Informatics

BIM data management using smart contracts in facility management

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Kreshnik Zuberi, Bsc
Matrikelnummer 01228681

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Thomas Preindl, Bsc

Wien, 5. März 2021

Kreshnik Zuberi	Wolfgang Kastner

Informatics

BIM data management using smart contracts in facility management

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Kreshnik Zuberi, Bsc
Registration Number 01228681

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof.Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Thomas Preindl, Bsc

Vienna, 5th March, 2021

Kreshnik Zuberi Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Kreshnik Zuberi, Bsc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 5. März 2021

Kreshnik Zuberi

Acknowledgements

First and foremost, I would like to sincerely thank my adviser, Prof. Wolfgang Kastner, for giving me the opportunity to work on my thesis within Automation Systems Group. Together with my co-adviser, Thomas Preindl, they provided me with valuable guidance and feedback throughout the process. I thank also the rest of my Automation Systems Group colleagues, who shared their knowledge and experience with me.

I wish to thank my friends and study colleagues, Ardit, Peter, Martin, Eugene, and Ardian. The time we spend together during our studies was very important to me and surely helped a lot when the course load was overwhelming.

I owe a debt of gratitude to my immediate and extended family. My sister, Kaltrina, who was one of the reasons why I decided to enroll at the Vienna University of Technology, also provided unconditional support, especially during my studies. My large group of cousins, among which I would like to mention Kushtrim, Leandra, Rudina, Veton, and Visar, even though spread around the world, gave me motivation by example and lifted me up when I felt down.

Lastly, and certainly not least, I will forever be grateful to my parents. My dad, Irfan, to this day, still supports all my undertakings. My mother, Lumturije, who sadly passed away only a few months before I began working on the thesis, will forever be the example that I follow. She was the biggest supporter of my education and every day I strive to be a person that would have made her proud. I dedicate this thesis to my mother.

Kurzfassung

Die Einführung der Blockchain-Technologie hat das Interesse von Akteuren und Wissenschaftlern in der Architektur-, Ingenieur- und Konstruktionsbranche geweckt. Die Herausforderungen, mit denen diese Branche konfrontiert ist, können teilweise durch die Verwendung des unveränderlichen Ledgers der Blockchain bewältigt werden, der ein sehr hohes Maß an Vertrauen in die Daten bietet. Diese Arbeit zielt darauf ab, die Lücke zwischen der theoretischen Forschung und den technischen Implementierungen im Kontext des Facility Managements zu schließen. Das vorgestellte Softwaresystem dient als Proof-of-Concept für die Änderungsverfolgung und -verwaltung im Facility Management, das eine dezentrale Architektur verwendet. Es speichert die bei den Facility-Management-Prozessen anfallenden Daten in einem Peer-to-Peer-Dateisystem und den Herkunftsnachweis der Daten in der Blockchain. Darüber hinaus werden die Daten mit BIM-Elementen verknüpft, die Teil des Gebäudemodells sind. Ein Smart Contract wird verwendet, um die Business-Logik zu kodieren und Prüfungen auf die Gültigkeit der Daten durchzuführen. Das Ledger der Blockchain speichert die historischen Daten dauerhaft und unveränderlich und ermöglicht so die Nachvollziehbarkeit der Daten. Die Ergebnisse zeigen die Vorteile eines solchen Blockchain-basierten Systems, das eine dezentrale Architektur im Rahmen des Facility Managements nutzt.

Abstract

The introduction of blockchain technology has sparked interest from different stakeholders from industry and academia of the Architecture, Engineering, and Construction domain. The challenges that this industry faces can be partly tackled by using blockchain's immutable ledger, which provides a very high degree of trust on the data. This thesis aims to narrow the gap between the theoretical research and technical implementations in the context of facility management. The presented software system serves as a proof-of-concept for change tracking and management in facility management that uses a decentralized architecture. It stores the data generated during the facility management processes in a peer-to-peer file system and the proof of origin of the data in the blockchain. In addition, the data is linked to BIM elements that are part of the building model. A smart contract is used to encode the business logic and perform checks on the validity of the data. The blockchain's ledger stores permanently and immutably the historical data and therefore facilitates data traceability. The results show the benefits of such a blockchain-based system that utilizes a decentralised architecture in the facility management environment.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The Architecture, Engineering, and Construction (AEC) Industry is a sector of the construction industry that handles architectural and engineering design and construction services. According to reports, performance inefficiencies and a significant carbon footprint make the AEC Industry a good example where a higher technology utilization can lead to better performance, next to financial and environmental benefits. Research on the subject has identified a few such enabling technologies. Among them are Building Information Modelling (BIM) and data exchange and management over the Internet [GZZS10, HKJ+18, For16, ACS16, BWM+17, LLS+15].

Building Information Modelling (BIM) is regarded as one of the most impactful and promising technologies in the sector. Geometric data of the building elements are combined with their properties to create a virtual model of the building, also referred to as a digital twin of the building. The model additionally contains relationships between the elements, their functional characteristics, and even project schedules, which makes BIM benefitial to the stakeholders involved with the building construction and maintenance. BIM and the different software tools that use BIM have gained popularity because they promise improvements in the level of collaboration and coordination and thus increase the performance of the stakeholders and the building itself throughout its life-cycle [SKT12, Azh11, NR19b].

During the last decades, the spread of computing platforms and broadband Internet has allowed for the development of data exchange and management technologies. Nowadays, the Internet connects the world in ways that were unimaginable only decades ago. What is more, the complex infrastructure built to maintain it is constantly improving. The data exchanged over the Internet has become a valuable asset to whoever possesses it, and the need to securely and efficiently manage it has never been more important than it is now. This holds for the AEC industry just as much, since BIM uses digital building data.

In his book, Crawford identifies multiple life-cycle stages that each building goes through: raw material extraction, manufacturing, construction (which includes planning and design), operation and maintenance, and end-of-life (demolition, disposal, reuse, and recycling). By their nature, all these stages require a great amount of collaboration. For example, during planning and design, multiple stakeholders work together to put the project into place. In the construction phase, contractors and subcontractors require coordination and collaboration, where they work closely with the other stakeholders, especially when issues and challenges arise. On the operation and management phase, different stakeholders, such as building inhabitants, owners, facility managers, and technicians interact with the building and its systems [Cra11, OAYY16].

1.2 Problem Statement

The last few decades have seen rapid advancements in the underlying technologies that facilitate collaboration. Numerous studies, reports, and expert interviews claim that higher usage of information and communication technologies (ICT) could increase the rate of digitally assisted collaboration in all of the above mentioned phases. However, according to the studies, in comparison to many other sectors, the AEC industry seems to have a lower technology adaption rate. The reasons for the limited ICT embeddedness are multi-fold. The managerial and financial reasons are not discussed further in this thesis. In what follows, some technical challenges specific to this industry are presented [LLS$^+$15, ACS16].

The standards to represent BIM data, such as the IFC file format specification, have been adopted by various BIM application developers. Nowadays, such applications offer advanced collaborative and productivity capabilities to their users. However, closed ecosystems and interoperability issues have had a negative effect on the collaboration. These challenges are especially prevalent when taking into account the whole life-cycle of the building, during which the large number of stakeholders make for a complex and fragmented environment. Presently, these applications and standards have not yet fulfiled the collaboration requirements of the industry. As a consequence, there is data loss and duplication of work, especially after the transition from one phase to the next [BPRR17, Yal17, HF20, GL10, HD18, GL10, Bar15, LGK19, SEPS19, AP18].

In light of these issues, the work presented in this paper limits itself to the operation and maintenance stage, which is often referred to as facility or facilities management (FM). This way, the focus can be placed on the relevant computer science aspects, while simultaneously leaving out of scope some industry-wide institutional problems, such as the IFC file format specification. This phase begins upon building handover from the owner to the building user and includes all processes and tasks related to the operation and maintenance of the building. It spans for a long period of time, up to a few decades, and this brings the need for structured and efficient management of the data generated during the phase. Moreover, buildings requiring facility management processes usually have a large number of stakeholders interacting with the building in a variety of ways and often in a decentralized manner [Yal17].

According to multiple publications, some concerns with centralization are hindering the deployment of cloud infrastructure and services in the AEC industry. This is occuring despite the trend showing that the percentage of enterprises in Europe using cloud services has been increasing yearly. A cloud-based centralized approach raises questions regarding the security and privacy of the data. In addition, good cyber security is not easy to achieve and maintain in such a highly collaborative and decentralized environment. Improper handling and unclear ownership of the data can result in negative consequences, such as intellectual property loss and legal disputes. Negative consequences can stem also from a possible malicious third party not involved in the building or the lack of trust between the many parties involved in the building operation. The dynamic nature of the building life-cycle phases, including facilities management, means that not every stakeholder knows or trusts the other stakeholders [KS18, RHAW12, GTG+17, TK17, SFM+20, MMB13, LNVD11].

1.3 Aim of the work

The challenges described above related to data security, data retention, and trust have received attention in academic research lately. One of the reasons is the introduction of distributed ledger technology, also known as blockchain. Blockchain has a few properties that appeal to the actors in the AEC industry. The permanent immutable ledger, that stores each transaction, despite what entity initiated it, provides data provenance. The term data provenance refers to the origin and history of the data. The blockchain consensus algorithms validate the transactions and provide a high degree of trust among the participating nodes, even though each node is independent and does not know the other ones. Additionally, computer programs running on these blockchain nodes, known as smart contracts, provide a way to encode different functions in order to automate tasks or remove the human element from many processes. Many use cases and ideas have been presented on different research papers. There are studies claiming that Internet-of-Things devices can be integrated into such blockchain-based systems, with the potential of improving the security of these data, as well [LGK19, TK17, DGPS+20, LKCB19].

An important aspect that needs to be taken into account in any blockchain-based system is that blockchains are not necessarily suitable to store large amounts of data. In fact, in some implementations, the data that they store is very minimal and storing large amounts of data for a long period of time may become costly. Meanwhile, the building information models can potentially become very large data sets, especially when taking into account the data generated during the usage of the building. There are different ways the data can be stored, including centralized, cloud based approaches, or more unconventional ideas involving distributed file systems. The latter is proving to be an interesting research subject [TK17, NSA+19, DLJ19, SwF19].

1.4 Research Questions

Based on the conducted literature review, most of the studies regarding the application of blockchain in the AEC industry have been made from experts in the field of AEC. There still seems to be a gap between the academic theoretical research and actual technical implementations. In other words, the interest of the industry experts in these technologies has not been quite matched by the blockchain developers [LJOD19, Yal17].

The work performed for this thesis tries to dive a bit deeper into the underlying technologies. Instead of trying to remediate the challenges described above, the goal here is to start from the requirements and follow a more structured methodology to present a proof-of-concept. This approach revolves around the idea of decentralized architectures and the benefits that they carry. On that note, the following research questions are presented:

1. How can distributed ledger technology (blockchain) and smart contracts be used for change tracking and management in facility management?
2. How can the data generated during the facility management processes be joined with the building information model (BIM) on a blockchain-based environment?

The expected result is thus the presentation of a method to use blockchain and smart contracts in the facility management environment. One of the challenges described above that the industry experiences is the improper data handling and data loss during handover between the stages. As a result, consideration will be given to storing and handling data in a way that minimizes data loss between these stages. The applicability of this method will be presented via a proof-of-concept, which will consist of a description of its components and how they integrate with each other. The proof-of-concept will showcase how blockchain, a new technology, can be used in the facility management environment. Finally, a discussion will reveal how exactly the research questions are answered and will look at the proof-of-concept with a critical eye.

1.5 Thesis structure

The rest of the thesis is structured as follows. Chapter 2 presents some theoretical background, followed by a literature review. Chapter 3 presents the methodological approach that was used to tackle the research questions and explains the structure of the chapters following that one. Next, in Chapter 4, the main results of the work done for this thesis are presented, including an explanation of the proof-of-concept and some details regarding the experimentation part. This is followed by Chapter 5, which presents a discussion of the main results. This includes an evaluation and analysis of the results from Chapter 4. Finally, Chapter 6 presents some final thoughts on the usage of blockchain in facility management. The chapter ends with a section on possible future directions and concludes the thesis.

CHAPTER 2

Theoretical background

2.1 Overview

This chapter presents the technologies and tools used on the thesis. The goal is to present the reader with enough information to allow for a clear understanding of the work done for this thesis. The user should, however, not expect a thorough description of these technologies, as the scope of each of them is broader than their usage in the thesis. The chapter ends with the a section on related work, which presents some more recent papers mainly about the integration of blockchain and facility management.

2.2 Blockchains and Wallets

Blockchain is specific type of distributed ledger technology (DLT). A ledger[1] is defined as "a book containing accounts to which debits and credits are posted from books of original entry". DLT is simply a distributed database containing such records. Among different implementations of blockchains, one particular application was able to muster users because it combined different computer science inventions into a novel way with the goal of making financial transaction without the need for a trusted third party possible. The application in reference is Bitcoin [Nak09].

Bitcoin

The main contribution of the Bitcoin paper is the introduction of a payment system that is based on cryptographic proof, which eliminates the need to place trust on any other entity. This system is peer-to-peer, so the transactions are done directly between the two parties involved. One of the biggest problems that this system had to solve

[1]Ledger. Accessed on 31.12.2020. In Merriam-Webster's Dictionary. https://www.merriam-webster.com/dictionary/ledger

was the double-spending problem, which is simply the risk that any digital currency, because of its digital nature, can be easily copied and thus spent twice. To prevent the double-spending problem, each entity that places transactions must be aware of all previous transactions. In a peer-to-peer system with no trusted third party, this means that all transactions must be publicly announced. Additionally, each entity must agree on a single history of past transactions and the order in which they were created. This single history is the history to which the majority of the entities have agreed to [Nak09].

The process of agreement is determined by what is referred to as a consensus protocol. Bitcoin's consensus is called proof-of-work. In simple terms, it means that before a transaction is added to the chain of transactions, a sufficiently difficult computational challenge must be solved. In this case, it is the scanning of a string that with relation to a hashing function will yield a hash starting with a certain number of zeroes, a process referred to as mining. Since the duration of the mining process is proportional to the computational power of the computer node performing it, the difficulty of the challenge can be altered by changing the number of zeroes required in the hash. As an incentive for the nodes to perform mining, a certain amount of digital currency (coins) is rewarded to the nodes that successfully mine transactions. This is analogous to miners that expend resources to add minerals into circulation [Nak09].

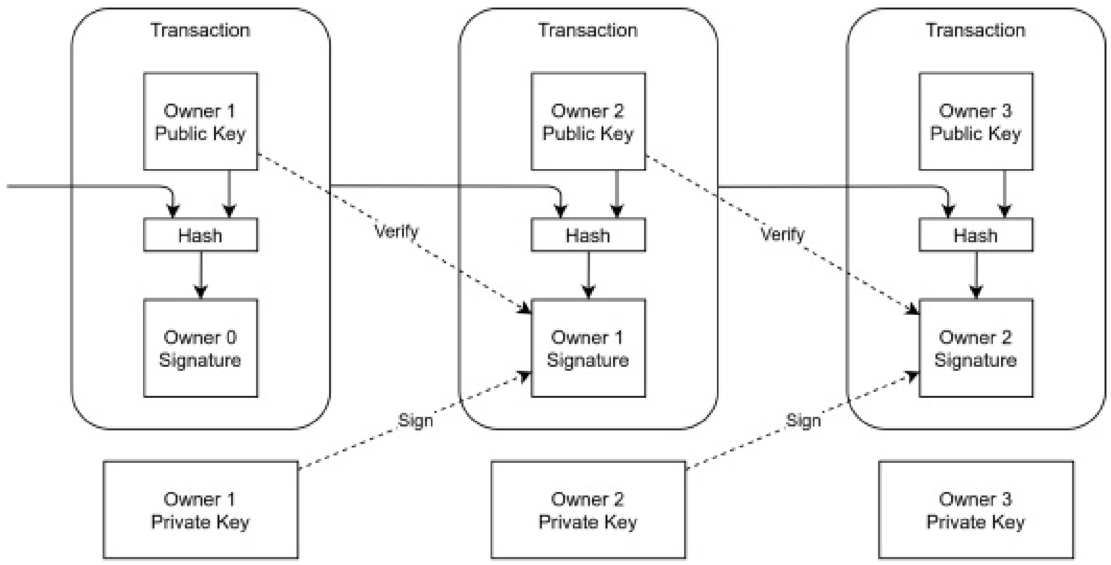

Figure 2.1: Bitcoin Transactions [Nak09]

The Bitcoin paper defines an electronic coin as a chain of digital signatures. The coin is transferred between entities by using public-key cryptography to sign the hash of the previous transaction and the public key of the receiver with the payer's private key, as shown in Figure 2.1. This way, the receiver of the transaction (payee) can verify the signature and the last transaction in this chain of transactions. Each transaction is broadcast in the network. When a computer node receives transactions, it puts a few of them together to form blocks. Once a block is constructed, the node tries to find the proof-of-work for that block and when it does so, it broadcasts this block to all other nodes. The receiving nodes accept the blocks that have valid transactions, meaning transactions that have not already been processed. Once a block is accepted, its hash is included in the next block consisting of new transactions, forming what is known as a Merkle Tree[2] [Nak09].

The first transaction in a bitcoin block is a special transaction that creates a new coin, which is owned by the creator of the block. In other words, the block miners get rewarded with coins. Another reward for the miners is the transaction fee, which is a price that every payer has to pay to the miner for processing the transaction. Both these rewards provide strong incentive to the miners to support the network [Nak09].

An important property of this algorithm is that it is tolerant to the Byzantine's General Problem, which shows that in a distributed system consisting of nodes that send messages, it is hard to know if a message received from a node is faulty or malicious. In Bitcoin, this tolerance comes from the fact that when the nodes accept new blocks, they always consider the longest chain of blocks as the correct one. As long as honest and correct nodes are a majority in the network, the longest chain of blocks can be considered as the correct one. In addition, because of the difficulty of the proof-of-work consensus protocol, it becomes harder for any attacker to change past transactions, as doing so would require changing all the transactions added after the changed transaction and redoing the proof-of-work for each block [Nak09].

The structuring of the blocks into a connected chain has led to the term blockchain. In the years following the introduction of Bitcoin, other implementations have been introduced that are based on Bitcoin's ideas but vary in different ways. This includes different consensus protocols, the use of fungible or non-fungible assets, but also extensions of the network with additional features. One such example is the Ethereum network [B$^+$14].

Ethereum

Ethereum is a blockchain with a built-in Turing-complete programming language. Each node running the Ethereum blockchain can be used to run custom-written pieces of code, referred to as smart contracts. The smart contract code is executed in every node that receives the block containing the transaction that triggered the code execution. Even though the smart contracts run individually on nodes, their state is agreed upon by all the nodes in the network, using the proof-of-work consensus protocol. Therefore, the Ethereum network is a distributed computer, which is called the Ethereum Virtual

[2]In computer science, a tree is a graph that contains one root node and a set of children nodes, whose parent is the root node. Each child is a subtree, so may have its own children. If a node does not have any children, it is called a leaf node. In cryptography, a Merkle Tree, also known as a hash tree, is a tree where every leaf node contains a hash, and every non-leaf node contains the hash of its child nodes' hashes. Merkle Trees are interesting data structures because the hash of the root node is dependent on the hashes of all of its subtrees. If any one of the hashes changes, then it is computationally very easy to check that the hash of the root is not valid.

