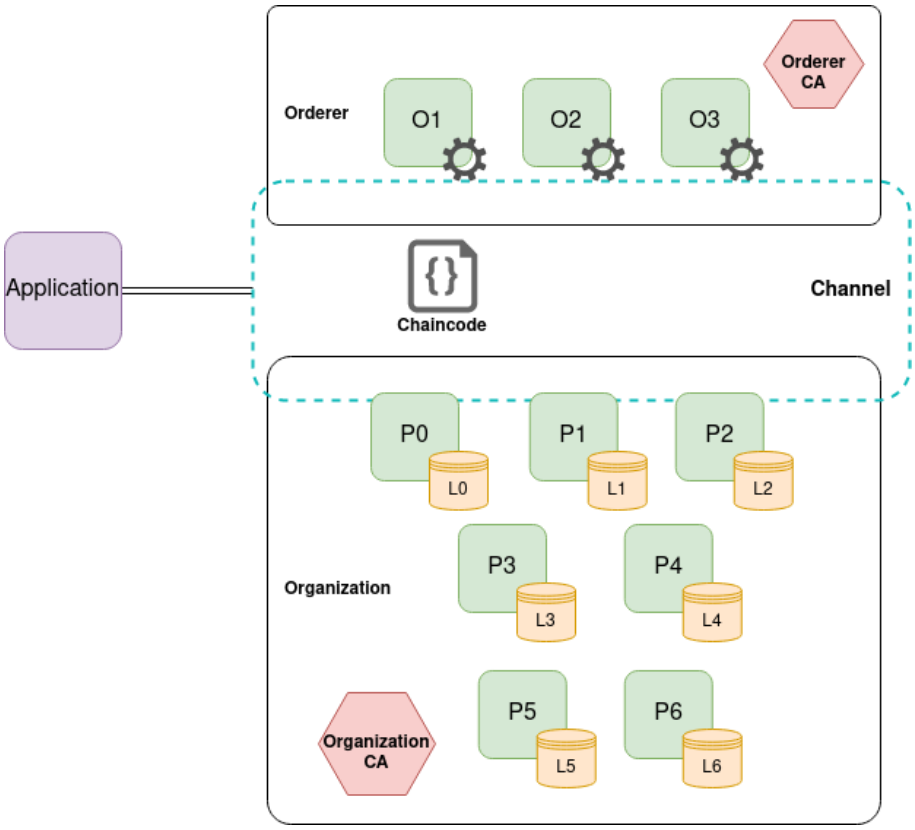


Figure 5.1: Structure of the blockchain network.

The peers in the network can initiate transactions, but to reach a consensus on the order and content of transactions, another type of node is used. The orderer nodes are responsible for the ordering and delivery of transactions to the peers in the network.

Figure 5.1.1 shows a representation of the proposed architecture. It is composed of 7 peers (P0, P1, P2, ..., P6) and 3 orderers (O1, O2, O3) of the same organization. All nodes communicate in the same channel, peers use the channel's chaincode and store the distributed ledger (L0, L1, L2, ..., L6). The organization has two certificate authorities (CAs), the organization CA creates the certificates required for the peers, and the orderer CA is responsible for the orderer certificates. A client application connects to peers in the channel and invokes or query chaincodes. In the image, each component is shown separately, but it is possible for a single device to execute multiple roles.

A single orderer in the network would create a single point of failure. To address this issue using the raft consensus protocol, there must be at least 3 orderer nodes in the network. This is due to the necessary quorum to achieve consensus in the network, which requires the majority of nodes (N/2 + 1, being N the number of nodes). Thus, 2 nodes would not solve the problem, but with 3 nodes, it is possible to handle one node fault. Ideally, the network should have more orderer nodes to enhance its fault tolerance. However, this minimum number was employed so that it could be demonstrated in a test bed.

5.1.2 Data Stored

Fabric allows the specification of the asset's structure stored in the ledger. In this use case, assets represent the events of a sensor. Each event has its timestamp, the identification of the associated sensor, and the value of the event, which can represent data such as temperature, pressure, humidity, and so on. This asset is defined in a chaincode, with the required functions to operate it.

Two storage systems are available in Fabric, these are the LevelDB and InfluxDB. LevelDB is a key-value store that offers efficient storage utilization but does not support complex querying on the stored data. On the other hand, InfluxDB is a time-series database that handles large volumes of data while having features to allow analysis and querying of complex data. Although the InfluxDB time-series characteristics enable efficient real-time analytics, this work uses the LevelDB storage option as it is more lightweight for use in resource-limited devices.

5.1.3 Chaincodes

As mentioned in section 5.1.1, the peers interact with the ledger by using chaincodes.

A single Fabric chaincode can define multiple smart contracts that execute some transaction logic for a specific business process. These are associated with a channel and stored by its peers. When one chaincode is committed to a channel, it becomes available for the applications on the network to call them, and the required operation will be executed by a peer when requested. After the execution, the changes are not directly introduced to the blockchain. First, each peer in the respective channel will verify if the transaction is valid according to the endorsement policy, which states which organizations must approve the transaction. Finally, the peer will verify whether the transaction changes are being applied so that no other update happened in the meantime. The blockchain history is then updated after all validations are complete.

There are two ways of using an operation in a chaincode, by invoking or querying it. Invoke operations are used to modify the ledger's state, which requires the endorsement policies to be valid and the changes become immutably recorded. Query operations, on the other hand, allow the retrieval of data, including past states of an asset, or perform operations on it without updating any state. While invoke operations can read the ledger state, queries are more efficient for this purpose as they do not require additional validations.

The information gathered from the sensors is handled and tracked by multiple smart contracts. These are located within a chaincode where the sensor's asset is also defined. These smart contracts were implemented using the Go programming language (refer to Appendix A.2). The file encompasses contracts for adding assets, allowing the update and creation of these assets, and deleting assets. However, it's important to note that even when an asset is deleted, the ledger history still keeps track of it, and only from the current state it is removed. Moreover, it contains several methods to read the current state of one or multiple assets or to retrieve the historical changes associated with a specific asset.

Chaincodes also allow smart contracts to emit events that can be specified inside the chaincode's code. These events can notify external applications subscribed to the event and trigger

responses accordingly. This feature also enables the blockchain to be integrated with off-chain solutions, such as databases, as described in section 3.2. However, the feature was not implemented in the present work as the use case did not require it. Future implementations could benefit from it by extending the blockchain network to other systems.

5.2 Application Layer

The application layer is responsible for gathering the sensors' data, processing it, and storing it in the blockchain. This execution flow will be possible using the DINASORE, which allows the specification of FBs in compliance with the IEC 61499.

5.2.1 DINASORE

This work uses the DINASORE framework, a flexible framework that allows the creation of a distributed control system. The application developed with the help of DINASORE is capable of simulating a sensor and storing the data from it in the blockchain ledger.

To use the framework, the first step is to create the required FBs for the application. The process of creating a new function block starts by specifying the FB template. This template needs two files, an XML file, with the encoded metadata, and a Python file, that includes the functionality of the block.

5.1 presents an example of a simplified XML file for the FB named **FABRIC_INVOKE**. This file contains some important information such as the input events, that triggers the execution of the block. The file also defines the output events, that can work as input of another FB. The input and output variables, used by each type of event are also defined, with their respective type and name.

