INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Distributed Control for Building Automation with Smart Contracts

Bernardo Chastre Lopes

Master in Telecommunications and Computer Engineering

Supervisor:
PhD Rui Neto Marinheiro, Associate Professor
Iscte - Instituto Universitário de Lisboa

November, 2022

TECHNOLOGY
AND ARCHITECTURE

Department of Information Science and Technology

Distributed Control for Building Automation with Smart Contracts

Bernardo Chastre Lopes

Master in Telecommunications and Computer Engineering

Supervisor:
PhD Rui Neto Marinheiro, Associate Professor
Iscte - Instituto Universitário de Lisboa

November, 2022

Abstract

There are some interesting, centralized solutions for managing smart buildings, whether opensource or not. However, there is the need for some of these functions to be decentralized. Decentralization calls for consideration of security and trust standards that enable a coordinated approach. This dissertation aims to fill the knowledge and research gaps that still exist in this field.

As such, the objective of this dissertation is to integrate smart contracts with an IoT platform for distributed control in smart buildings. The proposed solution aims to grant control of a function or the monitoring data in a given smart building to an external entity that can manage this service.

To achieve the goal of this dissertation, an integration was proposed with opensource technologies. Two of these technologies are Daml and OpenHAB, and the created SCIApp application enables their communication.

Functional tests, confirm that it is possible to achieve the proposed integration. Response time in the order of seconds was obtained, with an average value of 4061ms. Performance tests, allow to verify the response time for different loads. Results confirm that response time remains constant when new contracts are created every 20s. For the remaining frequencies, the response time increases.

This dissertation leads to the conclusion that it is feasible to integrate smart contracts with IoT to control and manage functions of intelligent buildings. By analyzing the tests conducted on the developed system, it was observed that the control is possible for operations that do not require a real-time response time.

Keywords: Smart contracts; internet of things; home automation.

Resumo

Existem algumas soluções centralizadas interessantes para a gestão de edifícios inteligentes, sejam elas open-source ou não. Contudo, existe a necessidade de algumas destas funções serem descentralizadas. A descentralização exige a consideração de normas de segurança e confiança que permitam uma abordagem coordenada. Esta dissertação visa preencher as lacunas de projectos existentes neste campo.

Como tal, o objectivo desta dissertação é integrar contratos inteligentes com uma plataforma IoT para controlo distribuído em edifícios inteligentes. A solução proposta visa conceder o controlo de uma função ou monitorização de dados num determinado edifício inteligente a uma entidade externa que possa gerir este serviço.

Para atingir o objectivo desta dissertação, foi proposta uma integração com tecnologias de código aberto. Duas destas tecnologias são Daml e OpenHAB, e a aplicação SCIApp criada permite a sua comunicação.

Testes funcionais, confirmam que é possível alcançar a integração pretendida. Foram obtidos resultados com a duração de vários segundos, sendo a duração média típica dos testes de 4061 ms. Os testes de desempenho, permitem verificar o tempo de resposta. Os resultados confirmam que o tempo de resposta com a frequência de 20s é constante. Para as restantes frequências, o tempo de resposta aumenta.

Esta dissertação leva à conclusão de que é viável integrar contratos inteligentes com IoT para controlar e gerir funções de edifícios inteligentes. Ao analisar os testes realizados no sistema, observou-se que o controlo é possível para operações que não requerem um tempo de resposta em tempo real.

Palavras-chave: Contratos inteligentes; internet das coisa; domótica.

Acknowledgements

I would like to express my very great appreciation to Professor Rui Neto Marinheiro, my supervisor, for the patient guidance, support and useful critiques of this dissertation project and also Instituto de Telecomunicações (IT).

I thank my family for their encouragement and constant motivation. Without them it would be impossible for me to achieve what I have been achieving. Without the support and opportunities they have given me none of this would be possible.

I would also like to thank my girlfriend, Susana, who has been on this journey with me from the beginning. Who not only supported me in moments of more uncertainty and frustration, but also helped me to overcome them and, in a way, in the revision of this dissertation.

Finally, i would like to thank my friends, specifically Pedro, for their constant encouragement and support.