Machine (EVM). The applications that use smart contracts are called distributed applications (dApps). Smart contracts have their own storage, which means that they can store data in the blockchain [Eth21c, B$^+$14].

In Ethereum, transactions are defined as "cryptographically signed instructions from accounts". Transactions may transfer coins between two accounts or call smart contract functions. To safeguard against wasteful exploitation of computational resources, Ethereum uses the concept of gas, which puts a price on every unit of computation, bandwidth, and storage that the transactions or the smart contract code execution consume. The price is measured in Ether, which is Ethereum's coin. Analogous to Bitcoin, nodes performing mining are rewarded with Ether [Eth21f, B$^+$14].

Ethereum uses two kinds of accounts, both identified by unique addresses: externally owned accounts and contract accounts. The externally owned accounts are controlled by private keys and can create and sign transactions, so they can be used by human agents. Contract accounts are computer programs controlled by their own code. They only react after they have been transacted by another account. Their code runs autonomously if it is triggered by a transaction. When an externally owned account initiates a transaction, if the state of the EVM is changed, then the transaction is broadcast to the whole network [Eth21a, B$^+$14].

Blockchain properties

Based on how nodes join the network, blockchains can be permissioneless or permissioned. Both Bitcoin and Ethereum are permissionless blockchains. This means that any node can enter or leave the network without requiring permission. A contrasting type of blockchains are the permissioned blockchains. Permissioned blockchains have an additional layer of security, where before a node joins the network, some method is used to grant or deny access. EVM, if run on a private environment, can be considered as a permissioned blockchain. There are blockchain implementations that are more focused for enterprise use and thus have more privacy features built-in, such as Hyperledger Fabric and Quorum [Eth21b, PRL20].

Bitcoin was designed for use primarily in a financial context. As such, it handles very well the transfer of coins between two parties in a peer-to-peer manner, eliminating the need for a central authority. However, since all transactions are public, anyone can see the transaction history of every account and thus know their coin balance. When it comes to privacy, this means that the data are as private as the user is able to keep their account. Keeping the account private is not always feasible, because to make a transaction on Bitcoin, the account must be revealed. Even though the accounts are often pseudonyms, they may be de-anonymized by analyzing usage patters. Similarly, on Ethereum, the data is also public. This includes, in addition to the user accounts, also the smart contract accounts and their internal storage values. However, there are other blockchain implementations that place more importance in data privacy, one such example being Zcash[3] [HP17, Eth21b].

Data storage on the blockchain

By definition, a blockchain is a digital ledger. In other words, it is a distributed database of entries. Financial applications usually do not require large data sets, as transactions on a financial setting only contain a limited amount of data, such as addresses and currency amounts. As such, initial blockchain implementations, such as Bitcoin and Ethereum, are not designed for storing large amounts of data. However, many blockchain applications work with large data sets. In this thesis, the data set is the building information model (Section 2.5), which can potentially become very large. Because the network of blockchain nodes replicate the data and store copies of them, in these cases, where the data set is large, this may not be optimal or even desirable. As a result, there are researchers that are considering alternative file systems in parallel to blockchains, while others argue that the data replication is beneficial, in order to provide trust [TK17, NSA$^+$19, DLJ19, SwF19].

Application development on Ethereum

Development on the Ethereum blockchain requires developing in two separate environments. On one side, there is the smart contract development, which is referred to as the back-end. Smart contracts can be written in different programming languages, the most popular being Solidity[4]. Solidity is an object-oriented and high-level language based on C++, Python, and JavaScript. A convenient way to begin building and debugging smart contracts is by using the Ethereum integrated development environment (IDE) Remix[5]. On Remix, code can be written and tested without the need to have a running blockchain, as the IDE handles all what is needed. On the other side is the website development, known as the front-end. The front-end is essentially the end-user application: it provides a user interface and makes calls to the smart contract.

Transacting with the smart contracts has been made fairly straightforward, as well, thanks in part to the different libraries available, which are also known as client APIs. One of them is web3.js[6]. web3.js offers a collection of JavaScript modules enabling functionality for the Ethereum ecosystem. It is used to create an instance of the `web3` object, which, in turn, is used to call smart contract functions. In order to do that, the smart contract code written in Solidity is encoded into an application binary interface (ABI). Once the ABI is loaded into the JavaScript application, it is used as a parameter on the `web3` object functions. For example, to create a `contract` object, the ABI and the smart contract address are needed as parameters. The created `contract` object exposes functions for each of the public smart contract functions. This way, transactions can be sent to the smart contracts running on the Ethereum. The available libraries can also be used to query the blockchain for, among others, block numbers and smart

[3]Zcash. Accessed on 09.02.2021. https://z.cash/
[4]Solidity. Accessed on 25.01.2021. https://docs.soliditylang.org/
[5]Remix. Accessed on 25.01.2021. https://remix.ethereum.org/
[6]web3.js. Accessed on 25.01.2021. https://web3js.readthedocs.io/

contract events. The prerequisites of this process are that there needs to be already a running blockchain to connect to and a deployed smart contract in the blockchain. Running a test blockchain, along with the smart contracts deployment on the blockchain, can be done in multiple ways, one of which is by using Truffle Suite's Ganache[7].

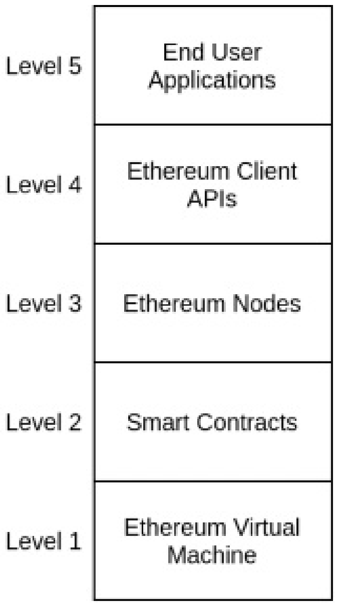

Figure 2.2: The Ethereum Stack [Eth21e]

To provide a complete model, Figure 2.2 shows the Ethereum stack. The lowest level, the EVM, is the runtime environment for smart contracts that also handles all the transaction processing on the Ethereum network. It creates a level of abstraction between the executing code and the executing machine. The next level, smart contracts, are computer programs that compile to EVM low-level machine instructions. The third level, the Ethereum nodes, are the computers that run the Ethereum client, which is an implementation of Ethereum. So, in order for applications to interact with the Ethereum blockchain, they must connect to an Ethereum node. The Ethereum Client APIs, the fourth level of the stack, are simply the (optional) libraries that help developers by making application development more convenient. Finally, the fifth level are the end-user applications, which can perform a variety of actions, just as standard applications [Eth21e].

The smart contract can be seen as services which run non-stop and can be accessed by all members of the blockchain. Any member of the blockchain can deploy smart contracts on the network. However, since deploying a contract is a transaction in itself, there are costs associated with it. Smart contracts are guaranteed to execute in predictable ways, since the data that they work with and their code is publicly available. Other benefits

[7]Ganache. Accessed on 25.01.2021. https://www.trufflesuite.com/ganache

of smart contracts and distributed applications include zero downtime and resistance to censorship. However, there are also some implications. For example, once a smart contract is deployed, it is hard for developers to modify them. The distributed nature also causes a large overhead, as every computation is replicated among all nodes [Eth21d].

Wallets

Wallets, or cryptocurrency wallets, are devices, programs, or services with the primary function of storing cryptographic keys. Public key cryptography mandates the use of a private and public key, so the wallets provide the means to store and use this key in a secure manner. Throughout the years, many cryptocurrency wallets have been developed and their functionality has been extended. For example, in conjunction with blockchains, they are used to encrypt and sign transactions. Since the public and private key pairs are used to create and verify the signatures, their protection against theft or compromise is also dependent on the owner of the wallet [Eth21c, KBS17].

One such crypto-wallet is Metamask. Metamask is a browser plug-in where the user can securely add their blockchain accounts. It stores the accounts on the device and uses different authentication methods to protect them. In addition, Metamask maintains a connection to the blockchain, and can even be used to connect to local blockchains for testing purposes. Once connected, Metamask can be used to make transactions. In addition, if a web site makes a transaction on the blockchain, Metamask can display a pop-up notification to the user, asking for a signature. Metamask is just one of the many crypto-wallets, but it shows some of the functionalities that wallets can offer [Lee19].

2.3 Blockchain-oriented software engineering

Software engineering is an engineering discipline that concerns itself with all aspects of software production. It is intended to support professional software development and defines techniques to support program specification, design, and evolution. The importance of software engineering lies in the fact that advanced software systems have nowadays become part of the society and therefore it is important that they are reliable and trustworthy. Additionally, as software systems age, these software engineering techniques, when properly applied, make their maintenance and further development cheaper. Software engineering approaches vary dramatically depending on the type of software being developed, the organization that is developing it, and the people involved in the process. Therefore, there is no universal software engineering method [Som11].

During recent years, blockchain has not only risen in popularity among computer scientists and researchers, but it has become a buzzword among different industries, as well. The term *Blockchain-oriented Software Engineering* has been coined to capture the main considerations that need to be taken into account when building software that contain blockchains. These engineering practices and techniques add to traditional software engineering additional steps that make sure that the benefits of the blockchains are indeed understood and correctly implemented [WG18a].

In reality, it often happens that the industry experts that are exploring the possibility of using blockchain in their domain are not fully aware how blockchain works and what exactly it represents. In this regard, there are scientists working towards establishing common grounds and a methodology that can be used by anyone to determine if blockchain is indeed needed or suitable for their use case. Wüst and Gervais have devised a simple questionnaire, which with a series of yes or no questions guides the reader if blockchain is needed for their use case. In addition, if the answer is yes, the questionnaire also determines the type of blockchain needed for their use case, permissioned or permissionless [WG18b].

Determining that blockchain is needed for the use case is only the first step in the quest of designing a blockchain-based product. The next steps would be to follow a well structured methodology to engineer a system with blockchain as a building component. Designing software that uses blockchain is considered to be a new approach in software engineering. Such software systems need to be built around the blockchain from the ground-up, because adding blockchain to an already designed system brings about challenges and security risks. Wessling and Gruhn have conducted some research on the subject that aims to provide a methodology and structure how such a process shall look like. Central to this methodology is the identification of the actors that will interact with the system, the trust relations between them, and the interactions that they have with one another. These are then used to derive a high-level architectural view, which will guide the software development process [WG18a].

Analysis of the different ways in which users can interact with blockchains have revealed that applications can be grouped into three different patterns:

- The user generates the transactions
- The user uses some application to self-confirm the transactions
- The back-end generates the transaction on the users' behalf

The first pattern gives more control to the user, however it is not user friendly, because the user will have to generate the transactions, so more technical knowledge may be required. The second brings a better trade off between trust and convenience, because a website can be built that has a simple user interface. In addition, a separate application, called a wallet, is used to confirm the transactions, again providing an easy user experience. Finally, the third method prioritizes user friendliness, while distancing the user from the inner workings of the application and thus lowering the trust degree that the application provides [WG18a].

2.4 Interplanetary file system (IPFS)

The rapid spread of computers and Internet access throughout the world, coupled with the increases in computing and storage resources led to the spread of different file-sharing systems. Soon, developers began to notice that the paradigm of using central servers to host most of the content on the Internet had some drawbacks, especially when the system users and the bandwidth requirements grew. As an alternative and distributed solution, peer-to-peer systems sparked the interest of many users and developers. On this note, the Interplanetary File System (IPFS) is described as a peer-to-peer distributed file system with the ultimate goal of providing one common file system for the Internet. The advantages of peer-to-peer systems can be summarized by the following characteristics: every computer node contributes to the system, the capabilities and responsibilities of the nodes are the same, despite them contributing varying resources, and there is no central component that controls the operation [Ben14].

IPFS should not be described as a separate invention or program, because it is the combination of the result of decades worth of development in different areas of computer science. Distributed Hash Tables (DHTs) are one of the technologies. DHTs, in simple terms, describe a method to efficiently lookup information in a distributed network consisting of arbitrarily many nodes. The participating nodes can join or leave the network without negatively affecting the network, even though there is no central entity necessary to coordinate it. The exchange of data in IPFS borrows ideas from BitTorrent, which uses protocols to incentivise nodes for contributing data and tracks the availability of pieces of data while simultaneously handling priorities, as well. Changes to the data are handled in ways that were made possible by version control systems, such as Git. In Git, files, directories, and changes are addressed by the cryptographic hash of their content, a process known as content-addressing. In a similar way to blockchain's Merkle tree, here links are embedded into the content of the nodes, forming a Merkle tree. That way, data changes are done by adding the additional data and updating the corresponding references. Lastly, a technique called self-certified file systems, that uses public-key cryptography, is used. It hashes the public key and the location of the file together, which allows for the user to verify its validity [Ben14].

In IPFS, each node has its own identity and performs different services, such as data transfer between each other, maintains routing information, and stores content-addressed files and objects. When files are added, they are broken up into blocks and stored in the file system. Each block is hashed, and its cryptographic hash is later used to retrieve the content. If the file is sufficiently large, a tree structure is formed, and the hashes are, in turn, hashed again, resulting in a hash that represents the root of the Merkle tree and which is used to retrieve the whole content of the large files. Many IPFS features can be customized. For example, custom data structures can be created and stored in the IPFS as custom blocks. This means that an application developer can decide how the blocks will be structured and therefore fulfill their custom requirements. As another example, IPFS allows for the creation of private networks of IPFS nodes, thus controlling where the data is stored and replicated and who has access to it [Ben14].

Using IPFS programatically

IPFS can be installed as a desktop application and used to store and retrieve files, along with setting different parameters to customize the installation and the nodes. To build applications that use IPFS programatically, different libraries can be used. IPFS nodes can either be spawn directly from the application, or the application can directly communicate with an already existing IPFS node. For the latter, a convenient library is ipfs-http-client[8], which provides a way to create an instance of the HTTP API client. The object exposes functions to directly perform various operations, such as add, get, and put files or blocks. The application uses the HTTP protocol to communicate with the IPFS node [ipf21].

2.5 Building information modelling (BIM)

Building information modeling is a term that has been used during the last few decades in the construction industry and built environment after the industry began using computers to digitally represent building information. With the evolution of the information and communication technologies, so did the term BIM. At first, the two-dimensional building plans were converted into a digital format. This was soon extended to three-dimensional interactive representations. The benefits of this digitization were so obvious, that functional characteristics were added to these 3D models, as well. During recent years, additional dimensions were added to BIM, such as time, cost, sustainibility, and asset performance [LL20].

Throughout the years, many companies have developed different software tools that are currently used in the industry. These tools offer adanced capabilities and perform their goal very well, but if their users try to exchange data between them, they might encounter issues. Fortunately, there seem to be positive movements in this direction, as in late 2018 ISO published the standard "ISO 19650 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — Information management using building information modelling". This standard defines BIM as "use of a shared digital representation of a built asset to facilitate design, construction and operation processes to form a reliable basis for decisions". Currently, however, the interoperability issues are only partly solved [BBK$^+$18, ISO18b].

Industry Foundation Classes (IFC) standard

The Industry Foundation Classes is an ISO standard for BIM data that are exchanged between different applications. This standard was firstly devised by the buildingSMART organization with the goal of promoting vendor-neutral building data format that facilitates sharing and exchange. It has definitions that cover data regarding the whole building life-cycle. Additionally, it specifies a data schema and an exchange file format structure for exchanging and sharing data according to the schema.

[8]ipfs-http-client. Accessed on 25.01.2021. https://www.npmjs.com/package/ipfs-http-client

One of these formats is the standard ISO 10303-21, known as the STEP-File. This standard presents a comprehensive description how building data should be stored, including the modelling of all building elements, the information regarding their properties, their geometric dimensions and positions, and the relationships between the elements. However, it does not specify how exactly the file is to be constructed. In simple terms, when different applications create an IFC STEP-file from the same BIM model, it can happen that the resulting files have different internal structures and contain inconsistent element identifiers. Even though every element has a Globally Unique Identifier (GUID), the two resulting files might assign different GUIDs of the same element. This makes it difficult to work with such files from a computer science perspective. Despite these difficulties, the IFC standard is accepted as one of the leading standards and is being adopted more and more, as recently there have been governments mandating the use of IFC in all public building projects [ISO18a, ISO16, BBK$^+$18].

2.6 The Persona and Scenario Method

The Persona and Scenario Method is a strategy that helps design software products. Its goal is to identify the typical end users or the target audience of the product and thus describe a scenario from their perspective. This gives the reader a clearer description of the context but also helps the software developers with their tasks. The first step is to present the personas, which are fictional characters. Once some characteristics of each of the personas is given, the next step is to present a scenario. In the scenario, the personas are used to describe a typical activity that involves the software product about to be designed. One of the main advantages of the Persona and Scenario method is that it provides a tool to determine requirements and design solutions [PMC17].

2.7 Software Requirements Specification (SRS)

The term software requirements specification is an industry standard used to define the functions and capabilities that a software product should fulfill. Its main goal is to describe as completely and detailed as possible the functionality, behaviour, and qualities of the software that is being designed. The SRS is documented prior to the planning, design, and coding phases. It is usually structured as a template and contains different sections or categories to make for a cohesive document for the different project stakeholders that rely on it, including customers, project managers, software developers, legal teams, etc. The SRS is an important step in any software design, because it is difficult to properly design a software product without an exact description of the requirements [WB13].

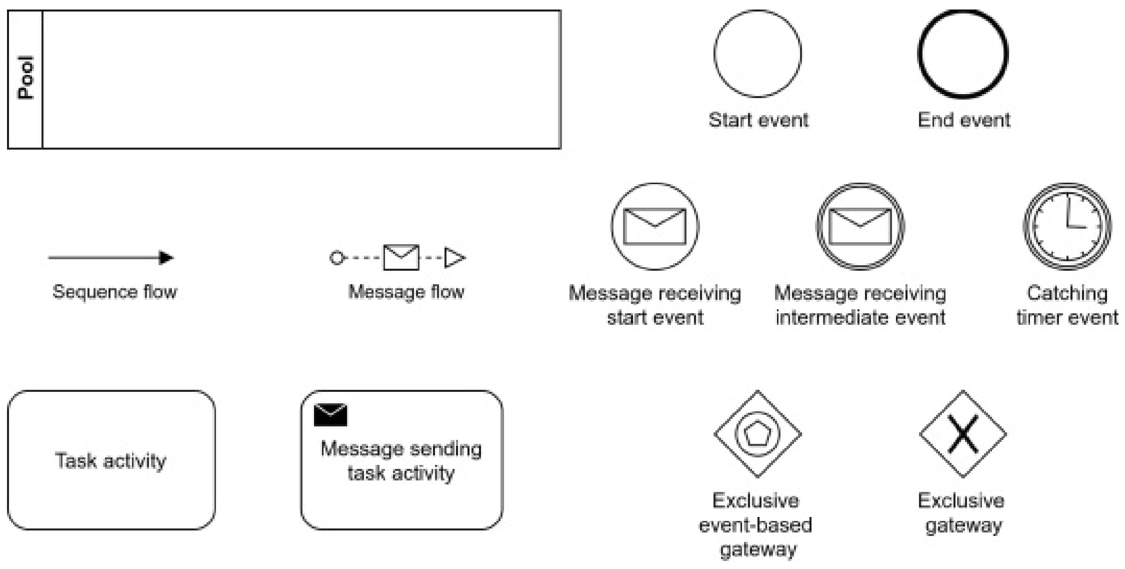
Pool
Start event
End event
Sequence flow
Message flow
Message receiving start event
Message receiving intermediate event
Catching timer event
Task activity
Message sending task activity
Exclusive event-based gateway
Exclusive gateway

Figure 2.3: BPMN elements

2.8 Business Process Modeling and Notation (BPMN)

Business Process Modeling and Notation is a standard developed with the goal of creating a notation to describe business processes in a understandable way for all stakeholders. This includes the business analysts who create the first process drafts, the developers who engineer the product, and the management personnel responsible for process monitoring. The products described by the BPMN standard are diagrams and models. The BPMN diagrams are graphical representations of the processes based on a flow-charting technique. The models support the generation of executable code. There are tools and research being done to automatically generate smart contracts based on BPMN models. In this thesis, there are two BPMN diagrams. In the following, the objects contained in these diagrams are described [Whi04].