Listing 5.1: FB XML.

```
<FBType Name="FABRIC_INVOKE" OpcUa="SERVICE">
    <InterfaceList>
        <EventInputs>
            <Event Name="INIT" Type="Event"/>
            <Event Name="RUN" Type="Event">
                <With Var="PARAMETERS"/>
            </Event>
        </EventInputs>
        <EventOutputs>
            <Event Name="INIT_O" Type="Event"/>
            ...
        </EventOutputs>
        <InputVars>
            <VarDeclaration Name="PEER" Type="STRING" OpcUa="Constant"/>
            ...
            <VarDeclaration Name="PARAMETERS" Type="STRING" OpcUa="Variable.RUN"/>
        </InputVars>
```

```
        <OutputVars>
            <VarDeclaration Name="OUT_JSON_RESPONSE" Type="STRING" OpcUa="Variable.RUN"/>
        </OutputVars>
    </InterfaceList>
</FBType>
```

The second file needed to create the FB is a Python file. This file defines a state machine inside the schedule method, where the event type is checked and the corresponding functionality is executed. This method uses the variables and returns values associated with the XML configuration. In the simplified code 5.2, a function block to invoke Fabric chaincodes is shown. There are two events possible in the code, the **INIT** event is executed once the DINASORE instance starts, and it deals with the blockchain client initialization. The event **RUN** is used to invoke the chaincode. (refer to Appendix A.1 for the complete XML and Python content)

Listing 5.2: FB Python.

```
class FABRIC_INVOKE:
    def __init__(self):
        self.chaincode_handler = None

    def schedule(self, event_input_name, event_input_value, ..., parameters):
        if event_input_name == 'INIT':
            ...
            return [event_input_value, None, None, None]
        elif event_input_name == 'RUN':
            ...
            return [None, event_input_value, parameters]
```

These FBs are loaded in 4DIAC-IDE, where it is possible to arrange the system FBs and map them to a device in the network. To use it with DINASORE, the DINASORE instance must be running so that 4DIAC can update and deploy the new configuration through the OPC UA, a communication protocol popular in the industrial environment.

5.2.2 Prometheus & Grafana Dashboard

The monitoring layer uses Prometheus and Grafana together to create real-time dashboards. The first dashboard contains information on the blockchain status, Figure 5.2. Fabric provides an endpoint for collecting Prometheus metrics. These include many attributes, such as the blockchain height, the processing time, and the rate of transactions of the multiple peers in the network.

Moreover, the developed DINASORE application also creates an endpoint where Prometheus can listen to and collect data. The data collected from this endpoint is the value associated with the sensor as seen in Figure 5.3. In this example, the value of the sensor represents the temperature. In the dashboard, the current status of each connected sensor is periodically updated.

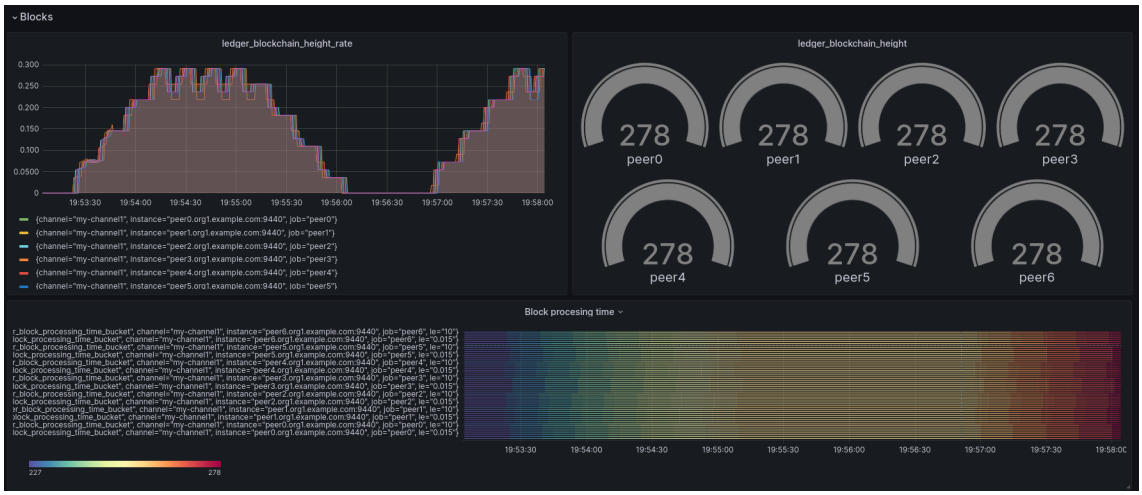


Figure 5.2: Dashboard to visualize the blockchain parameters.

5.2.3 IEC 61499 Specification

In total, the final application is composed of 9 interconnected FBs. The **SENSOR_SIMULATOR** generates uniformly distributed random values to simulate a sensor. Alongside this FB, other FBs were created to perform auxiliary operations. The **REAL_TO_STRING** and **STRING_TO_REAL** convert the values passed as input to string and real types respectively.

CONCAT_STR_WITH_SEPARATOR joins two string inputs by a separator value, the output from this FB is needed to build a JSON string with the FB **BUILD_JSON_STRING**. Finally, the **GET_JSON_VALUE** returns the value from a JSON string given the value's key. Figure 5.4 shows the structure of these FBs.

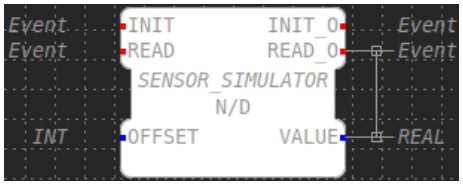
Two more FBs are used to interact with the blockchain. The **FABRIC_INVOKE** connects to one of the network peers using one of the available user's credentials and invokes the chaincode passed as input. The second one is the **FABRIC_QUERY**, which is used to read the state from the blockchain ledger without modifying it. Figure 5.5 shows the structure of both FBs.

An additional FB was created to update a Prometheus gauge value, a metric that represents a single value, the **PROMETHEUS_SET_GAUGE** as seen in Figure 5.6. This value is updated every time the respective asset updates in the ledger, and later queried by Grafana.

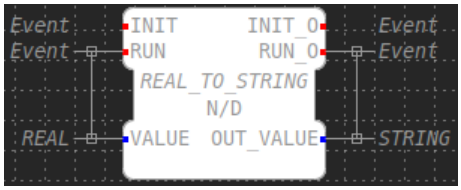
The complete system architecture is presented in Figure 5.7.

The use of FBs also allowed the reusability of blocks in this single system. The same blocks could be used in different systems, increasing the consistency of different applications and reducing the development effort. Moreover, additional FBs could be created and integrated into the system or replace existing ones, as long as the input and output interfaces are preserved.

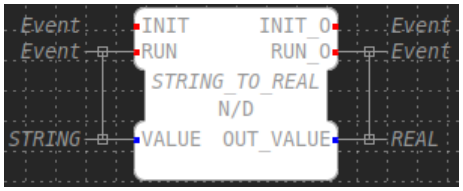
Figure 5.3: Dashboard used to visualize the sensor's status.



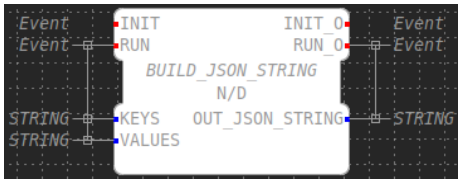
(a) FB that simulates the behavior of a sensor.