Contents

Abstract i

Resumo iii

Acknowledgements v

List of Figures xi

List of Tables xiii

Abbreviations xv

Glossary xvii

Chapter 1. Introduction 1

1.1. Context 1

1.2. Motivation 2

1.3. Research Questions 3

1.4. Objectives 3

1.5. Proposal 3

1.6. Research Methodology 4

Chapter 2. Literature Review 7

2.1. Internet of Things 7

2.2. Smart Contracts 12

2.3. Combination of IoT and Smart Contracts 15

Chapter 3. Proposed Architecture 19

3.1. Architecture Building Block 19

3.2. Smart Contract Integration 22

3.3. Architecture Flow Control 23

Chapter 4. Implementation 27

4.1. Testbed Setup 27

4.2. Smart Contracts 29

4.3. DAML 31

4.4. SCIApp 37

4.5. OpenHAB 38
4.6. Communications 40
4.7. Control Modifications 44

Chapter 5. Results and Tests 45
5.1. Functional Test 45
5.2. Performance Tests 50

Chapter 6. Conclusion 55
6.1. Future Work 56

References 57

Appendix A. Synchronization Setup 63

Appendix B. Daml 65
B.1. Installation 65
B.2. Smart Contract's Code 65

Appendix C. PostgreSQL Installation 67

Appendix D. IntelliJ Installation 69

Appendix E. SCIAPP Code 71

Appendix F. OpenHAB Installation 73

Appendix G. Basic UI Code 75

Appendix H. Mosquitto Setup 77

Appendix I. Article 79

Code Listings

1 Configuration files for connecting Participant 1 and 2 to Domain. 33
2 Generate the RSA keys. 34
3 Launch Domain Node. 35
4 Launch Bob Node. 36
5 Create .dar file. 36
6 Create the party. 36
7 JWT token format. 37
8 Launch JSON API. 37

List of Figures

1 Design science research methodology [18]. 5

2 IoT application domains. 7

3 OpenHAB Architecture [31]. 11

4 OpenHAB Communication Protocols Architecture. 12

5 Smart Contracts Use Cases. 13

6 Smart Contract applications in Internet of Things. 15

7 Communication between stakeholders. 19

8 Architecture proposal. 20

9 Smart Contract life-cycle. 22

10 Database architecture. 23

11 Interaction between the two nodes. 24

12 Technological Options of the Implementation. 28

13 Smart Contract Architecture. 29

14 Smart Contract Architecture. 30

15 Canton Architecture. 31

16 Configuration file for Participants 1 and 2. 33

17 Configuration file for Domain. 34

18 JWT Token. 35

19 Bob User Interface. 39

20 Alice User Interface Without Control. 39

21 Alice User Interface With Control Granted. 40

22 MQTT Connection. 41

23 Configuration of the MQTT Broker in OpenHAB. 41

24 Configuration of the MQTT Broker in OpenHAB. 42

25 Configuration of the MQTT Thing in OpenHAB. 42

26 Configuration of the Item linked to the Thing. 43

27 Item linked to the Thing. 43

28 Time spent in each system block. 46

29 Average time to reach the next application. 49

30 The moving average of the time it takes for a contract to become available on Alice’s node, for the last 5 contracts, without delay. 51

31 Delay Settings (a) and Delay Connections (b). 52

32 The moving average of the time it takes for a contract to become available on Alice’s node, for the last 5 contracts, with delay. 53

33 The moving average of the time it takes for a contract to become available on Alice’s node, for the last 5 contracts, with and without delay. 54

List of Tables

1	Comparison between IoT platforms.	10
2	Comparison between Smart Contract Languages.	13
3	Specifications of Host and Virtual Machines.	27
4	Delay time between VMs.	29
5	Average time of the UPDATE and GET test.	48
6	Average time spent on each application.	50
7	Average time of upstream and downstream connections.	50

Abbreviations

API - Application Programming Interface.

DAML - Digital Asset Modeling Language.

GDPR - General Data Protection Regulation.

HTTP - Hypertext Transfer Protocol.

IoT - Internet of Things.

JSON - JavaScript Object Notation.

MQTT - Message Queuing Telemetry Transport.

NAT - Network Address Translation.

REST - Representational State Transfer.

SCIApp - Smart Contract Integration Application.

Glossary

Bluetooth - Bluetooth is a wireless communication and data transfer protocol.

Channel - A channel represents a certain functionality of a thing and can be associated with an item.

Daml Ledger - The term ”Daml Ledger” refers to a distributed ledger system that uses Daml smart contracts and exposes the APIs for the Daml Ledger.

Domain - In Daml, participant nodes communicate with one another by sending end-to-end encrypted communications across the domain.

Entity - Participants of the system solution.

External Entity - Alice.

Item - In OpenHAB represents all properties and capabilities of the user's home automation. Items are basic data types and have a state which can be read from, or written to. Items can be linked to a Binding channel for interaction with the outside world.