The BPMN diagrams consist of pools for each participant in the business process. Inside the pools, there are flow elements and connecting objects. The flow elements are events, activities, or gateways. The connecting objects are sequence flows or message flows. Figure 2.3 lists the flow elements and connecting objects that are used in this thesis. Every process must have a start and end event. Events represent something that happens and usually have a trigger or result. There are different types of events, based on when they happen. Activities represent a work or task that is done. Similarly to events, there are different types of activities, based on whether they are performed by a specific user or a system, or if they send messages. Gateways control the splitting and merging and model decisions that are taken by the system. Sequence flows show the order and flow of the activities, while message flows show the flow of messages between different elements. The BPMN standard contains other elements, types, and rules, which are not used here and therefore not discussed further [Whi04].

2.9 Unified Modeling Language (UML)

The Unified Modeling Language is a family of graphical notations that are used to describe and design software products. It was designed to be used not only from developers, but also from other project stakeholders. UML defines a few diagram types, each with a different purpose. In this thesis, two of these diagrams are used, the use case diagram and the sequence diagram and therefore these two are described next [Fow04].

Use case diagrams

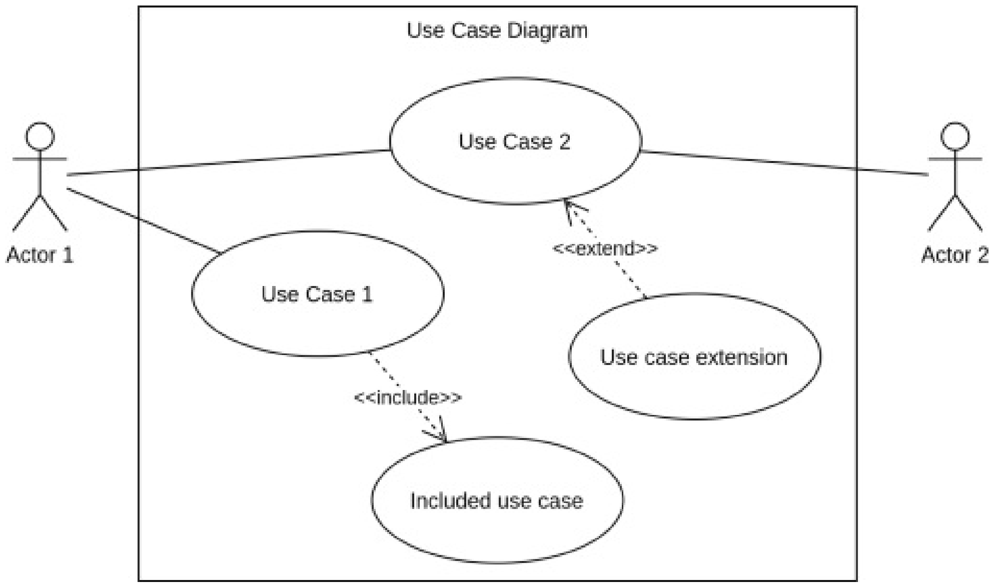

Figure 2.4: Use case diagram

Use cases are used to present the functional requirements of a system. They show the typical interactions between the users of the system and the system components. Every use case encapsulates an activity that one or more users perform using the system. These activities are grouped together into a diagram, called the use case diagram, as shown in Figure 2.4. The use case diagram consists of the system boundary, represented with the box. The actors lie outside of the system boundary and represent either human agents that interact with the system or the components that are part of the system. Every bubble inside the system boundary represents a use case, which is carried out by one or more actors. Use cases can have relationships between them, represented by arrows between the use case bubbles. The «include» relationship arrow points to use case that is included in the originating use case and this means that the use case is carried out every time the originating use case is carried out. In contrast, the «extend» relationship arrow point to a use case that the originating use case extends, which means that they are carried out only in some cases. Use case diagrams do not describe in detail the activities, but they provide a clear overview of a scenario. Thus, they are used as stepping stones during software development [Fow04].

Sequence diagrams

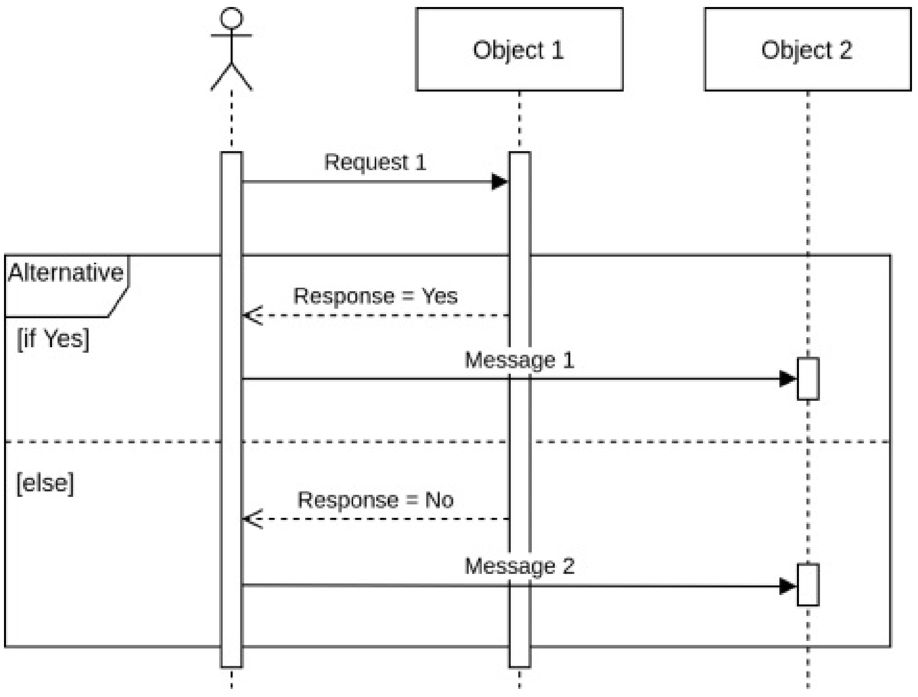

Figure 2.5: Sequence diagram

Use case diagrams are often followed by sequence diagrams. Each use case can be extended into a sequence diagram. These diagrams decompose the use cases into a sequence of activities. Here, the order of the activities has vital importance, because, in a sequence diagram, there is a flow of events between the actors and the system components. This flow is shown by the messages that are sent between the actors and components. The resulting diagram represents a more technical view of the use cases because they can define exactly what each component does in any given use case. As can be seen in Figure 2.5, on top of the diagram there are the actors and objects, which represent the system components, and the dashed lines below them are called the lifelines. The activation bar, which is the vertical bar below the actors and objects, shows the period during which each component is active. Message requests are represented with a line arrow, while responses with a dashed line arrow. There is also the possibility to show alternatives, in cases where responses can have different values. Such alternatives are shown in the alternative boxes [Fow04].

2.10 Related Work

The literature references made in the introduction chapter reveal that there is indeed plenty of interest in the use of blockchain and smart contracts in the facility management sector, especially from experts and academics of the AEC Industry. Nawari and Ravindran provide examples where these technologies have the potential to improve the collaboration between the stakeholders, bringing increases in complience and trust. Other benefits include preservation of data ownership and modification history. In addition to the usage of smart contract for the automation of business processes, Giuseppe et al. argue that automatic payments using smart contracts would respect the requests of each stakeholder [NR19c, NR19a, DGPS+20].

Turk and Klinc point out the importance of proper management of legally significant information in the AECOO industry. For example, what entity did the BIM modification and what was its timestamp can prove to be very valuable information on later stages of the project. Furthermore, four different scenarios are described how building information models can be combined with the blockchain or stored on it. Among them, according to the authors, the proper way would be to separate the large data files containing the building information model from the operations done on the model. For example, a BIM server will store the BIM data while the transaction data will be stored on the blockchain [TK17].

With arguments against centralization, Nizamuddin et al. provide an alternative perspective, where the BIM data can be stored in a decentralized file system. Since current blockchain implementations are inefficient at storing large files, the InterPlanetary File System is used. This runs in parallel to the blockchain and stores the files in a decentralized and secure manner, while providing a hash value that points to the file. The hash is stored in the blockchain, linking both components of the system together. Additionally, the authors propose a version control system. Every time a stakeholder submits a change to a file, a smart contracts handles the approval process. The smart contract only stores the hash of the new files if the they are approved and it keeps a record of all the different versions of the files [NSA+19].

Smart contracts in combination with IPFS are also explored by Dounas et al. Here, a collaboration mechanism was described that allows participants to submit design solutions to problems raised by a stakeholder. The design optimisation performance is used as a consensus mechanism. Again, BIM data is stored on a distributed file system, while the hash corresponding to the data is stored on the blockchain. Smart contracts are used to compare the most optimal solution against a predefined threshold. Once a new optimum has been established, the smart contract updates the state of each node to the current optimum [DLJ19].

Peherstorfer explored the integration of BIM and blockchain using a smart contract to manage changes to building information models. The architecture of the developed prototype consists of a smart contract back-end, a Windows desktop application front-end developed in C#, and GitHub[9] used to store the BIM data. The front-end is used by the stakeholders to submit a proposed change to the building model. This change request is then approved or denied by a certain stakeholder, named as the construction manager. If it is approved, the construction manager further selects a number of other stakeholders which will vote to either accept or reject the change request. If at least one of the stakeholders rejects it, then the change is not saved. In the developed prototype, this logic is handled by an Ethereum smart contract. The architecture of the prototype is unique in that it uses GitHub to store potentially large amounds of data, which might not be an efficient use case for GitHub [Peh19].

Li et al. are looking into the possibilities of implementing specific facility management requirements into the smart contract and thus using the smart contract for inspection services. With the use a different IoT sensors and data processing techniques, certain physical properties of the building can be detected and then checked against some parameters encoded in smart contracts. The benefits of this approach are the elimination of the human element from the inspection and verification steps, the automatic processing of payments in case the task requirements are met, and the use of the immutable ledger of the blockchain for traceability features. Nevertheless, the authors acknowledge that the smart contract developed for this paper is very small in scope, so it would require thousands of such smart contracts to cover the whole building [LKCB19].

Building upon this work, in a more forward-thinking conference publication, Li et al. describe a high-level system architecture where actors, such facility managers and contractors, interact with different automated systems that store data regarding products and legal requirements and even automate the tendering process. All this sits on top of a transaction processor based on a blockchain, which is described as a ledger storing immutable and chronological proof for the chain of activities performed during the facility management phase. This paper describes a scenario where the maintenance activities are automatically triggered by one or more of the system components and are managed automatically by a decentralized autonomous organization (DAO). A DAO is an organization formed from a collection of smart contracts that manage every aspect of the organization, eliminating the need for a human managing entity. While both of these papers provide interesting future prospects, it should be noted that they acknowledge that the industry and technology are not ready for such developments. In addition, there seems to be no direct mention of the BIM data that the smart contract or DAO manage [LKW20].

Xue and Lu pay attention to the IFC file structure and assesses that any attempt to integrate blockchain with BIM data faces the data problem. Namely, the amount of data contained in a BIM IFC file is very large, and its replication among the blockchain nodes is simply not feasible for any real world scenario, where a lot of data sharing and collaboration is involved. The paper takes into account the problems with the IFC file format specification and presents an algorithm that, according to the paper, manages to

[9]GitHub. Accessed on 18.02.2021. https://github.com/

find the difference between any two IFC files. This way, only this difference between the models can be stored in the blockchain's distributed ledger and thus the amount of data is greatly minimized. The paper seems to present an interesting and efficient way to find and separate the building model changes from the building model file. However, it stops short of an actual smart contract implementation. Instead, an online tool is used where the data is inserted into a visual representation of a blockchain, which demonstrates how blocks are mined [XL20].

Tran et al. attempt to streamline and automate the generation of smart contract code. By using model driven engineering, a tool is presented that generates smart contract code from the business processes described in the Business Process Model and Notation standard. Considering the fact that once a smart contract is deployed it cannot be amended, the advantage of this method is that it provides the means to generate code that is already tested and therefore safe to be deployed in the blockchain [TLW18].

Some interesting ideas can be brought from a seemingly irrelevant field, namely medicine. Azaria et al. describe a modular patient record management system that utilizes blockchain's properties. The prototype described in the paper uses different layers of connected smart contracts to store information for the different entities, such as patients, healthcare providers, or insurance providers. The idea is to have smart contracts that store entity data, other smart contracts that store the interaction between the entities, and a third layer of smart contracts that store the permissions for each of these interactions or medical records. The data is stored off-chain, which means that the data storage solution and providing data security is left out of scope. During data access operations, before any confidential data is looked-up or revealed, the component described as the database gatekeeper confirms the identities via cryptographic signatures and checks the smart contracts to verify the address and permission [AEVL16].

CHAPTER 3

Methodology

3.1 Overview

This chapter presents and describes the methodology that was used to carry out the work for this thesis. As presented in the introduction (Section 1.4), one of the expected results of this thesis is the creation of a proof-of-concept. The methodology chapter consists of a few sections. The case study approach describes the first step, which is to present and formalise a use case. Next, the requirements analysis describes the steps taken to deduce requirements from the use case. In the experimental approach, the steps to come up with the proof-of-concept are presented. Finally, the evaluation describes the evaluation methodology of the proof-of-concept. Figure 3.1 depicts the methodology in a graphical form.

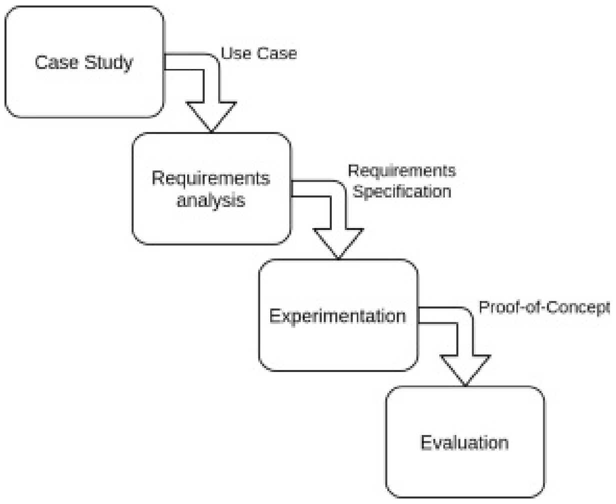

Figure 3.1: Methodology flow diagram

3.2 Case study approach

The case study approach is a methodology often used in research to gain in-depth and multi-faceted understanding of complex issues. The main idea of this approach is the exploration and understanding of some phenomenon in a natural context. In this thesis, the case study approach was used by presenting a use case in the facility management environment and analysing the use case to deduce a set of requirements [SCR+11].

As already mentioned, this thesis is in the setting of facility management. To make sure that this is well understood, an extensive literature review was carried out. The review looked at the AEC Industry practices and tried to focus on facility management. This revealed a research gap and thus the creation of the research questions (Section 1.4). Part of the literature review was also the review of related work (Section 2.10), which focused on computer science aspects to look at the current state of the technologies and presented some outstanding work.

With the goal of maintaining reader-friendliness, some steps were taken to ensure that the language and terminology of this thesis are understandable and do not present a barrier for the reader. To that extent, the Persona and Scenario method (presented in Section 2.6) was used to present a fictional user story. The user story describes a scenario involving different operations performed by the personas and thus presents the reader with a representative business process in the facility management context. To formalize the process, the BMPN notation (presented in Section 2.8) was used, which is a widely adopted method to present business processes in a clear way and can be used as a tool for further development.

As a final step in the use case presentation part, UML, because of its convenient graphical diagrams (as described in Section 2.9), was used to derive a set of use cases and actors. Together, these form the use case diagram. The use case diagram, constructed from the perspective of the personas, shows a straightforward overview of the presented use case and leads the way towards the creation of the next steps, presented next.

3.3 Requirements analysis

The next step after having described the use case was to deduce the system requirements, presented in Section 2.7. The requirements were separated into functional, data, and security requirements. This separation was needed because the development of the proof-of-concept requires the evaluation of the different aspects of the use case. Each requirement set deserves its own analysis and contributes to the proof-of-concept in a different way. The functional requirements specify the exact functionality that the proof-of-concept must contain. The data requirements define which data exactly is needed, from the perspective of the user. Finally, the security requirements specify what security aspects are important for the user and how the proof-of-concept must handle the data.

An important goal of the requirements analysis was to look at the requirements from a computer science perspective in order to determine if blockchain technology is suitable for the given use case. There has been some research done on this subject with the aim of providing a well-structured methodology for developing software which is based on blockchain, also called blockchain-based software engineering (as presented in Section 2.3). These best practices were applied to the use case and requirements analysis and the results of this process are presented.

3.4 Experimental approach

The results of the requirements analysis were used to design a high-level architecture of the proof-of-concept. This meant that all the components, their properties, and how they interact with each other had to be defined and presented, as well. Special attention was given to the data structures, especially how existing building data is combined with the newly generated data. Same as the data requirements, the functional and security requirements were considered, as well. The work involved the experimentation with current tools and libraries in order to analyse how exactly they function. The implementation was used to assist with the development of the proof-of-concept. The chapter ends with a description of the proof-of-concept, in part by employing UML's sequence diagrams (presented in Section 2.9).

3.5 Evaluation

Finally, an evaluation of the results is carried out. The discussion involved an assessment of the use case, by showing how this given use case is representative enough for the given purpose and how it is relevant with regard to the research questions. In addition, to further evaluate the proof-of-concept, a second use case was presented. This serves the purpose of validating that the presented architecture is indeed suitable for different scenarios and if it can be adapted to suit other requirements in a facility management environment. This is followed by an analysis of the results, which focused on the blockchain component in order to determine what are the advantages and disadvantages of using blockchain in the use cases presented in the thesis and in the facility management environment in general. To conclude the evaluation, a mapping to existing related work is carried out.

CHAPTER 4

Results

4.1 Overview

This chapter presents the main results of the thesis. As already mentioned in the introduction, one of the expected results of the thesis is to build an artifact that can be used as a proof-of-concept with regard to the research questions (presented in Section 1.4). The chapter begins by presenting a use case in the facility management environment. This use case is taken as the basis to deduce a set of system requirements, with the goal of defining the characteristics of the artifact, and what exactly its functionality should be. Next, the components of the artifact and the technologies behind them are presented. These components are then presented in a high-level architecture diagram. To provide a complete picture, everything is put together and a description is given on how the artifact is to be used. This is supplemented by some implementation details. The chapter ends with some implications of the results.