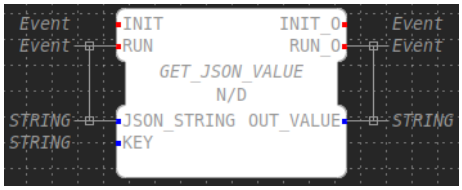
(b) FB used to convert real to string type.



(c) FB used to convert string to real type.

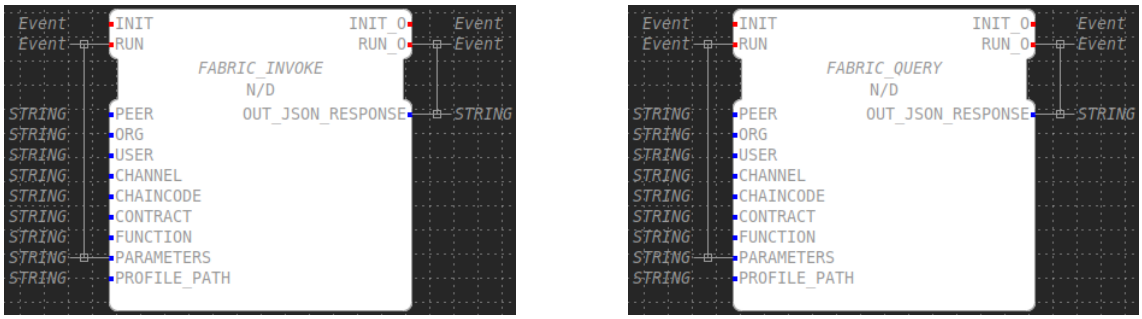


(d) FB to create a string in JSON format.



(e) FB to retrieve a value from a JSON format.

Figure 5.4: Generic FBs used in the application logic.



(a) Invoke FB. (b) Query FB.

Figure 5.5: FBs used to interact with the blockchain.

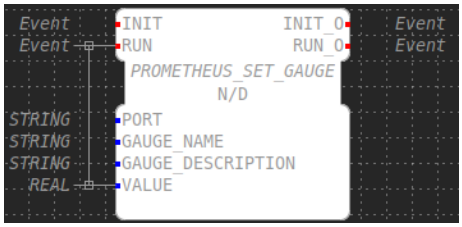


Figure 5.6: FB to interact with Prometheus.

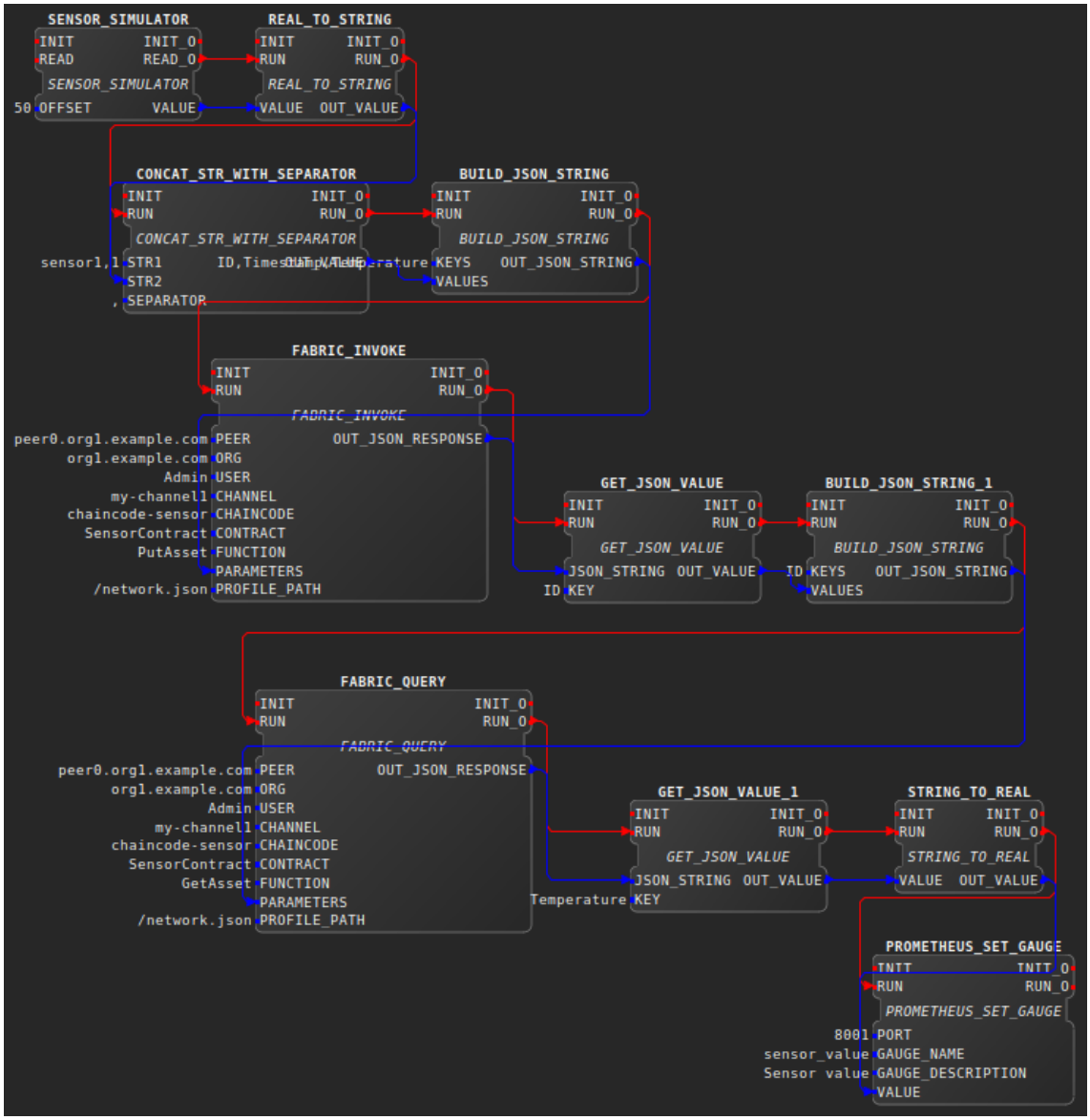


Figure 5.7: Interconnected FBs of the entire application.

Chapter 6

Validation

This Chapter presents the tests and result analysis of the implemented work.

6.1 Test Methodology

The tests will be conducted locally on a computer system equipped with 32GB of RAM and an Intel® Core™ i7-9750H CPU running at a base frequency of 2.60GHz with 12 processor cores.

To validate the suitability and performance of the proposed architecture, tests with an increasing number of peers will be performed. The first set of tests will not be conducted with the developed DINASORE application, as this will allow testing isolated network operations. The first test will address the network as described in 5.1.1. The number of nodes in this network was selected so that it would be able to prepare a testbed for further validation. In the second test, the number of peers in the network is going to be increased to 25, and also an additional test will be done with the same number of peers but with additional orderers. In the third test, the number of peers in the network is going to be also increased. This first set of tests will allow us to test the scalability of the system.