Node - The system of each entity.

Participant node - In Daml, a participant node is a server that offers users reliable programmatic access to a ledger through the Ledger API.

Party - A party in Daml represents a person or legal entity.

Rules - Rules are used to automate processes.

Sitemap - Sitemaps are one method for selecting and composing items and things into a user-oriented representation for different User Interfaces (UIs).

Smart Building's Entity - Bob.

Synchronization Technology - The database or blockchain that Daml uses for synchronization, messaging and topology.

Thing - Represents the physical layer of an OpenHAB system. From a configuration standpoint, Things tell openHAB which physical entities (devices, web services, information sources, etc.) are to be managed by the system. Things are connected to openHAB through bindings. Each Thing provides one or more Channels to access its functionality. These Channels can be linked to items. Items are used to control Things and consume their information. **Wi-Fi** - Wi-Fi is a wireless communication and data transfer protocol.

ZigBee - ZigBee is a wireless communication and data transfer protocol.

Zwave - ZWave is a wireless communication and data transfer protocol.

CHAPTER 1

Introduction

1.1. Context

Nowadays, with all the environmental concerns, related to climate change, it is increasingly important to use renewable energy and minimize energy waste [1]. New technologies arise to make energy use more efficient. Current and old buildings, all over the world, are adopting new technologies in order to make them more energy efficient and beyond [2]. A structure that uses automated processes to automatically control the building's operations [1] is called a smart building. This intelligent structure relies on sensors, controllers and actuators [3], to efficiently automate the building. After a sensor collects essential data, the controller analyses it, and commands the actuator to act [3]. The network that connects all these smart devices is called the Internet of Things (IoT) and it is one of the most important technologies of this century [4].

With IoT, a network of sensors and interconnected devices, data can be collected and shared to provide more comfort, safety, and efficiency for human beings on smart buildings, roadways, healthcare, industry [5], and in many other areas. IoT technology has grown a lot in the last years and according to recent reports, 17.3 billion IoT devices will be connected in 2023 [6]. Nowadays, buildings architecture plans are made to optimize its operations efficiency, aiming to reduce energy consumption and increase the comfort of its residents [3]. To achieve this goal in smart buildings, the devices interconnected by IoT must be well managed.

Traditionally the management and control of several smart building functions, including energy management, are centralized [7]. There has been a need for more of these functions to have their management delegated to external entities. Regarding distributed management of electricity production and consumption, the delegation of management functions has not been easy. This happens because each building usually has an autonomous and centralized control. It is required to create a system to trust external entities to manage the functions of a smart building. Smart Contracts technology allows anonymous transacting parties to trust each other without intermediaries [8].

A Smart Contract is a script, programmed by a developer and deployed across a distributed system [9]. Smart Contracts are simple immutable contracts written in code that are executed when certain conditions are met [9]. The contract auto executes itself when the terms of the contract, agreed upon two parties, are achieved [10]. They are commonly used to automate the execution of an agreement, without the need for an intermediary intervention [10]. There are several smart contract languages

such as Daml[1] and Solidity[2]. Depending on the language, smart contracts can be stored in blockchains or databases. As databases and blockchains can be integrated with Daml, smart contracts written in Daml can be stored in both of these types of storage. Solidity-based smart contracts, on the other hand, can only be stored in blockchains.

The Blockchain (BC) technology is a "decentralized, shared, and immutable database ledger" that stores a registry of assets, transactions, contracts, and events across a peer-to-peer (P2P) network [11].This Distributed Ledger Technology (DLT) was firstly introduced by cryptocurrencies [4] and has been promising in several scenarios including energy trading in smart buildings [12]. In other words, blockchain consists in a chain of blocks, linked together, called ledger [4], that is stored across a P2P network. A block contains a several number of transactions, occurred in the network, between two distinct parties [11]. A BC system eliminates the need of trusting a centralized authority, i.e., a third-party, to execute a transaction between two parties on a network [13]. A way of automating transactions on a blockchain is through codified rules integrated into a Smart Contract. Food supply chain management, energy market management and cryptocurrencies are examples of such applications of the integration of BC with Smart Contracts where data distribution and sharing among decentralized infrastructures is required [14].

Although blockchain is one of the most popular approaches today, databases can be used to store developed smart contracts in situations where the issue of trust is not as problematic. Despite smart contracts can be stored in databases, blockchain provides a more secure solution.