4.2 Formalizing the *Issue Management* use case

The use case taken as the basis for this chapter is called *Issue management*. To allow for better understanding, it is important to give some context and background regarding the use case. In the facility management environment, a common scenario is that of handling issues. The term *issues* used in this thesis, in this context, means problems, defects, damages, routine tasks, and requests for assistance. Organizations usually set up processes with the goal of handling these issues in a structured manner. Often, multiple actors are involved, which adds complexity to the environment, as each actor has their own perspective and might aim for a different outcome. These characteristics were mentioned in Section 1.2 of the introduction chapter. The use case *Issue management* is presented with a user story. User Story Box 4.1 describes a user story by introducing three personas and an example scenario.

In this user story there are three personas. Bob is a software engineer, and will be also referred to as an office user. Alice is the facility manager in the same company. Charlie is a contractor, working for another company, but the issues that he is responsible to handle are in Bob and Alice's workplace.

Alice has set up some business processes to make sure that she and the company can track all facility management affairs. That is why, when Bob noticed that his door is broken, he used the facility management application to *raise an issue*. He only needs to describe what the problem is, select the corresponding BIM element, and the application handles the rest automatically. Bob cannot do anything regarding the issue anymore, other than *view its status*, to check if it has been handled.

In this case, Alice got the issue assigned to her. She pulled it up and checked what the contents were. She paid special attention to the referenced BIM element in the issue content, since that is the most important detail that Charlie will need in order to finish the repair. When she made sure the BIM element was correct, she *sent the request*, which means that it is now assigned to Charlie.

The BIM element referenced in the issue gives Charlie a lot of information. Not only does he know the properties of the element, but he can also lookup the history of this exact BIM element to extract more relevant information. When Charlie finished the repair, he *updated the issue*. Alice then checked if the task was properly finished. If she is not satisfied, Alice would have *sent back another request* to the contractor. In this case, however, the task was finished and so Alice *closed the issue*.

User Story Box 4.1: Issue management

Granted, the user story is a simplification of a real world scenario, and some of the assumptions and limitations are discussed in Section 4.4. Nonetheless, it contains the relevant constituents to work towards an artifact that showcases the technologies presented later in this chapter. For instance, the actions that each persona takes, such as *raising an issue*, *sending a request*, *updating the issue*, and *closing the issue*, are quite general and appear in real-world use cases.

Figure 4.1 represents the user story formalised using the BPMN notation (Section 2.8). The three horizontal pools represent the three roles in the user story. The entry point is the start event in the *Office User* pool. The first activity, *Raise issue*, represents the tasks that the office user performs to raise an issue. Being a message sending task activity, it sends a message, which, in this case, is delivered to the *Facility Manager* pool. The activity *Raise issue* is followed by an exclusive event-based gateway, which is fired by the

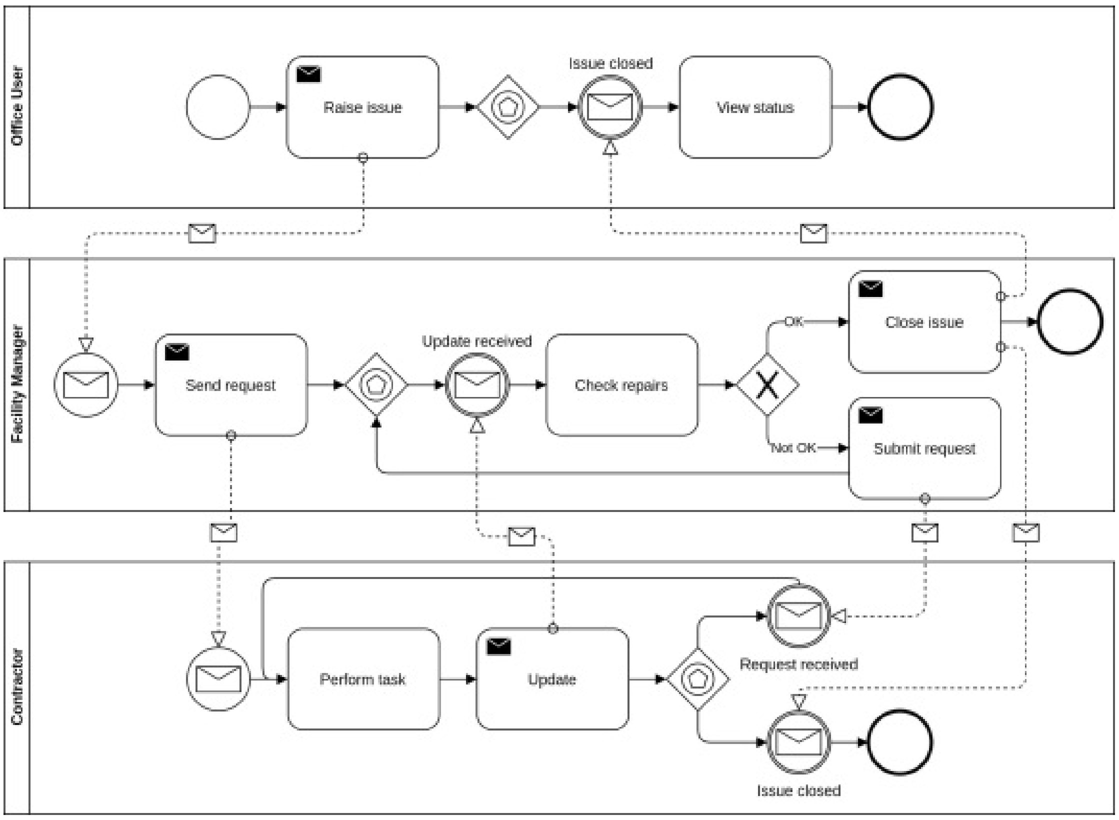

Figure 4.1: Issue management BPMN graph

message receiving event *Issue closed*. This means that the last activity, *View status*, is finally performed after the issue has been closed. At this point, the pool reaches the end event.

The pool *Facility Manager* begins with the start event triggered by the message from the *Raise issue* activity. The first activity in this pool is *Submit request*, which sends a message to the last pool, *Contractor*. The activity *Send request* describes the task of sending a request to the contractor. This activity is followed by an exclusive event-based gateway, fired by the message receiving event *Update received*. When this message is received, the facility manager performs the manual activity *Check repairs*. Based on the observations of this activity, the facility manager performs either the activity *Submit request* or *Close ticket*. In the former case, a message is sent to the *Contractor* pool, and the flow goes back to the last event-based gateway, where another *Update received* messaged is awaited. In the latter case, a message is sent to each of the other two pools, and this pool's end event is reached.

Similarly as the second pool, the *Contractor* pool's start event is again a message receiving event, in this case triggered by the message from the *Send request* activity. This is followed by the manual *Perform task* activity, representing the tasks that the contractor needs to perform to handle the issue. The next activity, named *Update*, describes the process of updating the issue with the latest information and sends a message to the *Facility manager* pool. Next, the exclusive event-based gateway waits for a message and is followed by one of two events. If the facility manager performs the *Submit request* activity, then the *Request received* event is fired, which brings the flow back to the beginning of the *Contractor* pool. Otherwise, if the facility manager performs the *Close issue* activity, then the *Issue closed* event is triggered and the pool's end event is reached.

It should be noted that Figure 4.1 is not a one-to-one translation of the user story presented in User Story Box 4.1. For example, in the uppermost pool, *Office User*, the activity *View status* comes only after the message event *Issue closed*. If such an implementation were to be carried out, it would mean that the office user can only view the ticket status once the issue has been closed. This is a limitation that strips away important functionality in a real-world scenario. However, this limitation will be rendered irrelevant in the following sections, once the requirements are defined.

Both the BPMN graphical representation and the user story provide different insights. The former, among others, makes a clear separation between the activities of each actor and at the same time defines the relationship between these activities. The latter, by means of personas, humanizes the process and therefore gives a perspective that is easier to understand. In order to derive the system requirements, both the user story and the BPMN graph were used as the basis to construct the use case diagram in Figure 4.2, which uses the UML notation (as described Section 2.9). This diagram disregards the manual activities and concentrates on those activities that are performed by the actors using the system.

In the following, the use case diagram is described from the perspective of the three actors.

The office user

Chronologically, the initial activity that kick-starts the process is *Raise issue*. This is modeled in the use case diagram by the identically named use case and represents the tasks that are performed when the office user uses the application to raise an issue. Every time an issue is raised, a BIM element must be referenced. This important task deserves its own place in the diagram with the use case *Select BIM element* and it has an *include* relationship with the *Raise issue* use case. According to the user story, the office user performs another activity, as well. This is captured by the *View issue status* use case.

The facility manager

According to the BPMN graph, the facility manager performs the activities *send request*, *submit request*, and *close issue*. The first two activities are grouped into a single use case, *Submit request*, because both have a very similar purpose. The activity that this use case represents has an optional step. Namely, the BIM element reference within the issue might need to be added or corrected. Hence, there is an *extend* relationship between the

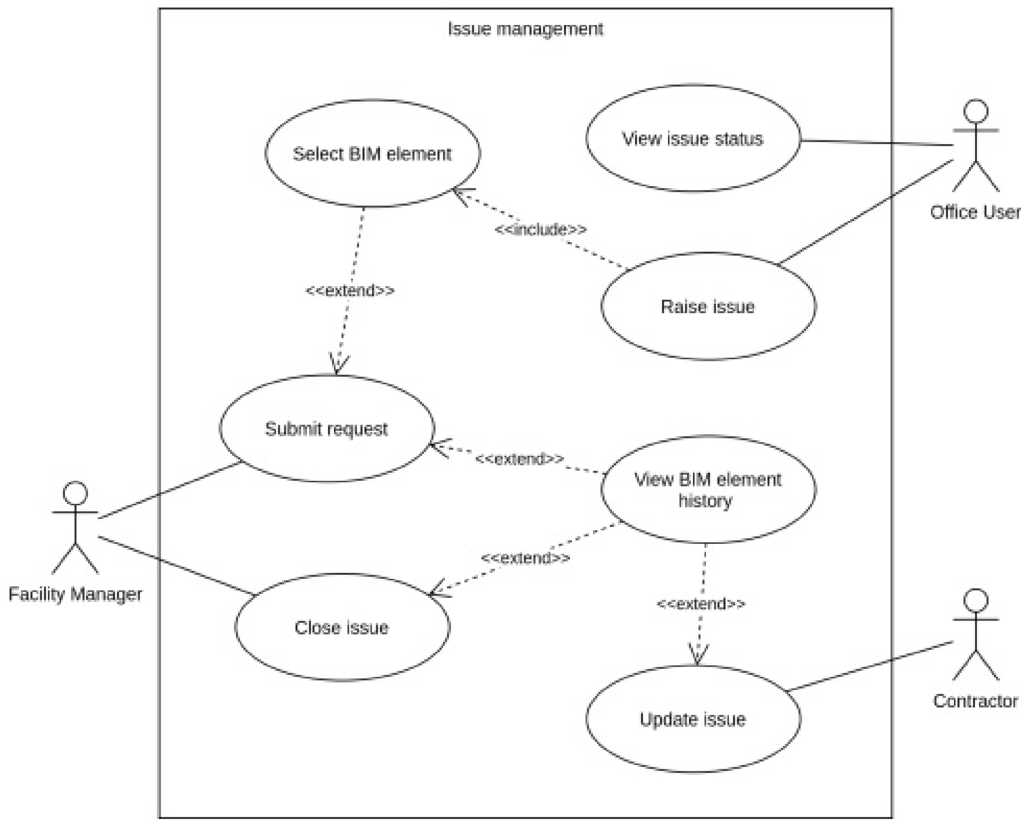

Figure 4.2: Issue Management use case diagram

Select BIM element and this use cases. Another *extend* relationship is with the *View BIM element history*. This use case represents an important activity, because it models the facility manager's access to the change history of the BIM. The same relationship exists also with the last use case, *Close issue*.

The contractor

The contractor does, in effect, just one activity, which is *updating the issue*. This is represented with the *Update issue* use case. The task of updating an issue, same as in the facility manager case, may require access to the BIM element history. For that reason, there is an additional *extend* relationship between the corresponding use cases. It should be noted that the use case *Update issue* includes the task of viewing the issue and its content, so that the contractor gets a view on the building model, as required for his work. In other words, the contractor is able to see the content of the last entry in the issue, but is not able to see any historical data. In cases when the contractor requires historical data, the *View BIM element history* use case is used. Lastly, the contractor cannot close an issue. The intended use is that the contractor updates the issue and the facility manager decides if the issue is closed or not.

4.3 System requirements for *Issue management*

The different use cases described in the previous section show that each actor has their own requirements. As described in Section 2.7, deriving the system requirements specification is a multi-step process that involves extracting information from the system users. To that effect, the second user story in User Story Box 5.1 provides more insights into the perspective of the three personas.

> For Bob, the use of such an application does not bring any direct benefits. In fact, using a separate application every time he needs to raise an issue might be a bit of a nuisance. So for him, it would be convenient to have to take as few steps as possible. At the same time, he wants to be sure that once he raises an issue, Alice will get the issue assigned to her.
>
> For Alice, on the other side, it is important to track the changes in the facility in a reliable way, not only to keep an overview of the tasks, but also to be able to derive valuable information from the data in the long run. Reliable for Alice means that whenever some data is inserted into the application, it is stored and processed correctly every time. Alice requires that she can rely on the data that she reads on the application. For instance, if the application shows that Charlie saved something, then it must be the case that Charlie did indeed save that information. This should hold the other way, as well. If Charlie saved something, then the application should show that it was him who did that.
>
> Charlie needs a way to get historical and issue data to do his job. Additionally his employer requires a reliable way to store the proof of the work that has been done. The requirement here is similar to that of Alice's, because neither side has no reason to fully trust the other one.

User Story Box 4.2: User requirements

A system requirements specification was derived based on the information presented in the user stories, BPMN graph, and the use case diagram. These are grouped into three parts, functional, data, and security requirements:

F1 Functional requirements

F1.1 The application shall assign the issue to the correct role, upon every change, as per the BPMN graph in Figure 4.1.

F1.2 Only the assigned role shall be allowed to make changes to the issue.

F1.2.1 The role office user shall be allowed to raise an issue.

F1.2.2 The role facility manager shall be allowed to submit requests and close issues.

F1.2.3 The role contractor shall be allowed to update issues.

F1.3 When the issue is closed, it shall not be allowed to be updated anymore.

D1 Data requirements

D1.1 An issue shall contain date, author, description, and shall point to a BIM element.

D1.2 An update shall contain date, author, description, and shall point to an open issue.

D1.3 The data shall be signed by the author.

D1.4 The BIM element referenced in each issue shall be part of the building information model (BIM).

S1 Security requirements

S1.1 The application shall be accessible by all users, but each shall have their own view.

S1.2 Each user shall only be allowed to perform certain functions, as defined in requirement F1.2

S1.3 The BIM data and the issue data shall only be accessible by the roles defined previously.

S1.3.1 The role office user shall be allowed to check the status of the issue.

S1.3.2 The roles facility manager and contractor shall be allowed to view the BIM element history.

S1.4 The data shall be stored in a tamper-proof environment.

4.4 Assumptions and limitations

The use case taken as the basis of the work in this chapter is a very limited representation of a real-world scenario. It should therefore be pointed out that it strips away a lot of the requirements of a typical facility management application. This was intentionally done so, in order to focus on the most important parts. For example, the way that notifications are handled is left out-of-scope. Furthermore, the classification of the issues based on their urgency, the reopening of closed issues, and anything having to do with payment handling fall into the list of out-of-scope elements, as well. For the purposes of this use case, but without loss of generality, the three roles used in the use case are assumed to be static. Lastly, the requirements about the user interface is that it is to be as simple as possible, but details are not discussed because they do not fall within the scope of this work.

Nonetheless, appropriate care was taken to make sure that this limited use case is representative enough with regard to the research questions in Section 1.4. In fact, the research questions were formulated based on the topic and research gap before the use case was devised.

4.5 Towards blockchain-enabled design

The research questions presented in the introduction are about using blockchain and smart contracts in the facility management sector. With that regard, it is worth revisiting why such questions even make sense, given that blockchain was originally developed for and used in the financial sector. To that effect, various research papers have tried to provide a structured way to determine the applicability of blockchain technology in different use cases and what approach should be taken when designing blockchain-enabled systems (presented in Section 2.3).

Issue management and blockchain

Firstly, it should be discussed if a blockchain is at all suitable for the given purpose. Blockchain has interesting properties, especially when it comes to the origin and history of the data (data provenence) and transaction verification. However, these come at a cost, as described in Section 2.2. As presented in Section 2.3, Wüst and Gervais provide a simple methodology of determining the applicability of blockchain in any given use case, which takes the form of a series of yes or no questions. For the *Issue management* use case the answers are the following:

1. Do you need to store state?

   ↪Yes

2. Are there multiple writers?

   ↪Yes

3. Can you use an always online trusted third party?

   ↪No

4. Are all writers known?

   ↪Yes

5. Are all writers trusted?

   ↪No

6. Is public verifiability required?

   ↪No

As can be seen in the *Issue management* use case, there are multiple actors interacting on a common environment. This leads to the need to store a shared state, part of which are all the actors. Thus, the first two questions answer with *yes*. Regarding the third question, as presented in the introduction, literature research suggests that in the AEC industry and in the facility management sector the large number of stakeholders result in a complex environment where having a trusted third party is often infeasible. The answer to the third question, therefore, is *no*. In the current context, all writers are known, because the use case is limited to three actors. However, they are not trusted, mainly because of the type of the relationship with the contractor. The contractor works for a different company, so the facility manager cannot assume that they will behave in a reliable way. This is also true vice versa, because the contractor does not have any assurance that the facility manager is trusted. Finally, the last question is answered with *no*, because the example use case *Issue management* describes a closed and private environment. The result of the questionnaire is a *private permissioned blockchain*.

In the facility management context, the fourth answer might occasionally be *no*. In this case, the result would be a *permissionless blockchain*. Nonetheless, irrelevant of the answers to these last questions, in both cases the result is that blockchain is needed. The only difference being the type of blockchain. While this does affect what the resulting system will look like, it does not change the fact that blockchain is suitable. This is a positive result in regard to the thesis hypothesis. Blockchain is suitable for the presented use case. In addition, the smart contracts play a vital role in it, as described later in this chapter.

Software design for blockchain

Having had established that blockchain is indeed suitable for the *Issue management* use case, the next step is to determine what is the best approach, when it comes to engineering software based on blockchain. As discussed in Section 2.3, blockchain-oriented software engineering involves, first and foremost, identifying the actors that will use the system, the trust relations between them, and the interactions that they have with one another.

In the Issue Management use case diagram (see Figure 4.2) the actors are well defined. However, there is no complete trust between them, as described in the previous section. The nature of the relationship between the facility manager and the contractor is not based on trust. Hence, they need to trust that the application will make sure the data is reliable. At the same time, it is desired that the application that they interact with has a friendly user interface. To find a balance between the user-friendliness and security features, this entails the need to have every transaction signed by the private key of the actor that initiated it. A transaction, in this sense, is any interaction with the application that involves the transfer of any data. For convenience, a separate application, called a wallet (presented in Section 2.2), is used, which provides signature services and stores the private keys securely. As a result, every time the users make a submission or request on the application, they must use their private key to sign the transaction.