For each test, a fixed load of transactions is going to be present in the backlog. Thus it is possible to test the maximum TPS while maintaining a pending transaction load. The transaction load was set to 20, which would simulate the sensors constantly issuing new transactions to maintain this pending number. This fixed load rate will ensure that the desired transaction rate is maintained consistently throughout the test duration. Furthermore, each test will be conducted within a standardized timeframe of 30 seconds. This short period allows us to quickly complete multiple tests, moreover, after some exploratory configuration testing, this period enabled the completion of a considerable amount of transactions. Thus, this duration will allow for consistent comparisons across multiple iterations and provide sufficient time to capture meaningful performance metrics and analyze system behavior. Furthermore, the tests will cover the two key operations within the network, namely *getAsset* and *putAsset*. This configuration will allow us to test the throughput (TPS), average latency, and total confirmed transactions in the timespan.

To monitor resources and network usage, the tests are going to be completed with the support of Hyperledger Caliper, a benchmarking tool for evaluating the performance and scalability of blockchain platforms. It allows the configuration of tests for different operations on the network and simulates various workloads. The collected metrics that are going to be analyzed are the average CPU and Memory usage, as well as the inbound and outbound data traffic from the nodes.

Additionally, a test will be conducted using the DINASORE application. In this setup, seven instances of DINASORE were employed to send requests for updating the ledger state at intervals of 10 seconds for 90 seconds. This will simulate the sensors associated with each peer in the created network. A script will be responsible for periodically collecting the CPU and memory usage of each instance.

After the collection of results, the data will also be compared to other solutions. It is difficult to compare directly the results between other available solutions, as the environment where the metrics were collected is not standardized. Moreover, not all data is made public or has easy access. To address this issue, the metrics will be compared to multiple existing technologies designed for different use cases. Table 6.1 contains each technology and its registered or stated TPS. Although most of these technologies are mainly being used on distinct use cases than the smart industry, they are helpful as they all require some level of real-time processing of transactions, which is also necessary for our IIoT use case.

In terms of transaction processing speed, these different technologies and services demonstrate varying levels of performance. Bitcoin, with a maximum throughput of 7 TPS. Ripple, on the other hand, claims to consistently handle 1500 TPS. Ethereum currently operates around 24 TPS, with a maximum recorded rate of 56 TPS. Visa, a global payment technology, boasts an average TPS of 1667, with a capacity exceeding 65,000 TPS. Paypal, a widely used payment system, is reported to have an average TPS of 192 according to some sources. IOTA, offering feeless data and value transactions, achieves around 1,000 TPS. VechainThor has a theoretical capacity of handling over 10,000 TPS in the real world.

Finally, a last test to measure the complexity and maintainability of the system will be performed. In this scenario, a second pipeline without the use of the blockchain will be developed, this will replace the blockchain-related FBs for ones connecting to a traditional database system. The test will use Halstead and McCabe's metrics to calculate the Maintainability Index (MI).

In addition to the aforementioned tests, an additional test leveraging a test bed infrastructure was planned but unfortunately not executed as intended. This particular test aimed to deploy the nodes on Raspberry Pi devices. However, during the test execution, it was observed that the nodes encountered difficulties in operating the consensus protocol. This issue could potentially be linked to a problem in the certificates required for network operation with TLS. The encountered challenges in setting up the network infrastructure exemplify the complexities involved in the deployment process.

Name	Succ	Fail	Throughput (TPS)	Avg Latency (s)
getAsset	43270	0	1512.5	0.01
putAsset	3269	0	113.7	0.16

Table 6.2: *getAsset* and *putAsset* metrics with 7 peers and 3 orderers network.

Technology/Service	Average TPS	Max TPS
Bitcoin (Decentralized digital currency) [8]	5	7
Ripple (Real-time gross settlement) [39]	1500	-
Ethereum (Decentralized blockchain with smart contracts and decentralized applications support) [11]	24	56
Visa (Global payment technology) [47]; [31]	1667	65000
Paypal (Payment system) [31]	192	-
IOTA (Distributed ledger designed for IoT) [19]	1000	-
VechainThor (Blockchain-based platform supply chain and business processes) [9]	-	10000

Table 6.1: TPS comparison for different technologies and services.

6.2 Tests

6.2.1 Scalability Tests

6.2.1.1 Test 7 Peers

The first execution used the same network described in Chapter 5, with 7 peers and 3 orderer nodes. In this test, 7 workers used by Caliper submitted the required transactions to maintain the defined load and simulate the sensors. This first test did not use the developed DINASORE application. Table 6.2, shows the performance metrics for the getAsset and putAsset operations.

The considerable disparity in the total number of successful transactions is evident and aligns with the expectations outlined in section 5.1.3. The *getAsset* operation, which does not modify the ledger state and therefore does not require endorsement, recorded more than 13 times the number

of transactions compared to the *putAsset* operation. Consequently, the throughput of *getAsset* was significantly higher, reaching 1512.5, in contrast to the comparatively lower throughput of 113.7 achieved by *putAsset*. Considering the transaction load and the number of peers involved in processing transactions within the constructed scenario, the obtained results appear reasonable and suitable for the specific use case in this test. It is worth noting that the number of sensors involved does not necessitate hundreds of transactions per second, thus the observed transaction volumes are consistent with the expected requirements and system behavior.

In addition to these network metrics, the resources of the nodes were collected. Tables 6.3 and 6.4, present the results for the *getAsset* and *putAsset* respectively.

Name	**CPU%(avg)**	**Memory(avg)[MB]**	**Traffic In [MB]**	**Traffic Out [MB]**
peer0	13.95	136	80.2	79.0
peer1	0.23	97.5	0.413	0.359
peer2	0.24	92.4	0.256	0.261
peer3	0.22	98.3	0.352	0.355
peer4	0.2	113	0.380	0.339
peer5	0.24	106	0.403	0.337
peer6	0.23	118	0.344	0.351
orderer0	0.05	71.6	0.134	0.195
orderer1	0.05	41.0	0.122	0.175
orderer2	0.04	69.6	0.0939	0.154

Table 6.3: Peers and orderers resource consumption for the *getAsset* operation.

The results for *getAsset* were interesting since peer0 used a considerable amount of resources compared to the rest of the network. Upon removing the first peer of the network, the second peer, peer1 emerged as the peer with the highest resource utilization. This is probably due to Caliper using the first peer of the configured network profile to initiate the requests, which would explain the large incoming traffic. However, no specific configuration for load balancing was identified to mitigate this issue. In the *putAsset* test, the load was more evenly distributed, although the peer0 still had a slightly higher resource usage. In this latter test, all peers communicated with each other to endorse the transactions, and this can be noticed by the outcoming traffic of each peer. Overall, the CPU usage remained relatively low despite the substantial number of transactions being processed, and both operations demonstrated comparable memory usage patterns.

Name	Succ	Fail	Throughput (TPS)	Avg Latency (s)
getAsset	25388	0	1011.9	0.02
putAsset	933	0	36.9	0.70

Table 6.5: *getAsset* and *putAsset* metrics with 25 peers and 3 orderers network.

Name	CPU%(avg)	Memory(avg)[MB]	Traffic In [MB]	Traffic Out [MB]
peer0	4.15	144	89.2	77.4
peer1	2.34	101	65.2	52.1
peer2	2.41	95.7	65.4	52.0
peer3	2.32	105	64.5	51.9
peer4	2.29	118	64.5	52.3
peer5	2.36	112	65.7	52.9
peer6	2.68	120	68.3	57.2
orderer0	2.48	87.5	18.3	47.9
orderer1	1.96	49.7	16.0	39.8
orderer2	2.56	88.6	18.3	47.9

Table 6.4: Peers and orderers resource consumption for the *putAsset* operation.