Thus, creating a system that integrates Smart Contract technology with IoT can be beneficial to solve trust and security problems [15] and allows the automated functioning of IoT processes in smart buildings [15]. This system is supposed to solve the problem of trust in the delegation of building control functions.

1.2. Motivation

As known, the building sector has a major contribution to the energy consumed worldwide [2]. With IoT technology, smart buildings were created assisting the need for energy efficiency, automated processes, improving comfort, and the simplification of building management [16]. Normally, the management and control of various functions in a smart building, related to electricity production and consumption, is centralized. Sometimes these functions are delegated to external entities, with the objective of making them more efficient. However, a trust-related problem arises, linked to the delegation of these functions to external entities.

Therefore, it is necessary to create a system that controls and verifies the transactions of the control functions, in order to solve the delegation trust in external entities. A Smart Contract-based system offers a solution able to overcome this obstacle, because of its specificity. This issue can be overcome with this technology because of its tamper-proof, security, reliability, automation, and

[1] https://daml.com/
[2] https://docs.soliditylang.org/en/v0.8.17/

trustworthiness characteristics [17]. As a result, there is no longer a need for a system operator or an intermediary because Smart Contracts allow anonymous transacting parties to trust each other [17]. Negotiations and contracting between the parties have the potential to be automated and accelerated [17]. Thus, ”Smart contracts offer a virtual means of reaching and enforcing a credible binding agreement and/or transaction” [17].

1.3. Research Questions

Many smart buildings already use IoT with the goal of making energy consumption more efficient. However, in the vast majority of buildings, the delegation of control functions is centralised, and Smart Contract technology is seen as a mechanism to solve this problem. The main question that this work intends to answer is:

- Is it feasible to integrate Smart Contract in order to achieve decentralised control of smart buildings, regarding energy management?

The following questions may answer the feasibility of this work:

- Is the response time too excessive, in a system that integrates Smart Contract technology with IoT, especially in the control of electricity production and consumption in smart buildings?
- Is it possible to implement functionalities that require interactivity in this system without worrying about the delay?
- Is this type of integration viable in small single-board computers with the complexity and processing that this system requires?

1.4. Objectives

The major objective of this work is to integrate Smart Contract technology with an IoT platform for distributed control in smart buildings. The purpose of this integration aims to address the problem of trust in the delegation of building control functions in an open way to control and verify management function transactions.

1.5. Proposal

In the solution proposed by this work, the Smart Contracts Application Daml will be integrated with the IoT platform OpenHAB in order to grant the control of a smart building to external entities. It will be demonstrated how the entity in charge of the smart building will delegate control to the external entity. To this end, functional and performance tests will be carried out. When the entity that manages the smart building grants control, the external entity will be allowed to change the condition of a light in the smart building.

Functional tests were carried out to determine whether it was possible to integrate smart contracts with IoT for distributed control of intelligent buildings. After running the tests, we come to the conclusion that this integration is feasible. It was also concluded from these tests that the proposed system is appropriate for functions that do not require an immediate response, such as turning on

a washing machine or determining the status of an item. However, for functionalities that require immediate response or feedback, such as turning on and off a light, the response time is too long for a user to wait for the light to change. As a result, interactive features cannot be implemented.

Performance tests were also conducted, and the results revealed that the response time is very high when a large number of contracts are created with a predetermined time frequency. It is also possible to conclude that the system is not scalable as the response time increases.

The main contributions of this dissertation are:

- The integration of smart contracts in building control using a database.
- The management through this kind of integration for functionalities that don't require real time responses.
- The possibility of implementing this type of solution using open source platforms such as Daml and OpenHAB.

The current work is organized as follows:

- Chapter 2 is divided in 3 sections and presents a literature review on the coexistence of IoT with buildings, regarding the delegation of control of building functions, as well as the integration of smart contracts to rely on external entity control. In the first section, the IoT and its special application in smart buildings are studied, along with open source platforms. The second section studies the smart contract technology and a few programming languages that are employed. The third section studies the combination of Smart Contracts with IoT.
- Chapter 3 describes this work proposed solution and it is divided in three sections. The first section defines the stakeholders. The second section presents the overall architecture. The third section explains where the data is stored.
- Chapter 4 presents the implementation for the proposed solution. It is divided in four sections. These include the installation of the operating system on the virtual machines chosen; setting up the OpenHAB platform, API and Daml application; implementing the methods to integrate these three systems.
- in Chapter 5 is described the tests and validations for this system's design and implementation.
- Chapter 6 addresses conclusions and some relevant future work aligned with the current proposal.