4.6 The architecture of the *Issue management* proof-of-concept

The current chapter began by presenting a use case. Following that, a set of requirements were derived, and it was determined that blockchain was needed for the *Issue management* use case. The rest of the chapter deals with the construction of the artifact architecture, which will serve as a proof-of-concept. To that effect, the next sections explain, step by step, the resulting components and how they interact with one-another.

4.6.1 Data in a blockchain-supported application

Connection to the Building Information Model

Building applications for the AEC/FM Industry means, first and foremost, looking at the underlying formats and standards that store the BIM data. As described in Section 2.5, the IFC data model, including its file formats, is the most widely used standard and is therefore taken as the default for the work on this thesis. There are, however, some shortcomings of this standard, as discussed in Section 2.5. In order to be able to proceed with the building of the artifact without being negatively affected by these disadvantages, for the purposes of this thesis, it was decided that the file storing the Building Information Model will be taken as a static file. This static file is then referenced by the data structures, which are presented next in Section 4.6.1. If, in the future, changes to the IFC standards are done, or even a completely new standard is used, the work presented here does not get invalidated. It is to be thought of as a layer on top of the BIM data layer.

The implication of the decision to represent the BIM with a static file is that the model must be extracted at a certain point in time. This is done when the building enters the facility management phase. From that point forward, each raised issue references the corresponding element in the IFC file and therefore represents the changes done to the Building Information Model. This reference can also be used to derive historical information about each element. To do this, the BIM element Globally Unique Identifier (GUID) is used, which was described in Section 2.5.

Data structures in the FM phase

The second research question presented in the introduction (Section 1.4) is about the data generated during the facility management phase. In addition, there are some exact data requirements that need to be handled. In order to come up with a solution to these requirements and maintain some generality, it was decided that some of the functionality of the Interplanetary File System (IPFS) would be exploited. As described in Section 2.4, IPFS provides means to construct custom data structures. Interestingly, IPFS can be used to construct a Merkle tree, which is especially suitable in a blockchain-based system.

The issues can be seen as a blob of data. An issue contains a set of values, such as date, author, description, and a BIM element, as per the data requirements. The date is the issue creation date and time. The author is the user that, by interacting with the application, adds the issue description and selects a reference to a BIM element, such as a door, wall, HVAC unit, or a pipe. These data are put together to form an issue. An issue update contains a similar set of values, but it differs from an issue in that it does not reference a BIM element directly. Instead, it references the issue which it updates or a previous issue update. These requirements are generalized and formally described in Definition 4.6.4 and Definition 4.6.5, respectively. Definition 4.6.1, Definition 4.6.2, and Definition 4.6.3 provide some preliminaries.

Definition 4.6.1. P is list of properties $(p_1, p_2, ..., p_n)$ represented in a machine-readable format where each property p_x is a key-value pair.

Definition 4.6.2. h is a one-way hash function[1] that returns a fixed-length binary sequence, called the cryptographic hash.

Definition 4.6.3. B is the cryptographic hash of the BIM element's globally unique ID. Otherwise expressed with $B = h(GUID)$

Definition 4.6.4. An issue structure $\mathcal{I}$ is a tuple (P, B), where P is defined in Definition 4.6.1 and B is the hash defined in Definition 4.6.3.

Definition 4.6.5. An update structure $\mathcal{U}$ is a tuple (P, H), where P is defined in Definition 4.6.1 and H is the product of either the hash of the issue structure, $h(\mathcal{I})$, or the hash of another update structure, $h(\mathcal{U})$.

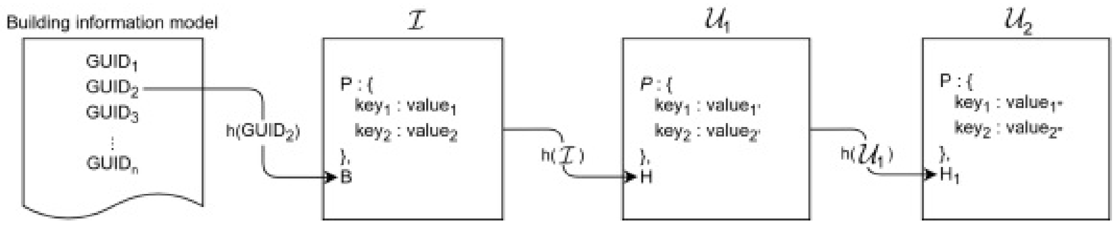

Figure 4.3: Graphical representation of the data structures

As can be seen in Figure 4.3, starting from the left side, the building information model is a file that contains all the building information. Each element has its unique GUID. As an example, the issue structure $\mathcal{I}$ contains the properties and a hash of GUID$_2$. This represents the connection of the issues to the corresponding BIM elements. The update structure $\mathcal{U}_1$ is the first update to the issue $\mathcal{I}$ and therefore, in addition to the properties, contains a hash of $\mathcal{I}$. Every subsequent update contains a hash of the previous update. This can be seen in the right side of the figure. The update structure $\mathcal{U}_2$ contains a hash of the update $\mathcal{U}_1$. In the rest of the chapter, the issue structure and the update

[1]A one-way hash function is a mathematical function that takes a variable-length input string and converts it into a fixed-length binary sequence that is computationally difficult to invert - that is, generate the original string from the hash [PG07].

structure are generalized into the structure $\mathcal{S}$, as presented in Definition 4.6.6. Once $\mathcal{S}$ is constructed, it is fed into the hashing algorithm $h(\mathcal{S})$ and this returns the fixed-length hash $\mathcal{H}$, as defined in Definition 4.6.7. The importance of these structures for the rest of the thesis lie in the fact that the they are used to construct a Merkle trees rooted at $\mathcal{H}$.

Definition 4.6.6. The structure $\mathcal{S}$ is a tuple (P, H), such that P is the list of properties defined in Definition 4.6.1 and H is a hash defined in Definition 4.6.2. Depending on how H was produced, $\mathcal{S}$ is either an issue structure $\mathcal{I}$ or an update structure $\mathcal{U}$.

Definition 4.6.7. $\mathcal{H}$ is the result of feeding $\mathcal{S}$ to the hash function. In other words, $\mathcal{H} = h(\mathcal{S})$.

Smart contract for data storage

In the previous section, the hash $\mathcal{H}$ was produced by hashing the structure $\mathcal{S}$. In this section, these two separate entities are used in conjunction to solve the data storage requirements. To do this, another property of IPFS, the content-addressed file system, is used. In concrete terms, the main idea is that the data structure $\mathcal{S}$ is stored in the file system, while the hash of the data structure H is stored on the blockchain. More precisely, smart contracts (described in Section 2.2) are used to store the hashes. The methods that the smart contracts expose are used to set and get the values of their internal variables. So, the hashes that are produced by the hashing function are set as the values of these smart contract variables. The values are then read from the smart contract. Therefore, the hash, being part of the blockchain, inherits its properties. At the same time, the content-addressable property means that the same hash is used to retrieve the data from the file system. If the data has been tempered with, then the hash itself will be the proof needed to show that this is the case.

The hash alone, however, is not sufficient in order to fulfill the functional requirements F1. In addition to the hash, the smart contract needs to store a set of additional data. This set of data, $\mathcal{C}$, is defined in Definition 4.6.8. The state of the issue stores whether the issue is open or closed, while the assignee stores the entity that the issue has been assigned to. For the purposes of the *Issue management* use case, a key-value pair that maps the BIM element to $\mathcal{C}$ is suitable. The result is that for every BIM element, the corresponding structure containing the issue or the issue update, and its accompanying set of data is stored in the smart contract. Definition 4.6.9 formalises this. Figure 4.4 builds upon Figure 4.3 to graphically show how $\mathcal{C}$ and the mapping is constructed.

Definition 4.6.8. $\mathcal{C}$ is a tuple $(\mathcal{H}, S, A)$, where

$\mathcal{H}$ is the hash of the structure $\mathcal{S}$, as defined in Definition 4.6.7;

S is the state of the issue; and

A is the issue assignee.

Definition 4.6.9. m is a mapping $B \mapsto \mathcal{C}$, where B is the hash defined in Definition 4.6.3 and $\mathcal{C}$ is the tuple defined in Definition 4.6.8.

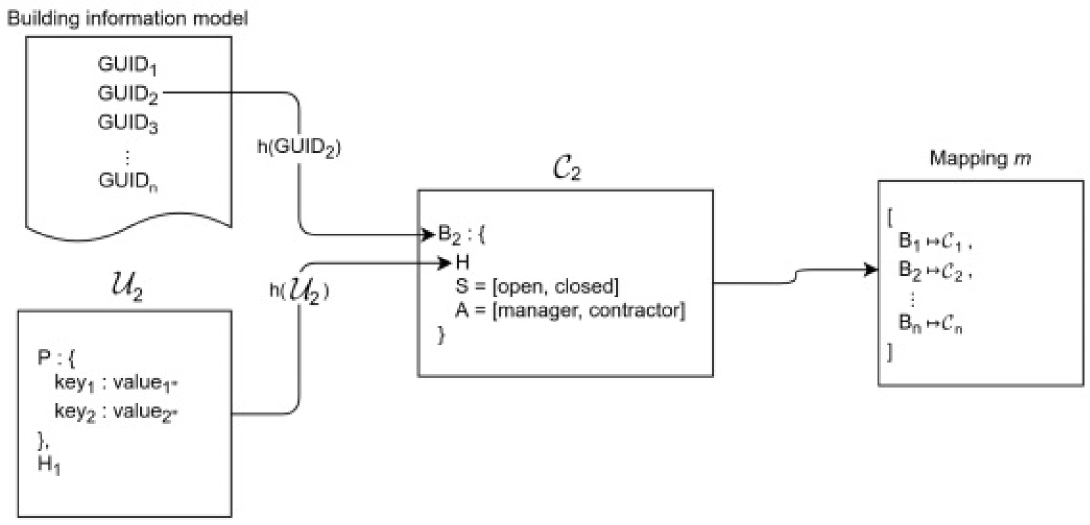

Figure 4.4: Construction of the mapping

Smart contract for business logic

To recall the definition from Section 2.2, smart contracts are computer programs and therefore replicate state machines. Each time a variable is changed, the smart contract switches to a new state. In the *Issue management* use case, each issue has only a limited number of states in which it can be. The smart contract code determines the state transitions, even though these state changes are triggered by the user-performed actions. The term business logic is used to describe this, because it represents the business process of managing issues. The exact state changes are defined in the requirements in Section 4.3. In fact, the smart contract handles all of the functional requirements and some of the security requirements. The state machine diagram is presented in Figure 4.5.

As it can be seen, an issue can have two states: raised (open) or closed. In addition, an open issue is assigned to either the facility manager or the contractor. The smart contract transitions from one state to the other based on the information it receives and who sends that information. The transitions *submit request* and *update issue*, from a data perspective, are built using the same data structures. The difference is only the assignee value. The internal processes that are fired every time a user interacts with the smart contract use the data contained in $\mathcal{C}$ and B to make these state transitions. To do this, it suffices to check who the initiator of the transaction was and what operation did they try to do. This is again, computationally, just a string comparison operation. The public key of each user is stored in the smart contract, and this is then used to verify the identity of the entity that initiated the transaction. Based on this verification, the change is either allowed or denied. This is described in more detail in Section 4.7.

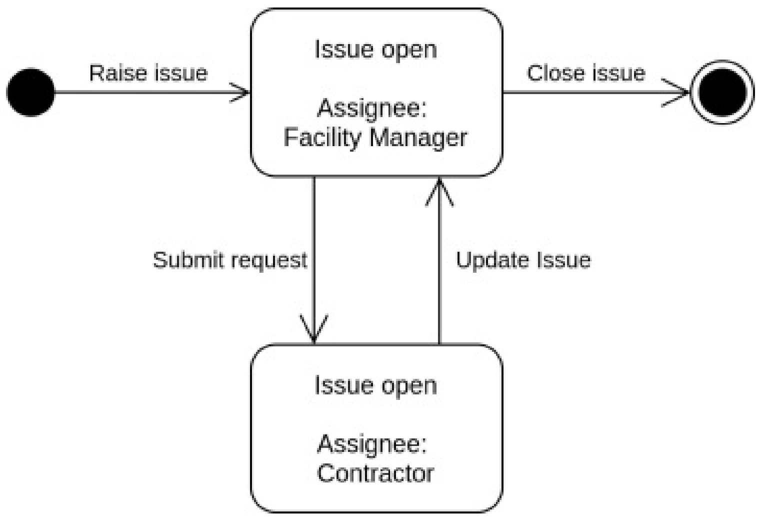

Figure 4.5: State machine diagram

Smart contract for data verification

The separation between the file system and the smart contract leaves a security weakness. A malicious actor might attempt to store a hash that does not adhere to the definition of the data structure $\mathcal{S}$. One way to prevent this is by adding some functions in the smart contracts that perform checks. The operations performed by these functions are described by the following two steps.

The first step checks if the reference hash is contained in the data structure. If the user is raising an issue, then the smart contract must receive an issue structure $\mathcal{I}$. On the other hand, if the user is updating an issue, then the smart contract must receive an issue update structure $\mathcal{U}$. It therefore suffices to check that the structure contains the hash H in the first case, or $\mathcal{H}$ in the second case. Computationally, this is a string comparison operation. If the hash is not contained in the structure, the process is aborted and the state of the contract is not changed. In the smart contract code, this was done by adding the necessary methods that are called each time its state is about to be changed.

As a second step, it must be verified that the hash $\mathcal{H}$ is indeed the hash of the structure $\mathcal{S}$, as per Definition 4.6.7. As discussed, the structure $\mathcal{S}$ is stored in a separate file system and only the hash $\mathcal{H}$ on the smart contract (blockchain). So, it must be checked that the equation $\mathcal{H} = h(\mathcal{S})$ holds. This verification step checks if feeding the structure $\mathcal{S}$ to the hash function results in the same hash, compared to the hash provided to the smart contract. This was done by encoding the same hash function in the smart contract. The corresponding smart contract method takes in as arguments the structure and the hash. It feeds the structure to the hash function and compares the resulting hash to the one sent to the smart contract. If the hash is the same, then the structure is valid and the hash correct. Otherwise, the process is aborted and the state of the contract does not change.

4.6.2 High-level architecture overview

The technologies and components presented in this chapter are summarized in the high-level diagram in Figure 4.6, which provides a simple and easy-to-read overview. The diagram also shows the relation between its components.

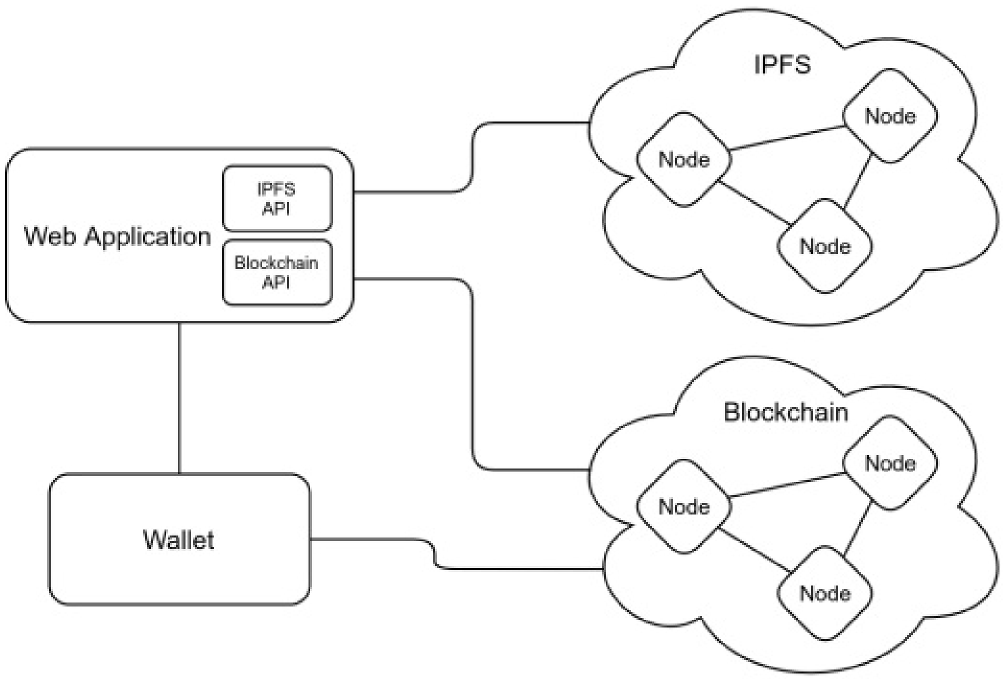

Figure 4.6: High-level architecture

The point of access for the user is the web application. Not only does it provide a user interface to the user, it is also the interface between the user and the functionality provided by the smart contracts. The web application uses an API to utilize the different tools that IPFS offers, such as constructing data objects and hashing algorithms to generate hashes. Furthermore, the web application connects to an IPFS node to save and retrieve the data. Similarly, the web application uses an API to communicate with the blockchain node, in order to send transactions to the smart contract. These transactions contain the data generated previously. As described in the previous subsection, the smart contracts, which run on the blockchain nodes, perform the two-step checks before changing their state. Additionally, they run the business logic. In other words, they decide the state transitions in the smart contract based on the input and the user initiating the transaction. The blockchain is represented by a cloud because it is a distributed network consisting of many nodes, as described in Section 2.2. Similarly, the file system IPFS is a peer-to-peer network consisting of multiple nodes communicating with one-another. Lastly, a separate entity is the wallet. The wallet maintains a connection to the blockchain and the web application, while also providing a user interface to the user. The purpose of the wallet is to provide signature services and store the private key of the user, as described in Section 2.2.

4.7 The *Issue management* proof-of-concept

The use case diagram presented in Figure 4.2 was constructed mostly from the perspective of the different actors. In light of the additional information presented following that use case diagram, and by incorporating the high-level architecture graph, a more technical use case diagram was constructed and presented in Figure 4.7. This diagram's purpose is to show what use cases each of the distinct components have.