6.2.1.2 Test 25 Peers

Besides the tests with the previous number of peers, additional tests with a higher number of peers were performed to analyze the scalability of the solution. The second test network consists of 25 peers issuing transactions to the blockchain maintaining the same load of pending transactions. The results are shown in Table 6.5. With this configuration, the number of successful transactions was reduced by 70% in the *getAsset* operation, and more than 3 times in the *putAsset* operation.

Table 6.6 shows results for the same configuration but with twice the number of orderers. All orderers contribute to the consensus, and although more orderers can increase the system's fault tolerance, it may also reduce the network performance. This test tried to check this impact, but the differences were not noticeable in a local machine.

6.2.1.3 Test 45 Peers

In the final test, involving 45 peers, Table 6.7, the TPS for the getAsset operation experienced a minor decrease compared to the previous test. However, the putAsset operation witnessed another

Name	Succ	Fail	Throughput (TPS)	Avg Latency (s)
getAsset	25619	0	1020.8	0.02
putAsset	917	0	34.5	0.68

Table 6.6: *getAsset* and *putAsset* metrics with 25 peers and 6 orderers network.

significant decline in TPS. Overall, the CPU and memory usage maintained a similar level to the previous two tests.

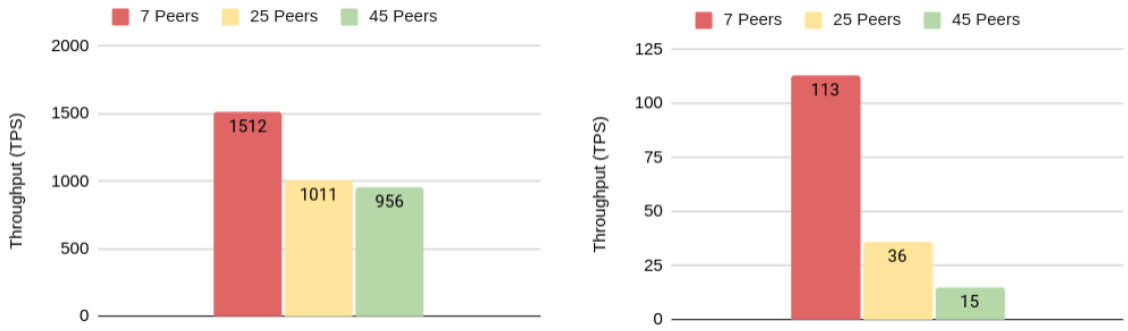
Name	Succ	Fail	Throughput (TPS)	Avg Latency (s)
getAsset	24001	0	956.2	0.02
putAsset	409	0	15.0	1.61

Table 6.7: *getAsset* and *putAsset* metrics with 45 peers and 3 orderers network.

6.2.1.4 Conclusions

The throughput results of the conducted tests are presented in Figure 6.1, showcasing consistent performance in the read operation despite the initial drop during the increase in the number of peers. This outcome aligns with expectations since the read operation doesn't necessitate consensus among multiple peers, allowing it to maintain a high level of performance. It's worth noting that these tests were conducted with a constant load of transactions rather than pushing the system to its maximum TPS limit, implying that the system's capacity might be even higher. In comparison to the findings presented in Table 6.1, the read operation demonstrates suitability for real-time use cases.

On the other hand, the put operation was heavily influenced by the number of peers. A drop in the TPS was expected due to the endorsement policies needed to accept a new transaction in the network. However, the low TPS results might affect real-time requirements. It is important to note that in practical scenarios, not all peers may be exclusively associated with a single sensor but instead handle events from multiple sensors, as exemplified in the study conducted by [17] in section 3.1.4. Moreover, it is worth mentioning that in networks with lower TPS, such as Bitcoin, scaling solutions like the [25] have been implemented to enhance transaction throughput and reduce latency. Similarly, Ethereum has also explored various solutions like layer 2 scaling solutions, such as Plasma, to improve the scalability and performance of the network [6]. Consequently, considering the system's scalability, there is potential for improvement in accommodating a larger number of sensors and optimizing the overall performance.



(a) *getAsset* throughput.

(b) *putAsset* throughput.

Figure 6.1: Throughput results.

6.2.2 DINASORE Test

In addition to conducting tests with the Caliper benchmarking tool, the network was subjected to local testing using the DINASORE application to execute transactions alongside the remaining operations detailed in section 5.2.1. In total, 61 transactions were processed in the 90 seconds interval by all peers. Although the number of submitted transactions was considerably lower compared to the previous tests, each DINASORE instance utilized an average of 0.4% of the available system memory (≈124MB), which was similar, but all 12 available cores were heavily utilized by the entire system.

With only this data it is not clear why the tests with DINASORE had a huge difference in CPU usage, even when considering the additional steps of the application. Caliper is mostly written in JavaScript and the SDK used to create the tests was also in this programming language. Although the performance of a language depends on its use case, JavaScript tends to have a better computational performance when compared to Python, the main language used in DINASORE, and the developed FBs, as demonstrated by the benchmarks in [27]. This factor combined that the developed DINASORE application is performing more operations could indicate the observed results. Overall, this utilization of resources appeared to present a performance bottleneck when employing DINASORE in particular use cases.

6.2.3 Maintainability Test

A similar IEC 61499 pipeline system was created without the use of the developed blockchain modules to compare its code complexity with our blockchain approach. The main focus was to evaluate the maintainability index of both solutions. The results revealed that the blockchain approach achieved a MI score of 103.94, while the solution without blockchain obtained a slightly higher score of 108.21. These results indicate a similarity in terms of maintainability between the two approaches. However, it is worth noting that the initial setup phase of the blockchain-based approach posed additional challenges.

Chapter 7

Conclusions

A blockchain architecture for IIoT management was developed. This architecture uses a permissioned approach, ensuring data security and confidentiality, while also enhancing the overall efficiency of the system. Furthermore, the solution uses an application compliant with the IEC 61499 to allow the interaction between sensors, the distributed ledger, and the monitoring system, facilitating real-time data capture.

The proposed solution could be used to maintain a management system for resource-limited devices. As seen in the tests, the architecture was capable of handling a considerable amount of transactions per second in our specified use case. Moreover, the system was able to perform its operations while maintaining efficient CPU and memory utilization on scalability tests. However, it is essential to address certain limitations encountered during this research. One notable limitation pertains to the utilization of the DINASORE application, which imposed a significant burden on CPU resource usage. As discussed in Section 6.2.1.4, the introduction of bases to aggregate the output of multiple sensors could alleviate the strain on individual instances, as relying solely on sensor resources proved challenging in terms of scalability. Additionally, the complexity involved in network setup presented difficulties in adding new peers, potentially resulting in higher implementation costs when deploying the solution in real-world scenarios.

The study also showed the successful integration of the IEC 61499 standard into the developed blockchain solution. In the proposed architecture, the standard enabled the use of smart contracts that were responsible for operating the use case logic, such as storing or reading the data from the distributed ledger through the action of function blocks. This integration of IEC 61499 not only enhanced the functionality of the blockchain solution but demonstrated a way for greater interoperability and standardization in IIoT management systems. For instance, the developed function blocks could be applied to other use cases and pipelines. Moreover, the solution presented a similar impact on the complexity and maintainability of the system when compared to a system without blockchain support on the executed tests.