1.6. Research Methodology

The research methodology used for the problem-solving in this project is Design Science Research (DSR) [18]. This methodology is composed by the following six activities represented in figure 1:

Problem Identification: The problem detected was that generally, each smart building has an autonomous and centralised control, which makes it difficult to delegate management functions.

Define objectives of a solution: The main goal of this dissertation is to develop a decentralized management system for smart buildings, to solve the problem of trust in the delegation of building control functions, regarding the production and consumption of electricity.

Design and Development: Create a decentralized, trusted, and tamper-proof system that provides irrefutable mechanisms to control and verify management function transactions. It can be accomplished using Smart Contract technology integrated with an smart building management platform.

Demonstration: The first implementation of this system will be a prototype, to connect the Smart Contract application with a smart building management platform and it will have the purpose of turning on and off a light in a room of the smart building.

Evaluation: After the implementation, some results can be concluded about the efficiency and viability of the system.

Communication: The last activity will be the presentation of the obtained results in a dissertation and an article.

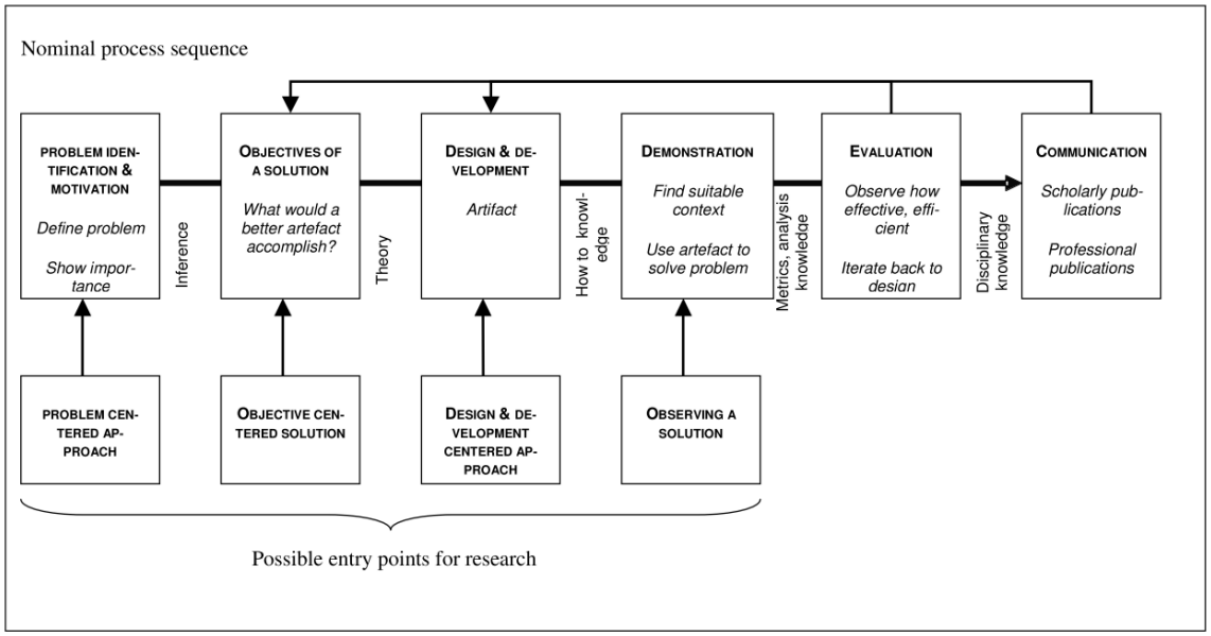


Figure 1. Design science research methodology [18].

CHAPTER 2

Literature Review

This chapter covers the state of art of how to integrate smart contract technology with an IoT platform for a distributed control of smart buildings. Section 2.1 discusses the topic of Internet of Things relating to smart buildings and IoT platforms. The topic of smart contracts is covered in section 2.2. In section 2.3 is addressed the combination of IoT with smart contracts.

2.1. Internet of Things

Although IoT technology is widely used nowadays, its concept has been around for a while. In the late 1970s, the idea of connecting computers and networks to control and monitor devices was firstly introduced by monitoring the meter, on an electric grid, via telephone lines [19]. As a natural evolution of the Internet [4], IoT can be defined as a network of intelligent devices called "things" that communicate with each other, via wired or wireless connection [19]. These devices, which contain sensors and/or actuators, collect, analyse, and share data with other devices, programs, and platforms [20]. This data can be used to monitor and interact with various equipment, as well as to provide better planning, control and coordination of a system [20]. These connected things can improve autonomy, communication and facilitate knowledge sharing [20] which helps to create smart solutions. This leads to greater efficiency in the performance and productivity of various systems that enhance the quality of life. IoT has emerged as a critical technology, and any smart system must incorporate connected intelligent devices [16]. Some of its applications are represented in figure 2 [21].