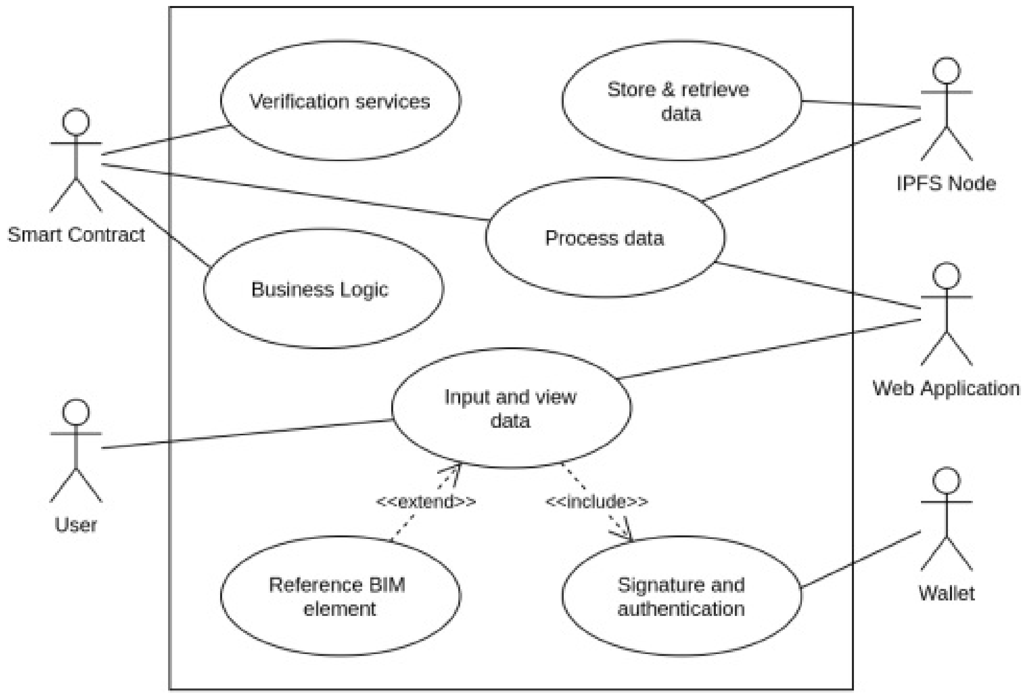

Figure 4.7: Technical use case diagram

In contrast to the first diagram in Figure 4.2, this use case diagram consolidates the different roles into one actor, called *user*. As a result, the corresponding use cases are also generalized into a single one, *Input and view data*. The *web application* actor is also present in this use case. This is clear, given the fact that the *user* uses the *web application* to perform the tasks. The *Reference BIM element* use case is an extension, because it is optional. Only in certain cases and by certain users is it to be executed. In contrast, the *Signature and authentication* use case has an include relationship, because every operation involving the user must be signed by the user using a *wallet*.

The actor *smart contract* takes part in three use cases. The *Verification services* use case represents the checks that are encoded in the smart contract to verify the correctness of the data and hash. The *Business logic* use case represents the opening, updating, and closing of issues, in addition to the related operations, such as issue assignments. The third use case, *Process data*, is shared between two other actors, the *IPFS node* and the *web application*. It represents the operations that the website performs with the data provided or requested by the user. It uses the IPFS tools to generate data structures, run the hashing algorithm, and store the hash in the *smart contract*. When data is retrieved, the opposite occurs. The *smart contract* provides the hash, while the data retrieved from the *IPFS node* is processed. Lastly, the *IPFS node's* use case *Store and retrieve data* is self-explanatory. As the file system, its purpose is to store and retrieve the data, among other functions, discussed in Section 2.4.

Using the proof-of-concept

To explain exactly how the artifact is meant to be put into use, it is best to use a bottom-up approach. In this section, the most relevant use cases are decomposed into the operations that are performed in them. These operations are then examined in a sequential manner, in part by employing an UML sequence diagram (described in Section 2.9).

As seen on top of the *raise issue* sequence diagram in Figure 4.8, the actor lifeline is complemented by four object lifelines, which correspond to the components also presented in the high-level architecture diagram in Figure 4.6. In this case, the sequence starts with the actor raising an issue. It is assumed that the actor has already inserted the data and selected the BIM element in the web application. The application then proceeds by storing the data in the file system (IPFS node), which returns the hash of the data. Next, using the wallet, the actor signs the transaction. Before storing the hash in the smart contract, the data and the hash must be verified, as described in Section 4.6.1. If the data is valid, then the smart contract state is changed and the confirmation message is shown to the actor. Otherwise, if the data is not valid, the process is terminated and the actor is informed.

There are some operations that take place between the messages that are not represented in the sequence diagram in Figure 4.8. Of importance are the operations that happen in the smart contract. These were previously described as *business logic*. The Section 4.6.1 presents the needed structures that allow the smart contract to handle the business logic. This is best explained with an example.

Consider the sequence diagram in Figure 4.8 as the basis for the example. The issue raised in that case would trigger the smart contract to store a key-value pair, where the key is the hash of the BIM element it references, and the value is a set containing three values: the hash of the data stored in IPFS, a boolean value to denote that the issue is open, and the address of the facility manager user, who is the assignee, as per the use case presented in the beginning of the chapter. Since the smart contract has these values stored, it can perform the necessary checks when the next user tries to update the issue. More precisely, if the update is not signed by the facility manager, the smart contract

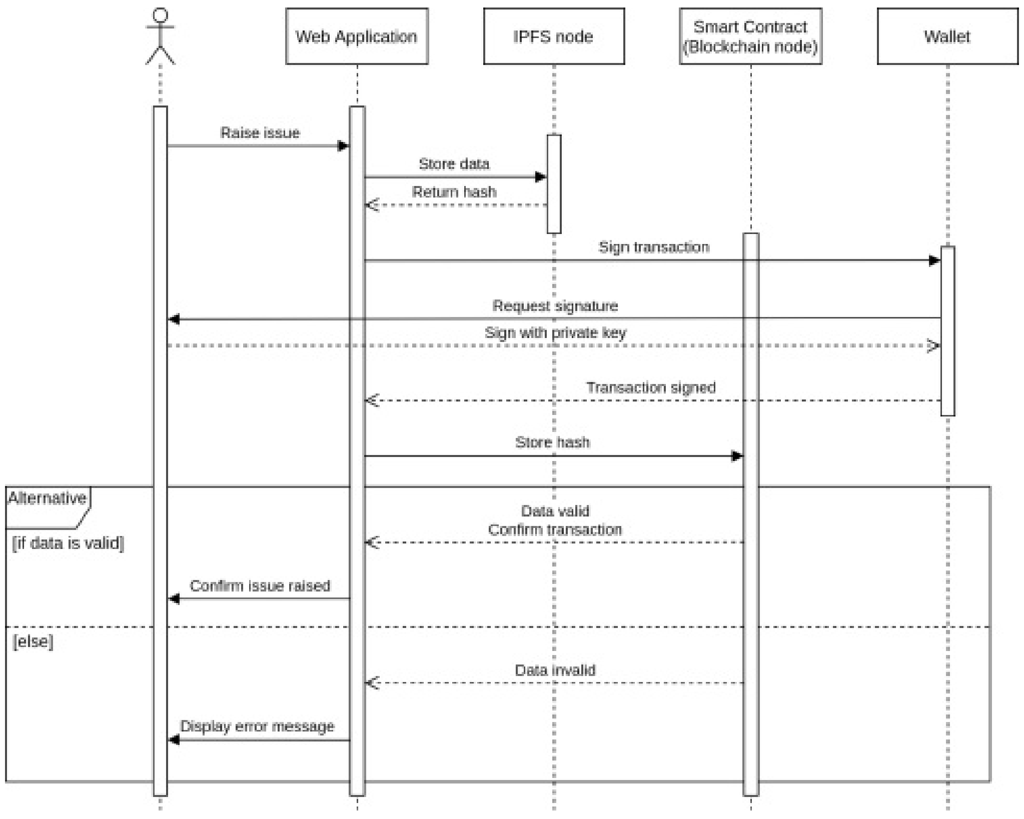

Figure 4.8: Raise issue sequence diagram

will not allow its state to be changed. Otherwise, if the facility manager does indeed submit an update, the smart contract updates the key-value pair with the new hash and new assignee.

4.8 Some notes on the proof-of-concept implementation

As part of the experimentation, some of the features and components of the proof-of-concept were implementated, albeit simplified in some ways. In a Linux environment a local Ethereum blockchain node using the Truffle Suite's Ganache (Section 2.2) was set up. Additionally, an IPFS (Section 2.4) node was spawned in the same local environment. A simple JavaScript web application was developed which interacts with both the blockchain and IPFS nodes. The libraries *ipfs-http-client* and *web3.js* were used for those purposes. The smart contract was written in Solidity. The Metamask (Section 2.2) browser plug-in was used as the wallet. The accounts added in the wallet are local and self-generated by Ganache, when the local blockchain node is spawned. Ganache initializes each account with 100 Ethereum units, which are used to make transactions on the blockchain.

To implement the proof-of-concept, the developed dApp consists of a smart contract and a simple HTML website with a few text boxes and buttons to enable user interaction. In the website, the users, in the cases when they need to raise an issue, have the ability to input the data, select the BIM element, and raise the issue. Similarly, upon issue updates, they can load the BIM element history and write an update. These operations, as described previously, are performed with the help of a smart contract and an IPFS node. The libraries used are the *web3.js* to enable functionality for the Ethereum blockchain and the *ipfs-http-client* to connect to the IPFS node.

4.8.1 Contents of the smart contract

The smart contract is written in Solidity, which is one of the languages that can be used in Ethereum to write smart contract code. The smart contract has three central functions: `raiseIssue(string)`, `updateIssue(string, string)`, and `closeIssue(string)`.

The function `raiseIssue(string)`

The `raiseIssue(string)` function accepts a string as a parameter and returns a string. The string that it accepts as a parameter is the hash of the issue that was stored in IPFS and is used to retrieve the data from IPFS. When the web application calls this function, the first operations that this function performs is it that checks if the user trying to raise an issue is allowed to raise an issue and that there is not already an open issue. If the conditions are satisfied, then the next operations assign the issue to the facility manager, save the hash in the smart contract, and set the issue status as open. Otherwise, the function returns a `not allowed` string.

```
function raiseIssue(string memory newHash) public returns (string memory) {
   if (
      assignedTo == keccak256(abi.encodePacked(msg.sender)) &&
      !issueOpen
     ){
         assignedTo = facilityManager;
         hash = newHash;
         issueOpen = true;
   } else {
      return "Not allowed";
   }
}
```

Code Snippet 4.1: The `raiseIssue` function

The function `updateIssue(string, string)`

The `updateIssue(string, string)` function takes in two parameters. The first one is the hash from IPFS, as in the previous function, while the second one is the content of the update structure, as described in Section 4.6.1. The function performs the following operations. Firstly, it is checked whether the previous hash, which is already stored in the smart contract, is included in the update structure. If it is not, then the function returns a `not allowed` string. Otherwise, it is checked that the issue is open and whether the user updating the issue is the user that the issue is assigned to. The function has a nested `if-else` statement, because it must perform different operations based on which user is interacting with the smart contract: the facility manager or the contractor. For example, if the assignee is the facility manager, then the contractor is set as the next assignee, and vice versa. As a final operation, in either case, the new hash, which is the function's first parameter, is also stored in the smart contract. If the checks do not pass, the function returns a `not allowed` string.

```
function updateIssue(string memory newHash,
                     string memory content) public returns (string memory){

   bool hashIncluded = checkHash(hash, content);

   if (hashIncluded) {
      if (
         assignedTo == facilityManager &&
         keccak256(abi.encodePacked(msg.sender)) == assignedTo &&
         issueOpen
         ){
            hash = newHash;
            assignedTo = contractor;
      } else if (
               assignedTo == contractor &&
               keccak256(abi.encodePacked(msg.sender)) == assignedTo &&
               issueOpen
               ){
                  hash = newHash;
                  assignedTo = facilityManager;
      } else {
         return "Not allowed";
         }
   } else {
      return "Not allowed";
      }
}
```

Code Snippet 4.2: The `updateIssue` function

Other functions

The function `closeIssue(string)` does not have any additional operations. It encodes the operation of sending a hash to the smart contract with the goal of closing an issue. The function checks that the user calling this function is the facility manager, because if not then it returns `not allowed`. The hash is then saved in the smart contract. In addition to these three, there are other functions needed for the implementation of the other features. These are the helper functions that perform sub-tasks which are part of the operations described previously, such as storing and loading the hashes from the smart contract and performing the different checks.

As it can be seen, not all functionality described previously in the proof-of-concept is covered in the implementation. The experimentation part was mainly meant to test out the ideas of the thesis, and not to provide a fully-fledged application. For example, the IPFS hashing algorithm was not implemented in the smart contract. Additionally, the function `closeIssue(string)` does not contain a second parameter, as does the function `updateIssue(string, string)`. The main difference from the proof-of-concept and the implementation is that the implemented smart contract does not store a list of key-value pairs to represent the issues. Instead, it only works with one issue at a time. It does not allow the opening of a second issue, if there is one already open. Finally, error handling was also simplified, in order to avoid using Solidity-specific keywords.

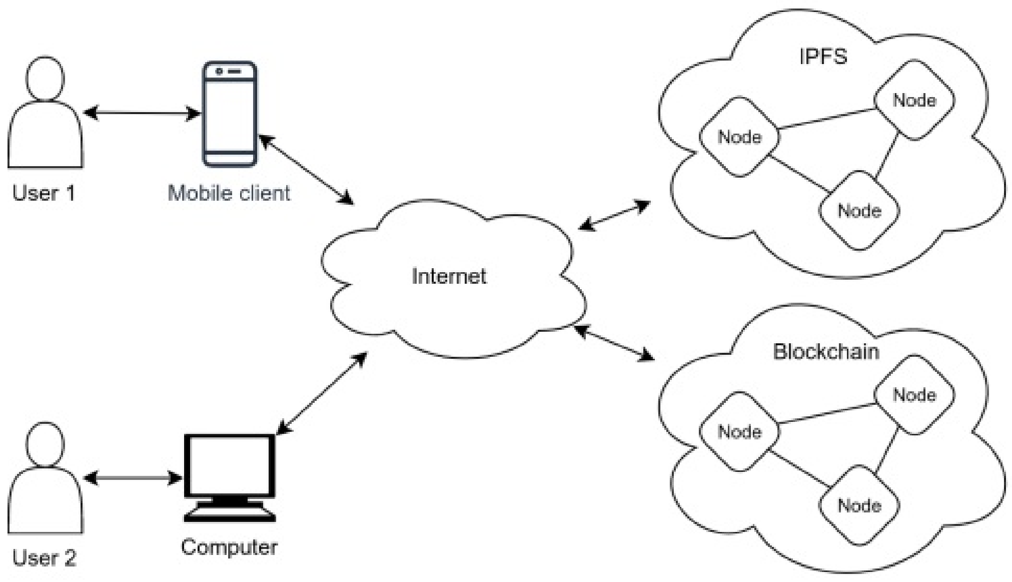

Figure 4.9: System architecture

Figure 4.9 shows a system architecture derived from the implementation. It displays how the users interact with the system. The IPFS and Blockchain components of can be seen as part of the Internet or as virtual networks where the location of the nodes is not known or relevant. Noteworthy is that the architecture lacks any centralized components, such as servers. For the implementation, the website was stored in IPFS, so the user loads the website by simply querying the website files from IPFS using the hash.

The implemented components worked as expected. It should be noted that there is no obvious reason why it would not be possible to implement also the features that were not implemented, as smart contracts are Turing-complete (Section 2.2). However, these technologies are very young and are experiencing rapid development and changes. The goal of the work described in the thesis is to be more general in nature, so in order not to risk describing implementation details that might very soon become obsolete, these implementation details are not discussed any further.

4.9 Implications

During each of the steps in the current chapter, decisions were taken which determined how the problems were solved. In this section, some implications of these decisions are discussed. This does not include any implications that come directly from the use of certain technologies, such as blockchain, as these are left to be discussed in the next chapter.

Storing the hash of the issue and its subsequent updates in the smart contract means that ultimately, when the issue is closed, the last hash stored in the smart contract will be the hash of the issue's last update. When the next issue for the same BIM element is opened, according to the data structure defined in Definition 4.6.6, the issue will reference again the BIM element, and this hash will be stored in the smart contract. Thus, the hash of the first issue would be lost. However, there are multiple ways to handle such scenarios, for example by simply considering new issues as issue updated.

Another implication that stems directly from the way the data is constructed is that, if taken as presented, data lookup may not be very efficient. The process of loading the data using a hash, extracting the old hash from the data, and repeating this loop until the hash points to the BIM element requires a potentially large number of repetitions of the same operation. Again, there are different ways to handle this, for example by archiving all old issues and adding the corresponding hash to the smart contract.

A second implication with regard to efficiency has to do with the smart contract verification (Section 4.6.1). Running the verification steps on the smart contracts can be computationally resource demanding, mainly because, by design, that code will need to be re-executed in multiple nodes. However, given the distributed nature of the system, such steps are necessary for security reasons, even though they decrease the efficiency and introduce redundancy.

Hard coding the public key of each user in the smart contract leads to another implication. If a user is changed, then the smart contract will need to be redeployed, because such a change would represent a new smart contract. In other words, every user change would result in a new smart contract. This was just a design decision and can be implemented differently without any negative side-effects.

4.10 Summary

The results chapter began by presenting a use case and concluded with a proof-of-concept that shows an unconventional and possibly novel way to build an application used for change management and tracking in facility management. The proof-of-concept uses modern technologies, such as blockchain, and combined them with interesting concepts for efficient data structures, such as Merkle Trees. These results deserve, however, some discussion, presented in the next chapter.

CHAPTER 5

Discussion

5.1 Overview

This chapter is divided into multiple sections with the aim of discussing the results presented in the previous chapter. Firstly, the use case presented in the previous chapter is evaluated by discussing how it connects the research questions with facility management practices. Next, to validate the previously presented proof-of-concept, a second use case is presented, using the same methodology as in the first one. This use case presents another common scenario in the facility management context and is used to evaluate how well the presented architecture is able to handle different requirements. To build a complete picture, the last section of the discussion chapter maps the work done in this thesis to current related research work.

5.2 Evaluation of the *Issue management* use case

The previous chapter presents a use case, called *Issue management*, and describes how some ideas and technologies are used in possibly novel ways to build a proof-of-concept. In the following section, the solution evaluation is performed using two steps. First, it is discussed how well the proof-of-concept fulfills the system requirements specification defined in Section 4.3. This step makes sure that the system is indeed designed as per the requirements of the use case *Issue management*. The second step places the proof-of-concept and the research questions side-by-side for comparison and analysis.

Revisiting the *Issue management* system requirements

The functional requirements F1 describe the way issues are assigned and the different operations that each user performs. These are complemented by the security requirement S1.2, which reinforces that the functional requirements should be taken as a strict rule and deviations from them are not allowed. Such requirements were shown to be fulfilled by the way the smart contract was constructed. Namely, the smart contract has encoded rules that handle the issue assignments. It allows or denies operations based on who performs them and what exactly the operation is. As an example, the smart contract has encoded rules that state that only the facility manager shall close an issue. If another user tries to close an issue, then the smart contract does not allow the operation.

Moving forward, the fulfillment of the data requirements D1 presented an important section of this thesis. There is a requirement that the issues reference a building information model. The data structures built for the purposes of this thesis connect each issue to a BIM element by firstly including a BIM element reference inside every issue structure and secondly, by including a reference to an issue structure in every update structure. Every structure constructed in this manner has a reference to either a previous structure or to the corresponding BIM element. The requirements as to what exactly should be stored for each issue are also easily fulfilled, because the data structure presented is very flexible and includes different types of information. The use of the wallet, as a separate application, paves the way for the signing of the data, which is also part of the requirements.

The security requirements S1 are partly covered by the smart contract code. However, there are some data security aspects that are worth mentioning here, especially because of the use of blockchain and the distributed file system. Since these are very important when it comes to the whole thesis, they are covered separately later in the Section 6.2

The *Issue management* use case and the research questions

The use case *Issue management* presents a limited scenario where multiple actors interact with a system with the goal of creating and updating issues. Since the issues are in the facility management environment, each one of them is, in effect, a change of the building itself. In other words, the issues and the subsequent issue updates are all changes to the building information model (BIM), and thus, also part of the BIM. This establishes that the environment described in the use case matches the environment of the first research question, which is *change tracking and management in facility management*. In the latter part of the previous chapter, a method to handle such issues is presented. In more concrete details, the data structures capture the data created during the process. The smart contract is then used to not only manage the process but also set up some specific rules that fulfill even detailed requirements, such as which entity is allowed to create or update issues. Using blockchain (and smart contracts), the requirements of the use case are indeed fulfilled. Furthermore, the proof-of-concept even gives a concrete example and thus provides an answer to the first research question.