In summary, this research contributes to the advancement of blockchain-based solutions for IIoT management, mitigating the single point of failure and providing an example of how the IEC 61499 could be used. Nevertheless, the solution showed possible problems with its scalability and

the challenges encountered highlight the need for careful consideration of the specific use case requirements when implementing a similar solution. This work thus provides insights that can guide future efforts in optimizing and tailoring the solution for various industry use cases.

7.1 Future Work

The developed work approach and the limitations encountered during its implementation provide valuable insights for potential future work. From the results and information collected in this study, further investigations can be conducted to address the identified limitations and explore new possibilities.

Although the implemented use case didn't require much storage space for the sensor events, this might be a problem for other use cases. To address this challenge, the integration of an off-chain solution could be explored. By using off-chain storage mechanisms, such as distributed file systems or decentralized storage networks, the system can offload sensor data to external storage layers while maintaining necessary references or proofs on the blockchain.

Furthermore, the adoption of cloud-based solutions for hosting the blockchain nodes can greatly enhance the efficiency and flexibility of the blockchain system, improving its scalability. In this context, sensors can utilize cloud-based services to invoke smart contracts and update the ledger state.

One notable limitation of the implemented approach was the exclusive reliance on the Python programming language within the DINASORE application. For instance, the frameworks used for blockchain development were selected also based on the criteria of having this language available in its development kit. While Python was used in DINASORE to leverage its robust machine learning and data handling libraries, the framework's support for other programming languages could significantly enhance its versatility and applicability across various use cases. Enabling the execution of function blocks in multiple languages would offer developers greater flexibility and the ability to tailor their solutions to specific requirements, expanding the framework's potential reach and impact.

Moreover, the framework used for the blockchain, Fabric, although having multiple useful features for introducing use case policies, governance, and modularity in its architecture, posed challenges due to its complexity and steep learning curve. Exploring alternative blockchain frameworks that offer a more simplified development experience without compromising security and functionality should be considered.

For instance, as described in Chapter 4, IOTA could be another great option to be tested in the future. Many of the same concepts and characteristics of a blockchain are present in this technology, including decentralized, transparent, and immutable ledger storage. The technology is designed specifically for low-power and low-bandwidth devices. But this one still has its limitations that should be studied and addressed. At the moment, IOTA uses a coordinator, a temporary solution maintained by the IOTA team to ensure the system's integrity, however, the decentralization of this component is in development and could be tested in later studies.

Appendix A

Example Codes

A.1 Function Block Example

Listing A.1: XML FB part for Fabric Invoke.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd">
<FBType Name="FABRIC_INVOKE" OpcUa="SERVICE">
    <InterfaceList>
        <EventInputs>
            <Event Name="INIT" Type="Event"/>
            <Event Name="RUN" Type="Event">
                <With Var="PARAMETERS"/>
            </Event>
        </EventInputs>
        <EventOutputs>
            <Event Name="INIT_O" Type="Event"/>
            <Event Name="RUN_O" Type="Event">
                <With Var="OUT_JSON_RESPONSE"/>
            </Event>
        </EventOutputs>
        <InputVars>
            <VarDeclaration Name="PEER" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="ORG" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="USER" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="CHANNEL" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="CHAINCODE" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="CONTRACT" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="FUNCTION" Type="STRING" OpcUa="Constant"/>
            <VarDeclaration Name="PARAMETERS" Type="STRING" OpcUa="Variable.RUN"/>
            <VarDeclaration Name="PROFILE_PATH" Type="STRING" OpcUa="Constant"/>
        </InputVars>
        <OutputVars>
            <VarDeclaration Name="OUT_JSON_RESPONSE" Type="STRING" OpcUa="Variable.RUN"/>
        </OutputVars>
```

```
    </InterfaceList>
</FBType>
```

Listing A.2: Python FB part for Fabric Invoke.

```
import asyncio
import time
import json
from hfc.fabric import Client
from hfc.fabric.chaincode import ChaincodeExecutionError

class FABRIC_INVOKE:

    def __init__(self):
        self.chaincode_handler = None

    def schedule(self, event_input_name, event_input_value, peer, \
                 org, user, channel, chaincode, \
                 contract, function, parameters, profile_path):

        if event_input_name == 'INIT':
            # Initialize node connection
            attempts = 3
            while attempts > 0:
                try:
                    self.chaincode_handler = ChaincodeHandler(profile_path, channel, chaincode) \
                        .setPeer(peer) \
                        .setUser(org, user)
                    break
                except:
                    attempts -= 1
                    time.sleep(1)
            return [event_input_value, None, None, None]

        elif event_input_name == 'RUN':
            if self.chaincode_handler == None:
                return [None, None, None, None]
            json_parameters = json.loads(parameters)
            list_parameters = [json_parameters[param] for param in json_parameters]
            try:
                self.chaincode_handler.invoke(contract, function, list_parameters)
            except ChaincodeExecutionError as e:
                print("Chaincode execution failed", e)
                return [None, None, None]
            return [None, event_input_value, parameters]

class EventHandler:
    def maxValueReached(event, block_num, tx_id, tx_status):
        print(event['payload'])
```

```
class ChaincodeHandler:
    def __init__(self, net_profile, channel, chaincode) -> None:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)
        self.client = Client(net_profile=net_profile)
        self.client.new_channel(channel)
        self.channel = channel
        self.chaincode = chaincode

        self.loop = None

        self.peer = None
        self.user = None
        self.eventHub = None
        self.eventTask = None

    def setPeer(self, peer: str):
        self.peer = self.client.get_peer(peer)
        return self

    def setUser(self, organization: str, username: str):
        self.user = self.client.get_user(org_name=organization, name=username)
        return self

    def invoke(self, contract: str, function: str, args: list):
        if self.user == None:
            return
        try:
            loop = asyncio.get_event_loop()
        except:
            loop = asyncio.new_event_loop()
            asyncio.set_event_loop(loop)
        print("args:", args)
        return loop.run_until_complete(self.client.chaincode_invoke(
                requestor=self.user,
                channel_name=self.channel,
                peers=[self.peer],
                args=args,
                cc_name=self.chaincode,
                fcn=contract + ":" + function,
                transient_map=None,
                wait_for_event=True,
                ))

    def query(self, contract: str, function: str, args: list):
        if self.user == None:
            return
```

```
            try:
                loop = asyncio.get_event_loop()
            except:
                loop = asyncio.new_event_loop()
                asyncio.set_event_loop(loop)

            return loop.run_until_complete(self.client.chaincode_query(
                    requestor=self.user,
                    channel_name=self.channel,
                    peers=[self.peer],
                    args=args,
                    cc_name=self.chaincode,
                    fcn=contract + ":" + function,
                    transient_map=None
                    ))

    def registerEvent(self, event: str, function):
        self.eventHub.registerChaincodeEvent(self.chaincode, event, onEvent=function)
        return self

    def createEventHub(self):
        if self.user == None or self.peer == None:
            return
        self.eventHub = self.client.get_channel(self.channel).newChannelEventHub(self.peer, self.user)
        return self

    def connectEventHub(self):
        self.eventTask = self.loop.create_task(self.eventHub.connect(filtered=False))
        return self

    def disconnectEventHub(self):
        self.eventHub.disconnect()
        return self

    def closeLoop(self):
        self.loop.close()
```

A.2 Sensor Chaincode

Listing A.3: Sensor Chaincode.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6     "log"
7
```

```
8       "github.com/hyperledger/fabric-contract-api-go/contractapi"
9   )
10
11  type SensorContract struct {
12      contractapi.Contract
13  }
14
15  type SensorAsset struct {
16      ID string `json:id`
17      Timestamp string `json:timestamp`
18      Temperature float32 `json:value`
19  }
20
21  func (s *SensorContract) HasAsset(ctx contractapi.TransactionContextInterface, id string) (bool, error) {
22      sensorAsset, err := ctx.GetStub().GetState(id)
23
24      if err != nil {
25          return false, fmt.Errorf("%v", err)
26      }
27
28      return sensorAsset != nil, nil
29  }
30
31  /*
32  Creates or updates a sensor asset. Emits an event if successful.
33  */
34  func (s *SensorContract) PutAsset(
35      ctx contractapi.TransactionContextInterface,
36      id string,
37      ts string,
38      value float32,
39  ) error {
40
41      timestamp, err := ctx.GetStub().GetTxTimestamp()
42      ts = fmt.Sprint(timestamp.Seconds)
43
44      sensorAsset := SensorAsset{
45          ID: id,
46          Timestamp: ts,
47          Temperature: value,
48      }
49
50      assetJSON, err := json.Marshal(sensorAsset)
51
52      if err != nil {
53          return err
54      }
55
56      ctx.GetStub().SetEvent(
```

```
57                  "put-asset",
58                  []byte(fmt.Sprintf("{ id: \"%s\", timestamp: \"%s\", value: \"%f\"}", id, ts, value))
59              )
60
61      return ctx.GetStub().PutState(id, assetJSON)
62  }
63
64  /*
65  Deletes a sensor asset
66  */
67  func (s *SensorContract) DeleteAsset(
68                  ctx contractapi.TransactionContextInterface, id string
69              ) error {
70      exist, err := s.HasAsset(ctx, id)
71      if err != nil {
72          return fmt.Errorf("Error: %v", err)
73      }
74      if !exist {
75          return fmt.Errorf("Asset with id: %s does not exist", id)
76      }
77
78      return ctx.GetStub().DelState(id)
79  }
80
81  /*
82  Gets a sensor asset given its id
83  */
84  func (s *SensorContract) GetAsset(
85                  ctx contractapi.TransactionContextInterface, id string
86              ) (*SensorAsset, error) {
87      sensorAssetJSON, err := ctx.GetStub().GetState(id)
88
89      if err != nil {
90          return nil, fmt.Errorf("Error: %v", err)
91      }
92
93      if sensorAssetJSON == nil {
94          return nil, fmt.Errorf("Asset with id: %s does not exist", id)
95      }
96
97      var sensorAsset SensorAsset
98      err = json.Unmarshal(sensorAssetJSON, &sensorAsset)
99      if err != nil {
100         return nil, err
101     }
102
103     return &sensorAsset, nil
104 }
105
```

```
106 /*
107 Gets multiple assets given a JSON array of ids
108 */
109 func (s *SensorContract) GetAssets(
110                 ctx contractapi.TransactionContextInterface, ids []string
111         ) (assets []*SensorAsset, err error) {
112
113     for _, id := range ids {
114         assetJSON, err := ctx.GetStub().GetState(id)
115
116         if err != nil {
117             return nil, err
118         }
119
120         if assetJSON == nil {
121             continue
122         }
123
124         var sensorAsset SensorAsset
125         err = json.Unmarshal(assetJSON, &sensorAsset)
126
127         if err != nil {
128             return nil, err
129         }
130
131         assets = append(assets, &sensorAsset)
132     }
133
134     return
135 }
136
137 /*
138 Gets all sensor assets
139 */
140 func (s *SensorContract) GetAllAssets(
141                 ctx contractapi.TransactionContextInterface
142         ) (assets []*SensorAsset, err error) {
143     stateIterator, err := ctx.GetStub().GetStateByRange("", "")
144
145     if err != nil {
146         return nil, err
147     }
148
149     defer stateIterator.Close()
150
151     for stateIterator.HasNext() {
152         assetJSON, err := stateIterator.Next()
153
154         if err != nil {
```

```
155                 return nil, err
156             }
157
158             var sensorAsset SensorAsset
159             err = json.Unmarshal(assetJSON.Value, &sensorAsset)
160
161             if err != nil {
162                 return nil, err
163             }
164
165             assets = append(assets, &sensorAsset)
166         }
167
168         return
169 }
170
171 /*
172 Gets a sensor asset state history given its 'id'. Returns only the 'max' most recent states
173 */
174 func (s *SensorContract) GetAssetHistory(
175                 ctx contractapi.TransactionContextInterface,
176                 id string, max int
177             ) (assets []*SensorAsset, err error) {
178     historyIterator, err := ctx.GetStub().GetHistoryForKey(id)
179
180     if err != nil {
181         return nil, err
182     }
183
184     defer historyIterator.Close()
185
186     for historyIterator.HasNext() && len(assets) < max {
187         assetJSON, err := historyIterator.Next()
188
189         if err != nil {
190             return nil, err
191         }
192
193         var sensorAsset SensorAsset
194         err = json.Unmarshal(assetJSON.Value, &sensorAsset)
195
196         if err != nil {
197             return nil, err
198         }
199
200         assets = append(assets, &sensorAsset)
201     }
202
203     return
```

```
204 }
205
206 func main() {
207     chaincode, err := contractapi.NewChaincode(&SensorContract{})
208     if err != nil {
209         log.Panicf("Error creating sensor chaincode: %v", err)
210     }
211
212     if err := chaincode.Start(); err != nil {
213         log.Panicf("Error starting sensor chaincode: %v", err)
214     }
215 }
```

References

[1] Sasikumar A., Subramaniyaswamy Vairavasundaram, Ketan Kotecha, Indragandhi V., Logesh Ravi, Ganeshsree Selvachandran, and Ajith Abraham. Blockchain-based trust mechanism for digital twin empowered industrial internet of things. *Future Generation Computer Systems*, 141:16–27, 2023.

[2] Amal Alahmadi and Xiaodong Lin. Towards secure and fair iiot-enabled supply chain management via blockchain-based smart contracts. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[3] Muhammad Salek Ali, Massimo Vecchio, Miguel Rodrigo Pincheira Caro, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. Applications of blockchains in the internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, PP:1–1, 2018.

[4] Alain Aoun, Adrian Ilinca, Mazen Ghandour, and Hussein Ibrahim. A review of industry 4.0 characteristics and challenges, with potential improvements using blockchain technology. *Computers & Industrial Engineering*, 162:107746, 2021.

[5] C. Buratti, A. Conti, D. Dardari, and R. Verdone. An overview on wireless sensor networks technology and evolution. *Sensors*, 9(9):6869–6896, 2009.

[6] George Burlakov. Layer 2 scaling solutions for EVM-based blockchains: a look at Plasma, Rollups, Sidechains. https://www.linkedin.com/pulse/layer-2-scaling-solutions-evm-based-blockchains-look-plasma-burlakov/, 2023. [Online; accessed June 2023].

[7] Ricardo Colomo-Palacios, Mary Sánchez-Gordón, and D. Arias-Aranda. A critical review on blockchain assessment initiatives: A technology evolution viewpoint. *Journal of Software: Evolution and Process*, 32, 05 2020.