With regard to the second research question, this is tackled in two ways. Firstly, the data structures link each issue to the corresponding BIM element, the issue update to the issue it is referencing, and every consequent update to the corresponding update. Secondly, storing the data represents a different task, because, as presented in Section 2.2, it may not always be convenient to store large amounts of data inside the blockchain. In some cases, such as the Ethereum blockchain, there are financial costs involved with this. In others, even if there is no financial disadvantage, the amount of data replication among the nodes would overwhelm the storage capacities. Thus, a content-addressed file system was used to store the data, while the hash pointing to that data was stored in the smart contract. The advantage to using a separate file system, such as IPFS, is that the data management is done independently and separately (more on Section 2.4). Same as with the first research question, the proof-of-concept presents an example and thus is an answer to the second question.

5.3 The *Routine task management* use case

The second part of this chapter deals with the presentation of another use case and its evaluation. This use case describes a different scenario that appears often in facility management environments: recurring tasks that need to be performed on a regular basis. Same as in the first use case, the user story in User Story Box 5.1 gives an scenario. With the help of the BPMN notation, Figure 5.1 formalises this process and denotes the different activities that are part of it. Finally, the use case diagram in Figure 5.2 shows the user interaction with the system based on the different activities.

System requirements for *Routing task management*

Having presented the use case, the methodology of the previous chapter is followed by defining a set of requirements. Same as in the *Issue management* use case, they are again grouped into functional, data, and security requirements.

F2 Functional requirements

F2.1 The role facility manager shall raise maintenance tracking records.

F2.2 Only the role constructor shall be allowed to make contributions to the tracking record.

F2.3 The roles facility manager and HVAC unit operator shall only be allowed to view historical data.

F2.4 The maintenance tracking record shall only be updated once a month, on the 15th day, with a tolerance of 3 days to accommodate for weekends and holidays.

D2 Data requirements

D2.1 A tracking record shall contain date, author, description, the recurring schedule, and shall point to a BIM element.

In this second user story, two personas are the same as in the *Issue management* use case: Alice, the facility manager, and Charlie, the contractor. The third persona is Erin, a subordinate of Alice, who is in charge of, among other tasks, operating the building HVAC unit.

When the company installed the HVAC unit, the vendor instructed Alice that there are monthly maintenance tasks that need to be performed, otherwise the warranty would not be valid anymore. It immediately occurred to Alice that she can use the facility management application to make sure this is done consistently and reliably. So, she set up a task that would assign a task to the contractor on the 15th of each month. This way, she would be able to use the application to check and file the report every month-end.

Every month, on the 15th day, Charlie gets this task assigned to him. Each time, he goes to the HVAC unit room and performs the maintenance. He then proceeds by scanning the NFC tag and updating the tracking record using the facility management application.

Last month, however, Erin, the HVAC unit operator, received an error message from the HVAC unit. She started the facility management application to check the log. There she could see that Charlie had performed the maintenance that month, so the problem lies somewhere else. She proceeded by raising an issue...

User Story Box 5.1: Routine task management

D2.2 A record update shall contain date, author, description, a valid NFC scan, and shall point to a tracking record.

D2.3 The BIM data and the tracking record data shall only be accessible by the roles defined previously.

S2 Security requirements

S2.1 The application shall be accessible by all users, but each shall have their own view.

S2.2 Each user shall only be allowed to allowed to perform certain functions, as defined in the functional requirements F2.

S2.3 The data shall be stored in a tamper-proof environment.

Using the proof-of-concept architecture for *Routine task management*

It can be seen that there are some similarities between the two use cases presented in this thesis. Not only the use case diagrams, but also the requirements bear some similarities

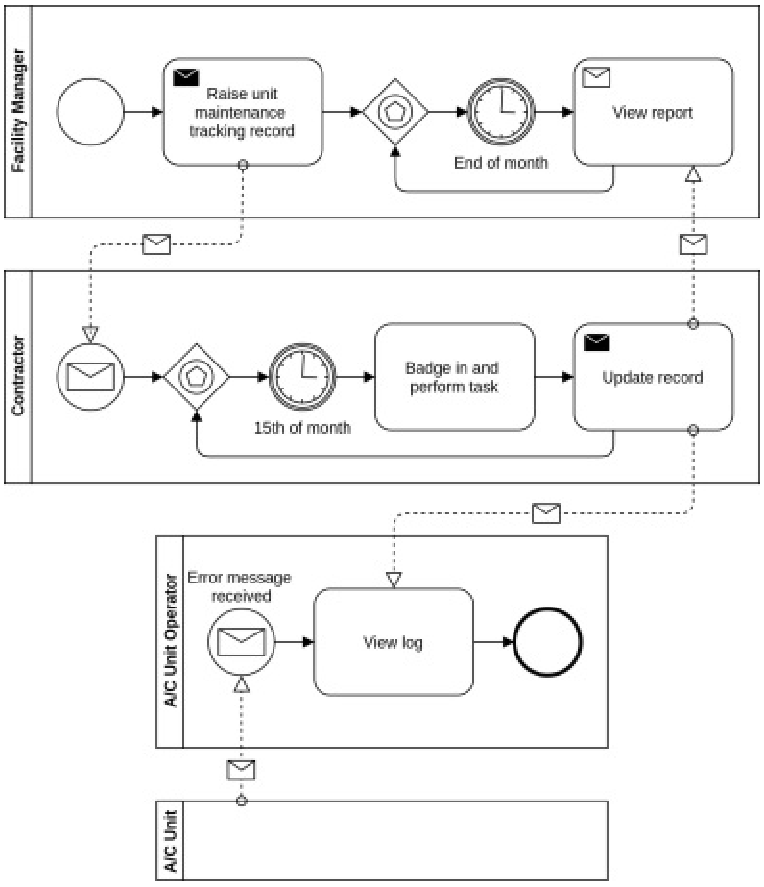

Figure 5.1: Routine task management BPMN graph

between each other. In order to show that the architecture of the proof-of-concept presented in the previous chapter is capable of handling the requirements of the *Routine task management* use case, in the following, each use case is revisited and the requirements are discussed.

Raise unit maintenance tracking record

A central part of use case diagram in Figure 5.2 is the *Raise unit maintenance tracking record*. Its centrality lies in the fact that it can be seen as a trigger for the other use cases. The purpose of raising the tracking record is to keep track of the maintenance tasks that will be performed routinely. This bears resemblance to the *Raise issue* use case in the first use case diagram. In fact, this use case can be reduced to *Raise issue*, because is it a subset of the activities included in *Raise issue*. A tracking record can be seen as a issue and a subsequent list of updates. This, it can be seen that the data structures presented in the previous section are indeed suitable to support these requirements. In addition, the smart contract code handles the business logic just as it did in the previous

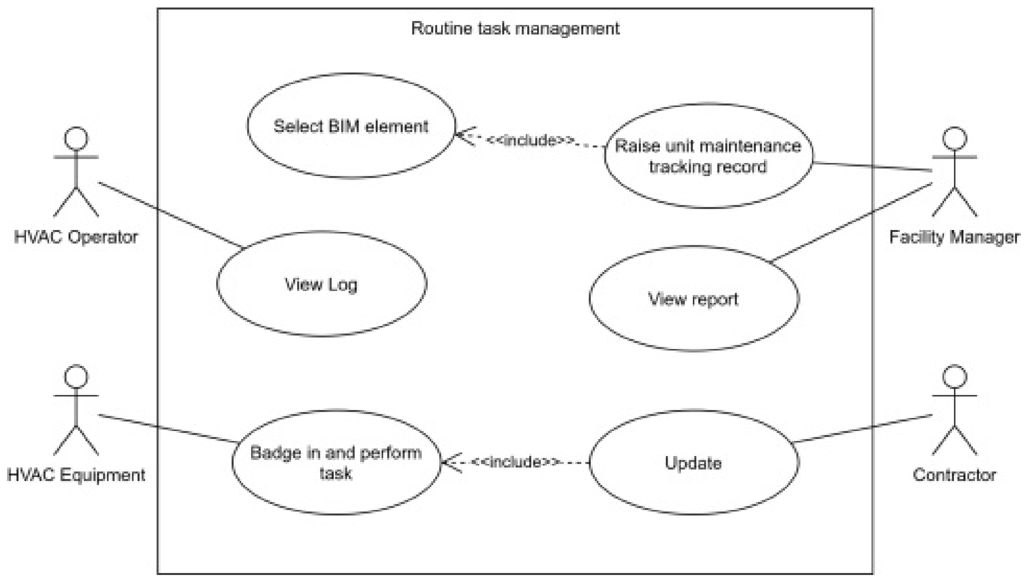

Figure 5.2: Routine task managemet use case diagram

case. Instead of the back-and-fourth between the contractor and the facility manager, here, the updates are only from the contractor. In other words, the assignee is always the contractor.

Select BIM element

As described in the previous chapter, this use case is concerned with the relation between the facility management application and the building information model. The data structure that was presented stores the needed issue data, while at the same time providing a reference to the BIM element which the issue is referencing. In this use case *Routine task management*, the BIM element is the GUID of the facility object that requires maintenance, such as the HVAC unit. This means that the same solution as in the first use case can be employed here, as well.

View log and *View report*

These two use cases are grouped together because they represent a similar operation. Both are about extracting data from the file system. The difference between the two has more to do with the way they are presented to the user. This means that this can be solved by the web application itself, since the data on the back-end is the same as on the *Issue management* use case. Details about the user experience and the web application were left out-of-scope from this thesis, as they are not part of the focus, which is the use of blockchain in facility management.

Update

The *Update* use case is also similar to the *Update issue* use case in the previous chapter. The difference from a functionality perspective is that, in this case, it is not required that the assignee is changed after every update, but instead stays the contractor. An additional requirement here is that the update is only allowed during certain days each month. In order to enforce this, an additional smart contract check can be added. This just checks the date stamp of the transaction and can therefore make the decision whether to allow or deny it. A second requirement is the NFC tag. Since the tag's unique ID will be included in the update structure, the smart contract can again be used to check whether it is the correct tag.

Badge in and perform task

Included in the *Update* is the *Badge in and perform task* use case. The badge-in refers to the process of scanning the NFC tag as proof that the contractor was indeed in that room. The process of scanning an NFC tag is nothing more than a data input process. In other words, some sensor is used to detect the unique ID of the NFC tag. This is then converted into the appropriate form and stored in the update together with the other data, as per the requirements. Again, this is not discussed further as it is out-of-scope, with regard to this thesis. However, it should be pointed out that it does not interfere with the other architecture components and the method in which they function. The HVAC equipment actor is also part of this use case, since the contractor works on the HVAC equipment to perform the task.

Evaluation of the *Routine task management* proof-of-concept

The discussion of each of the use cases shows that the architecture presented in the results chapter is suitable to handle the requirements of the second use case. Granted, there are some differences that need to be handled. However, they are manageable using the tools already presented. For instance, the smart contract that handles the issues, with a few changes, can handle the maintenance tracking records. Specifically, the changes needed are regarding the business logic. Once a tracking record is raised, it remains assigned to the contractor. The smart contract can also be used, same as in the first use case, to perform checks and validation. In addition to the checks performed in the first use case, here, additional checks are added. For example, it must be checked that the requirement F2.4 is fulfilled. Upon a record update, the smart contract rejects the transaction if it is not within the time period allowed, which is the 15th of the month and with a 3 day tolerance. Secondly, the smart contract also validates if the NFC tag scanned is included and, if yes, it validates if it is the correct one.

5.4 Evaluation of the proof-of-concept

The central component of the architecture presented in the previous chapter, with regard to this thesis, is the blockchain. As already presented in Section 2.2, one of the fundamental properties of blockchains is their distributed immutable ledger. As thus, one of the most important characteristics of the presented proof-of-concept is that it has a component that stores immutable data.

In the previous chapter, it is shown exactly what is stored in the blockchain. When the building enters the facility management phase, its state is represented by the building information model. This model contains every element that is part of the building, representing a so called digital twin of the building. Every operation that occurs during the facility management phase involves a building element, which is part of the BIM. The issue data structures described in the previous chapter are constructed in such a way so that each one of them references a BIM element. Once an issue is created, the hash derived from its content is saved on the blockchain. This record becomes part of the blockchain and will not get removed from it. The same holds for each update that follows the issue.

The significance of this approach, especially in the facility management environment, is that it provides full transparency to the stakeholders. At any given point in time, if information is requested regarding who created an issue and at what time, that information can be found on the blockchain. What is more, there is by design a very high degree of trust that the information is correct and it has never been tampered with. The benefits of this are that the stakeholders involved in the process can expect full accountability of everyone's actions.

On the front of automation, smart contracts provide many benefits, as well. The proof-of-concept uses a smart contract for data storage, business logic, and verification and checking services. The business logic handling represents an important element in the facility management environment, because it is used to automate many different tasks. In the presented proof-of-concept, it handles user assignments, so that, at any given point in time, there is only one user to whom the issue is assigned to. Additionally, the business logic makes sure that the system is self-managed by allowing or denying transactions based on who initiated the transaction and what the issue status is. In the first use case, these are mainly about verifying that the correct user is initiating the transaction. This simplistic realisation is easily extendable to include other tasks, as the second use case shows. In the second use case, the checks are extended to check if a transaction is being initiated at a specific time-slot.

Another functionality handled by the smart contract are the verification and checking services. The smart contract makes sure that the issues raised are compliant with the very important requirement to have every issue reference a BIM element. In addition, the smart contract also checks that every consequent update references the previous one. These are important in facility management because they give context to every issue and make sure that all building information that is created is connected to a BIM element.

Even though the use cases presented in this thesis only involved three actors each, they still represent an environment where multiple stakeholders with different interests and objectives interact on a common environment. So, despite the fact that the tasks are distributed among all the stakeholders, they are still referencing one common building information model.

Mapping to existing research

In this section, the results of this thesis are mapped to and compared with the research presented in the Introduction and Theoretical Background chapters. It is discussed how the work presented here was influenced by the literature and in what ways it compares to the related work.

The main motivator for the work here were the challenges described in numerous papers that the facility management sector experiences in relation to trust, as presented in the introduction (Chapter 1). The papers mentioned in the problem statement (Section 1.2) suggest that blockchain and smart contracts can directly help remediate these challenges. They do not, however, provide any details on what exactly it takes to implement or adapt these technologies for the needs of the facility management environment. To that extent, the work in this thesis tries to fill this gap by describing an architecture that contains the required components to make such a system work.

On the other side, the papers presented in the related work (Section 2.10) look at the problematic from a more technical perspective. To begin with, the papers that describe some actual smart contract implementations bear some similarities in that they typically have a smart contract that is designed to simulate tasks in the facility management environment. These smart contracts are mostly written as contracts, which means that they, in a way, hold the involved parties accountable for the actions that they have taken or must take. As a consequence, they are quite specific in their scope. Furthermore, they do not necessarily provide any relation to the BIM. In contrast, the work in this thesis aims at presenting a more general approach where the connection to the BIM element is a central idea. The smart contract part of the proof-of-concept is meant to be used as part of a change tracking and management system and it only enforces the entities allowed to make contributions. It should be noted that these two approaches are not disjoint, meaning that a system can certainly be envisioned that is general but allows for the development of specific use cases. In fact, the two use cases presented in this thesis are very specific in their scope.

In the presented proof-of-concept, the data is stored off-chain (not on the blockchain) and this bears similarities to other approaches, which were presented in Section 2.10. According to the literature review presented in this thesis, it seems that many researchers argue that the data should be separated from the blockchain's distributed ledger. In the blockchain, only minimal data should be stored, enough to have a proof that some entity performed some task or transaction. This is in line with the original financial application of the blockchain. Lastly, it is worth mentioning that the approach taken in this work to translate the use case into a business process using the BPMN standard might find application in many enterprises. This is mainly because there is research being done to have automatic code generation from the BPMN notation to smart contract code, as presented in Section 2.10.

5.5 Summary

The current chapter begins with an evaluation of the use case presented in the previous chapter and by showing how the research questions are answered. Following that, another use case is presented, called *Routine task management*, which presents a second scenario from facility management. The steps applied in the previous chapter are used again to show how the proof-of-concept is able to handle the requirements of the second use case. Finally, the chapter ends with an evaluation of the proof-of-concept and a mapping of the results to existing research.

CHAPTER 6

Conclusion and Outlook

6.1 Overview

The current chapter provides some final thoughts on the results and discussion from the previous chapters. Firstly, the advantages and disadvantages of using blockchains in the facility management sector are discussed. Next, some future outlook directions are presented. The chapter ends with a conclusion, which also represents the final words of the thesis.

6.2 Revisiting the blockchain properties

Just as any other blockchain-based distributed application, the previously presented proof-of-concept has some security properties that are not obvious, but require some discussion. In Chapter 4, based on the questionnaire to determine if and what type of blockchain is suitable for the given application, it is presented that for the *Issue management* use case the needed blockchain type is a permissioned blockchain. A permissioned blockchain can take many forms, but in any case it requires some method to determine who has access to the network. This method determines some of the security properties of the blockchain itself. In the presented proof-of-concept, there is no specific mention how exactly the nodes are added to the blockchain network. Such details are important, but they depend on the actual implementation. The experimentation that was carried out and is presented in the previous chapter used a local blockchain node. It is mainly concerned with demonstrating the other features of the proof-of-concept and, as a result, does not present any specific method to handle this type of security.

On the subject of security, as presented in the theoretical background (Section 2.2), the way blockchain provides trust on the data it stores is by assuming that at any given point in time the majority of the blockchain nodes will be honest. Compared to other blockchains with thousands of nodes, in cases with just a few blockchain nodes, it is easier for an attacker to change past transactions, because taking over at least half of the nodes in a small blockchain network is technically more feasible. However, it should be stated that changing the state of the blockchain requires a signature from a private key. So, in the presented proof-of-concept, even though the data can be read by the entities that are part of the blockchain, it cannot be altered without a signature from a pre-approved key. The users allowed to make changes are controlled by the smart contract checks, so any change made by a user that has not been previously approved and added to the smart contract would get rejected. This is the case for each of the main operations: raising, updating, and closing issues.

The data privacy of the presented proof-of-concept is discussed, as well. Blockchain does not necessarily have data privacy built-in. In some implementations, such as the Ethereum blockchain, the data, including the value of the smart contract state variables, is public and can therefore be accessed by anyone who has access to the network. Any application on the blockchain needs to consider their data privacy requirements separately and add privacy measures accordingly. The proof-of-concept described in the previous chapter does not specifically address any privacy issues. There are multiple dimensions in which privacy would need to be addressed. For example, the location where the data is stored is a fundamental property. In fact, the distributed nature of the proof-of-concept architecture demands more thought to be placed on this property, compared to a more conventional and centralized architecture. Each participating Ethereum and IPFS node stores some data that, if accessed by an unwanted actor, would represent a privacy breach. One obvious way to handle this is by controlling the nodes in a permissioned way. However, this brings back the need for a trusted authority that will carry out these decisions. Such an authority can take up different forms, such as a third party entity, a group of entities that by voting make such decisions, or possibly a DAO controlling the nodes that enter or leave the network. Once the nodes are added to the network, they still need to provide sufficient data safeguarding techniques, so that the privacy of the data is maintained. The nodes will likely be connected to the Internet, so the privacy of the data that they store must be kept by the their security properties. Another method, that is not mutually exclusive with the first one, is data encryption. By using data encryption, any entity would only be able to decrypt and access the data that they are allowed to access. These privacy properties are left as prospects for future work.