[8] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[9] CryptoEQ. Vechain. https://www.cryptoeq.io/corereports/vechain-abridged, 2023. [Online; accessed June 2023].

[10] Eclipse Foundation. 4diac. https://www.eclipse.org/4diac/, 2023. [Online; accessed June 2023].

[11] ethtps.info. ETH TPS. https://ethtps.info/, 2023. [Online; accessed June 2023].

[12] Xinxin Fan and Qi Chai. Roll-dpos: A randomized delegated proof of stake scheme for scalable blockchain-based internet of things systems. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MobiQuitous '18, page 482–484, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971, 2020.

[14] Gartner. Gartner predicts 75% of ceos will be personally liable for cyber-physical security incidents by 2024, 2020. Available at https://www.gartner.com/en/newsroom/press-releases/2020-09-01-gartner-predicts-75--of-ceos-will-be-personally-liabl, last accessed in December 2022.

[15] Andrija Goranović, Marcus Meisel, Stefan Wilker, and Thilo Sauter. Hyperledger fabric smart grid communication testbed on raspberry pi arm architecture. In *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, 2019.

[16] Andrija Goranović, Marcus Meisel, Stefan Wilker, and Thilo Sauter. Hyperledger fabric smart grid communication testbed on raspberry pi arm architecture. In *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, 2019.

[17] R. Goyat, G. Kumar, M. Alazab, M. Conti, M. K. Rai, R. Thomas, R. Saha, and T. H. Kim. Blockchain-based data storage with privacy and authentication in internet of things. *IEEE Internet of Things Journal*, 9(16):14203–14215, 2022. Export Date: 30 January 2023; Cited By: 11.

[18] Hyperledger. [Online; accessed June 2023].

[19] IOTA. IOTA. https://blog.iota.org/iota-shimmer-assembly/, 2023. [Online; accessed June 2023].

[20] IOTA. IOTA Org. https://www.iota.org/, 2023. [Online; accessed June 2023].

[21] IoTeX. IoTeX. https://iotex.io/, 2023. [Online; accessed June 2023].

[22] Markus Jakobsson and Ari Juels. *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*, pages 258–272. Springer US, Boston, MA, 1999.

[23] K. Košt'ál, P. Helebrandt, M. Belluš, M. Ries, and I. Kotuliak. Management and monitoring of iot devices using blockchain. *Sensors (Switzerland)*, 19(4), 2019.

[24] Ralph Langner. To kill a centrifuge a technical analysis of what stuxnet's creators tried to achieve. 2013.

[25] Lightning Network. Lightning Network. https://lightning.network/, 2023. [Online; accessed June 2023].

[26] Y.K. Liu, S.K. Ong, and A.Y.C. Nee. State-of-the-art survey on digital twin implementations. *Advances in Manufacturing*, 10(1), 2022. Cited by: 13.

[27] Konstantin Makarchev. Benchmarks of different languages. https://github.com/kostya/benchmarks, 2014. [Online; accessed June 2023].

[28] Somayeh Malakuti and Sten Grüner. Architectural aspects of digital twins in iiot systems. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ECSA '18, New York, NY, USA, 2018. Association for Computing Machinery.

[29] Suzana Maranhão, Jean-Marc Seigneur, and Ruifeng Hu. Towards a standard to assess blockchain & other dlt platforms. Technical report, ITU, 2019. ID: unige:112558.

[30] L. Marchesi, M. Marchesi, R. Tonelli, and M. I. Lunesu. A blockchain architecture for industrial applications. *Blockchain: Research and Applications*, 3(4), 2022.

[31] Daniela Mechkaroska, Vesna Dimitrova, and Aleksandra Popovska-Mitrovikj. Analysis of the possibilities for improvement of blockchain technology. pages 1–4, 11 2018.

[32] Mark Michaelis. Preparing for the exponential technology revolution. *MSDN Magazine*, 2019. Available at https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/november/exponential-technologies-preparing-for-the-exponential-technology-revolution last accessed in December 2022.

[33] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

[34] Kai Peng, Meijun Li, Haojun Huang, Chen Wang, Shaohua Wan, and Kim-Kwang Raymond Choo. Security challenges and opportunities for smart contracts in internet of things: A survey. *IEEE Internet of Things Journal*, 8(15):12004–12020, 2021.

[35] Eliseu Pereira, Joao Reis, and Gil Gonçalves. Dinasore: A dynamic intelligent reconfiguration tool for cyber-physical production systems. In *Eclipse Conference on Security, Artificial Intelligence, and Modeling for the Next Generation Internet of Things (Eclipse SAM IoT)*, pages 63–71, 2020.

[36] R. Pinto, G. Gonçalves, J. Delsing, and E. Tovar. Enabling data-driven anomaly detection by design in cyber-physical production systems. *Cybersecurity*, 5(1), 2022.

[37] Serguei Yu. Popov. The tangle. 2015.

[38] Benedikt Putz, Marietheres Dietz, Philip Empl, and Günther Pernul. Ethertwin: Blockchain-based secure digital twin information management. *Information Processing & Management*, 58(1):102425, 2021.

[39] Ripple. Ripple XRP. https://ripple.com/xrp/, 2023. [Online; accessed June 2023].

[40] Matthew N. O. Sadiku, Yonghui Wang, Suxia Cui, and Sarhan M. Musa. The industrial internet of things. *International Journal of Advances in Scientific Research and Engineering (IJASRE), ISSN:2454-8006, DOI: 10.31695/IJASRE*, 3(11):1–5, Dec. 2017.

[41] C. Sailaja. Industrial internet of things – an overview. *Journal of IoT in Social, Mobile, Analytics, and Cloud*, 4(4):257–271, 2022.

[42] Mehrdad Salimitari, Mainak Chatterjee, and Yaser P. Fallah. A survey on consensus methods in blockchain for resource-constrained iot networks. *Internet Things*, 11:100212, 2020.

[43] Brian A. Scriber. A framework for determining blockchain applicability. *IEEE Software*, 35(4):70–77, 2018.

[44] A. Singh, G. Kumar, R. Saha, M. Conti, M. Alazab, and R. Thomas. A survey and taxonomy of consensus protocols for blockchains. *Journal of Systems Architecture*, 127, 2022.

[45] A. Stanciu. Blockchain based distributed control system for edge computing. In *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, pages 667–671.

[46] Swarm team. Swarm. *https://www.ethswarm.org/swarm-whitepaper.pdf*, 2021.

[47] Visa. Visa Fact Sheet. https://www.visa.co.uk/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf, 2023. [Online; accessed June 2023].

[48] Valeriy Vyatkin. The iec 61499 standard and its semantics. *Industrial Electronics Magazine, IEEE*, 3:40 – 48, 01 2010.

[49] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[50] Xiwei Xu, Cesare Pautasso, Liming Zhu, Qinghua Lu, and Ingo Weber. A pattern collection for blockchain-based applications. In *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, EuroPLoP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[51] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

[52] I. Yaqoob, K. Salah, M. Uddin, R. Jayaraman, M. Omar, and M. Imran. Blockchain for digital twins: Recent advances and future research challenges. *IEEE Network*, 34(5):290–298, 2020.

[53] Dr. Alois Zoitl. Iec 61499 the new standard in automation. [Online; accessed June 2023].