To recall, each time a transaction is placed on the blockchain, it is broadcast to all nodes. In a financial setting, this setup is suitable because it prevents the double-spending problem and the consensus protocol makes sure that there is only one chain of events that will be taken as the valid one. In the facility management sector, however, this might prove to be problematic. Take, as an example, the case when two entities place issue updates on the same BIM element at the same time or very soon one after the other, so that when the second entity places it, the state of the smart contract in the node that it is connected to has not updated with the latest values, yet. On blockchains that use the proof-of-work consensus, this means that the node will mine the second transaction, but the consensus protocol will eventually determine the transactions as invalid. The problem here is that the issue update typically is placed after some physical event in the building has occurred and therefore there might be some data loss, because the physical change would have been performed, but the transaction containing the data referencing that change would be lost. This would be rendered irrelevant by using a concensus protocol where the mining happens very fast or instantaneously. However, this reduces the level of trust that blockchain provides. To explain, if the transactions are added very quickly to the blockchain, then a malicious actor might, with the help of the majority of nodes, change past transactions and redo the mining until it reaches a longer chain of blocks. As a result, this would represent the longest chain and will be taken as the correct one, even though it is, in fact, a tampered history of data. Conversely, if the mining process is kept more difficult, then adding transactions on the blockchain takes a longer time, which in the facility management environment might prove to be disadvantageous, as already discussed. Granted, there exist other consensus protocols that might provide the benefit of speed and, at the same time, do not consume computational resources. It should also be pointed out that the issues of concurrent data generation is not intrinsic to blockchains. Even in a centralized non-blockchain system, updating a single source of data from two different agents simultaneously can lead to data loss. As a result, these are left out-of-scope and as potential future work to be explored and evaluated.

Another characteristic of blockchain that is worth mentioning is the cost of performing transactions. Namely, blockchains were originally envisioned as technologies to be used for financial purposes, and therefore typically have some built-in currency. They use this currency to place costs on transactions. This was left out of scope in the proof-of-concept, because the research questions were concerned with other concepts. However, there are certainly use cases in facility management where the handling of financial transactions would be desired. On the other hand, if such functionality is not needed, a blockchain implementation would be used that has no costs associated with performing transactions.

6.3 Future outlook

As the previous chapters suggest, there are certainly some areas that are interesting for future work. Worth mentioning here is the need to carefully select the appropriate consensus protocols, because current blockchain implementations that use proof-of-work require a large number of participating nodes to guarantee a sufficient level of trust and data security. When the number of nodes is low, then the data stored in the blockchain is more vulnerable. In order successfully change past transactions in the blockchain, an attacker would need to have under control more than half of the nodes, which is technically not difficult in a small network. Additionally, with regard to blockchains, given that recent years have seen many developments in the types of blockchain, including the different consensus protocols that they use, other interesting future work include exploring whether other kinds of blockchains and consensus protocols are more suitable for the AEC industry and the facility management sector. A similarly interesting future work prospect is the integration of payments into these applications, so that all the stakeholders can process payments between each other in a peer-to-peer and trusted manner.

Data security and privacy must be carefully evaluated as well, because the use of blockchain itself does not guarantee those properties. The proof-of-concept reveals that blockchain offers advantages to the actors in the field, especially enterprises and businesses, but careful considerations would need to be placed on these safety aspects. User authentication is a critical component of every system, so one possible future direction can be a comprehensive analysis and study of the best practices of authentication in blockchain-oriented software engineering. The smart contract code can also be prone to errors that might lead to security and privacy issues, which is another direction for further work.

The work presented in this thesis opened up different future work possibilities relating to the data structures, as well. Instead of using custom data structures, as was done here, an interesting topic is to explore how the IFC file formats can be integrated more directly into the blockchain-based applications used in the facility management phase. The advantages of having a direct connection between the IFC file format and the blockchain are multi-fold. Firstly, the usage of industry standards benefits greatly all stakeholders, as it provides the basis for efficient collaboration. Secondly, there is much less overhead in establishing links between the building data and data generated during the facility management processes. Finally, such an integration would allow for an efficient handover between the building phases, where there is little to no data loss.

6.4 Conclusion

This thesis aims to present a proof-of-concept architecture for a system in the context of facility management. The research questions presented in the introduction are concerned with change tracking and management using blockchain and smart contracts. Importance was placed in the data that this system would manage and especially how this data can be efficiently connected to a building information model. Facility management is only one of the stages that buildings go through during their life-cycle, so the data used in all the stages need to be joined. The resulting work described in this thesis presents what are the needed components that would make such a system possible.

In what follows, short summaries are given as answers to the research questions, based on the results of this thesis.

1. How can distributed ledger technology (blockchain) and smart contracts be used for change tracking and management in facility management?

The presented proof-of-concept details a system that captures data produced during different facility management processes and uses blockchain in combination with IPFS as a means to permanently store them. The system's business logic is handled by a smart contract, which makes sure that the system behaves as defined and as expected. The system inherits the blockchain properties to provide an environment where the stakeholders trust that it is practically impossible for anyone to tamper the data.

2. How can the data generated during the facility management processes be joined with the building information model (BIM) on a blockchain-based environment?

The data structures used in the proof-of-concept are built in a way that makes sure that every piece of data stored has a connection to the building information model. Every issue raised and every consequent issue update have a reference to a BIM element, either directly or indirectly. Once an issue is stored in IPFS, the hash pointing to that data is later stored in the consequent update, forming a chain, similar to how blocks are linked with one-another in the blockchain. The result is that if some data is changed, then this can be easily verified, because it is computationally easy to check that the hash is not valid.

The results show the blockchain properties would certainly benefit the facility management sector. For example, the origin and history of the data generated during the facility management phase is something that is bringing some challenges to the industry, as is shown in the literature review in the introduction chapter and related work section. In this sense, the concept of permanent immutable data is of grave interest to the actors in the field of facility management. The presented proof-of-concept describes what it takes to use blockchain and smart contracts to build a facility management change tracking and management system, because blockchain is a technology build around the idea of immutable data. The architecture overview describes the components of such a system and how they interact with one another. The data structures presented provide a method to efficiently store arbitrarily sized pieces of data, which at the same time are linked with one another and with the building information model. This connection to the BIM is a property that is highly beneficial in the facility management environment, because it provides context to every activity. The smart contract controls the business logic, while simultaneously providing verification and checking services, so that the transactions follow a predefined set of rules and requirements. When these components work together, the system is able to provide self-managed change tracking, where the data produced and stored is trusted and permanently accessible, as long as the system exists.

List of Figures

Bibliography

[ACS16] Rajat Agarwal, Shankar Chandrasekaran, and Mukund Sridhar. Imagining construction's digital future. `https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/imagining-constructions-digital-future`, Jun 2016. Accessed on 2020-07-22.

[AEVL16] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30, 2016.

[AP18] Ahmed Alnaggar and Michael Pitt. Towards a conceptual framework to manage bim/cobie asset data using a standard project management methodology. *Journal of Facilities Management*, 11 2018.

[Azh11] Salman Azhar. Building information modeling (bim): Trends, benefits, risks, and challenges for the aec industry. *Leadership and Management in Engineering*, 11(3):241–252, 2011.

[B$^+$14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. 2014.

[Bar15] Hans-Joachim Bargstädt. Challenges of bim for construction site operations. *Procedia Engineering*, 117:52 – 59, 2015. International Scientific Conference Urban Civil Engineering and Municipal Facilities (SPbUCEMF-2015).

[BBK$^+$18] Andre Borrmann, Jakob Beetz, Christian Koch, T. Liebich, and Sergej Muhic. *Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models*, pages 81–126. 09 2018.

[Ben14] Juan Benet. Ipfs - content addressed, versioned, p2p file system, 2014.

[BPRR17] Thomas Beach, Ioan Petri, Yacine Rezgui, and Omer Rana. Management of collaborative bim data by federating distributed bim models. *Journal of Computing in Civil Engineering*, 31(4):04017009, 2017.

[BWM+17] Filipe Barbosa, Jonathan Woetzel, Jan Mischke, Maria João Ribeirinho, Mukund Sridhar, Matthew Parsons, Nick Bertram, and Stephanie Brown. Mckinsey global institute, reinventing construction through a productivity revolution. `https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/reinventing-construction-through-a-productivity-revolution`, Feb 2017. Accessed on 2020-07-22.

[Cra11] Robert Crawford. *Life Cycle Assessment in the Built Environment*, volume 30. 03 2011.

[DGPS+20] Giuseppe Martino Di Giuda, Giulia Pattini, Elena Seghezzi, Marco Schievano, and Francesco Paleari. *The Construction Contract Execution Through the Integration of Blockchain Technology*, pages 27–36. Springer International Publishing, Cham, 2020.

[DLJ19] Theodoros Dounas, Davide Lombardi, and Wassim Jabi. Towards blockchains for architectural design consensus mechanisms for collaboration in bim. pages 267–274, 12 2019.

[Eth21a] Ethereum.org. Ethereum accounts. `https://ethereum.org/en/developers/docs/accounts/`, 2021. Accessed on 09.02.2021.

[Eth21b] Ethereum.org. Ethereum mainnet for enterprises. `https://ethereum.org/en/enterprise/#key-differences`, 2021. Accessed on 09.02.2021.

[Eth21c] Ethereum.org. Intro to ethereum. `https://ethereum.org/en/developers/docs/intro-to-ethereum/`, 2021. Accessed on 09.02.2021.

[Eth21d] Ethereum.org. Introduction to dapps. `https://ethereum.org/en/developers/docs/dapps/`, 2021. Accessed on 09.02.2021.

[Eth21e] Ethereum.org. Introduction to the ethereum stack. `https://ethereum.org/en/developers/docs/ethereum-stack/`, 2021. Accessed on 09.02.2021.

[Eth21f] Ethereum.org. Transactions. `https://ethereum.org/en/developers/docs/transactions/`, 2021. Accessed on 09.02.2021.

[For16] World Economic Forum. Shaping the future of construction: A breakthrough in mindset and technology. `https://www.buildup.eu/en/node/48834`, 2016. Accessed on 2020-07-22.

[Fow04] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[GL10] Ning Gu and Kerry London. Understanding and facilitating bim adoption in the aec industry. *Automation in Construction*, 19(8):988 – 999, 2010. The role of VR and BIM to manage the construction and design processes.

[GTG+17] Ali Ghaffarianhoseini, John Tookey, Amirhosein Ghaffarianhoseini, Nicola Naismith, Salman Azhar, Olia Efimova, and Kaamran Raahemifar. Building information modelling (bim) uptake: Clear benefits, understanding its implementation, risks and challenges. *Renewable and Sustainable Energy Reviews*, 75:1046 – 1053, 2017.

[GZZS10] A. Gerrard, Jian Zuo, George Zillante, and Martin Skitmore. Building information modelling in the australian architecture, engineering and construction industry. *Building Information Modelling and Construction Informatics Underwood*, pages 521–544, 01 2010.

[HD18] Lai Huahui and Xueyuan Deng. Interoperability analysis of ifc-based data exchange between heterogeneous bim software. *JOURNAL OF CIVIL ENGINEERING AND MANAGEMENT*, 24:537–555, 11 2018.

[HF20] Mahmoud Halfawy and Thomas Froese. Modeling and implementation of smart aec objects: an ifc perspective. 07 2020.

[HKJ+18] Lizhen Huang, Guri Krigsvoll, Fred Johansen, Yongping Liu, and Xiaoling Zhang. Carbon emission of global construction sector. *Renewable and Sustainable Energy Reviews*, 81:1906 – 1916, 2018.

[HP17] H. Halpin and M. Piekarska. Introduction to security and privacy on the blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 1–3, 2017.

[ipf21] ipfs.io. Ipfs docs. `https://docs.ipfs.io/reference/`, 2021. Accessed on 09.02.2021.

[ISO16] Industrial automation systems and integration — product data representation and exchange. Standard, International Organization for Standardization, Geneva, CH, March 2016.

[ISO18a] Industry foundation classes (ifc) for data sharing in the construction and facility management industries. Standard, International Organization for Standardization, Geneva, CH, November 2018.

[ISO18b] Organization and digitization of information about buildings and civil engineering works, including building information modelling (bim) — information management using building information modelling. Standard, International Organization for Standardization, Geneva, CH, December 2018.

[KBS17] P. K. Kaushal, A. Bagga, and R. Sobti. Evolution of bitcoin and security risk in bitcoin wallets. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 172–177, 2017.

[KS18] Magdalena Kaminska and Maria Smihily. eurostat - cloud computing - statistics on the use by enterprises. `https://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises`, Dec 2018. Accessed on 2020-07-22.

[Lee19] Wei-Meng Lee. *Using the MetaMask Chrome Extension*, pages 93–126. Apress, Berkeley, CA, 2019.

[LGK19] Jennifer Li, David Greenwood, and Mohamad Kassem. Blockchain in the built environment and construction industry: A systematic review, conceptual models and practical use cases. *Automation in Construction*, 102:288 – 307, 2019.

[LJOD19] Zhen Liu, Lijun Jiang, Mohamed Osmani, and Peter Demian. Building information management (bim) and blockchain (bc) for sustainable building design information management framework. *Electronics*, 8:724, 06 2019.

[LKCB19] Jennifer Li, Mohamad Kassem, Angelo Ciribini, and Marzia Bolpagni. A proposed approach integrating dlt, bim, iot and smart contracts: Demonstration using a simulated installation task. 07 2019.

[LKW20] Jennifer Li, Mohamad Kassem, and Richard Watson. A blockchain and smart contract-based framework to increase traceability of built assets. pages 347–362, 08 2020.

[LL20] Samir Lemeš and Lamija Lemeš. Blockchain in distributed cad environments. In Isak Karabegović, editor, *New Technologies, Development and Application II*, pages 25–32, Cham, 2020. Springer International Publishing.

[LLS$^+$15] Yujie Lu, Yongkui Li, Miroslaw Skibniewski, Zhilei Wu, Runshi Wang, and Yun Le. Information and communication technology applications in architecture, engineering, and construction organizations: A 15-year review. *Journal of Management in Engineering*, 31(1):A4014010, 2015.

[LNVD11] Albertus Laan, Niels Noorderhaven, Hans Voordijk, and Geert Dewulf. Building trust in construction partnering projects: An exploratory case-study. *Journal of Purchasing and Supply Management*, 17(2):98 – 108, 2011.

[MMB13] A. Mahamadu, L. Mahdjoubi, and C. Booth. Challenges to bim-cloud integration: Implication of security issues on secure collaboration. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 209–214, 2013.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[NR19a] Nawari Nawari and Shriraam Ravindran. Blockchain technology and bim process: review and potential applications, 05 2019.

[NR19b] Nawari O. Nawari and Shriraam Ravindran. Blockchain and building information modeling (bim): Review and applications in post-disaster recovery. *Buildings*, 9(6), 2019.

[NR19c] Nawari O. Nawari and Shriraam Ravindran. Blockchain and the built environment: Potentials and limitations. *Journal of Building Engineering*, 25:100832, 2019.

[NSA$^+$19] Nishara Nizamuddin, Khaled Salah, Muhammad Azad, Junaid Arshad, and Muhammad Habib ur Rehman. Decentralized document version control using ethereum blockchain and ipfs. *Computers and Electrical Engineering*, 76, 03 2019.

[OAYY16] Gozde Ozturk, David Arditi, Ibrahim Yitmen, and Mehmet Yalcinkaya. The factors affecting collaborative building design. *Procedia Engineering*, 161:797–803, 12 2016.

[Peh19] David Peherstorfer. Bim and blockchain; a decentralized solution for a change management workflow in construction projects, 2019.

[PG07] Thomas Porter and Michael Gough. Chapter 6 - confirm user identity. In Thomas Porter and Michael Gough, editors, *How to Cheat at VoIP Security*, How to Cheat, pages 159 – 184. Syngress, Burlington, 2007.

[PMC17] Mario Pérez-Montoro and Lluís Codina. Chapter 3 - designing user experience. In Mario Pérez-Montoro and Lluís Codina, editors, *Navigation Design and SEO for Content-Intensive Websites*, pages 65 – 84. Chandos Publishing, 2017.

[PRL20] Julien Polge, Jérémy Robert, and Yves Le Traon. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 2020.

[RHAW12] Alan Redmond, Alan Hore, Mustafa Alshawi, and Roger West. Exploring how information exchanges can be enhanced through cloud bim. *Automation in Construction*, 24:175 – 183, 2012.

[SCR$^+$11] Sarah Slight, Kathrin Cresswell, Ann Robertson, Guro Huby, Tony Avery, and Aziz Sheikh. The case study approach. *BMC medical research methodology*, 11:100, 06 2011.

[SEPS19] Marjan Sadeghi, Jonathan Elliott, Nick Porro, and Kelly Strong. Developing building information models (bim) for building handover, operation and maintenance. *Journal of Facilities Management*, 17, 06 2019.

[SFM$^+$20] Alireza Shojaei, Ian Flood, Hashem Izadi Moud, Mohsen Hatami, and Xun Zhang. An implementation of smart contracts by integrating bim and blockchain. In Kohei Arai, Rahul Bhatia, and Supriya Kapoor, editors, *Proceedings of the Future Technologies Conference (FTC) 2019*, pages 519–527, Cham, 2020. Springer International Publishing.

[SKT12] Azhar Salman, Malik Khalfan, and Maqsood Tayyab. Building information modeling (bim): Now and beyond. *Australasian Journal of Construction Economics and Building*, 12:15, 12 2012.

[Som11] Ian Sommerville. *Software Engineering*. Addison-Wesley, 2011.

[SwF19] Alireza Shojaei, jun wang, and Andriel Evandro Fenner. Exploring the feasibility of blockchain technology as an infrastructure for improving built asset sustainability. *Built Environment Project and Asset Management*, ahead-of-print, 11 2019.

[TK17] Ziga Turk and Robert Klinc. Potentials of blockchain technology for construction management. *Procedia Engineering*, 196:638 – 645, 2017. Creative Construction Conference 2017, CCC 2017, 19-22 June 2017, Primosten, Croatia.

[TLW18] A. Tran, Qinghua Lu, and I. Weber. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In *BPM*, 2018.

[WB13] Karl E Wiegers and Joy Beatty. *Software Requirements 3*. Microsoft Press, USA, 2013.

[WG18a] F. Wessling and V. Gruhn. Engineering software architectures of blockchain-oriented applications. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 45–46, 2018.

[WG18b] K. Wüst and A. Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54, 2018.

[Whi04] Stephen A White. Introduction to bpmn. *Ibm Cooperation*, 2(0):0, 2004.

[XL20] Fan Xue and Weisheng Lu. A semantic differential transaction approach to minimizing information redundancy for bim and blockchain integration. *Automation in Construction*, 118:103270, 2020.

[Yal17] Mehmet Yalcinkaya. *Understanding the Technical and Cognitive Challenges, and Closing the Gaps in Architectural, Engineering, Construction-Facility Management (AEC-FM) Standards.* PhD thesis, 12 2017.