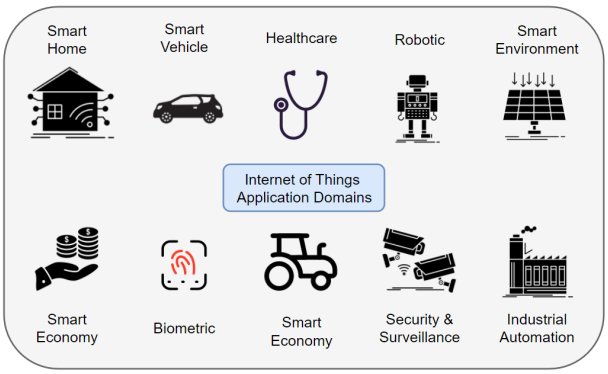


Figure 2. IoT application domains.

In Figure 2, Smart Homes, Healthcare, Smart Vehicles, and Agriculture are the most important and relevant applications. This dissertation falls under the category of smart homes. In addition to

the applications shown in figure 2, the Internet of Things has a significant impact on smart grids and smart cities.

2.1.1. Smart Buildings

Smart buildings are one of the many applications of IoT. The general public defines this concept as the idea of automating processes performed in a building [1]. An smart building can control its operations automatically with the help of sensors, controllers, and actuators [1]. Supported by automated control, a building that integrates smart things can help to reduce energy consumption, improve resident comfort and safety, control air conditioning automatically, which increases the building's efficiency and leads to cost savings [3]. Some researchers define a smart building as a "multidisciplinary effort to integrate and optimize the building structures, systems, services and management in order to create a productive, cost-effective and environmentally approved environment for the building occupants" [22]. As the building sector represents 40% of the total energy consumed in developed countries, energy efficiency is one of the most important research topics, and not only in the development of smart buildings [16]. Therefore, controlling and monitoring an smart building with integrated devices is efficient, useful, profitable, and reliable [3].

The system created in [1] has the purpose of reducing energy wastage in smart buildings. To accomplish this goal an energy consumption predictive model was developed. In [23] a solution is presented regarding smart buildings for implementing an energy consumption monitoring system using an open-source IoT infrastructure, aiming specifically for educational buildings. The model introduced in [24] presents a form for improving energy sustainability in smart buildings. This solution provides an integrated optimisation model for conventional heating and HVAC systems to achieve energy-balance requirements. This model demonstrates the effectiveness of the smart BEMS (Building Energy Management Systems) by comparing it to the conventional BEMS.

Another problem is to reuse an old building to implement smart solutions. The authors of [3] propose an approach for smart retrofit of buildings. Old buildings, containing obsolete or even absent equipment, are a challenging scenario to deploy IoT networks and devices. This solution aims to ensure interoperability between different IoT devices in retrofit environments and to adopt efficiency measures.

Typically, the management and control of the operations in an smart building are centralised, which implies the existence of a single point of failure. There has been an increasing need for the management of these operations to be delegated to external entities.

An example is a distributed management of electricity production and consumption in self-sustainable communities [12]. The problem of trust, in the delegation of building management functions arises. In order to solve this problem, it is necessary to create a system that provides irrefutable mechanisms to control and verify the transactions of the management functions/operations in an smart build. A new technology known as Smart Contracts provides the tools required to overcome the issue of trust in external entities.

In [25] it is proposed to use smart contracts and blockchain technology to create a secure and automated decentralized renewable energy trading platform within the microgrid for Smart Homes. The microgrid is made up of nodes that represent smart homes, and each node is an energy consumer and/or producer. This system trades energy using pre-programmed smart contracts on a private blockchain.

In [26], a decentralized framework for managing electrical consumption in a community of Smart Buildings is presented. Through a series of local optimization processes, Smart Contracts technology enabled participants to collaboratively decide on a planning profile that minimizes the overall aggregated cost.

A distributed demand side management system among multiple homes in a community microgrid, with the integration of an IoT smart meter and the presence of renewable energy sources is described in [27]. The proposed scheme is distributed on blockchain, which provides a trusted communication medium between the participants. It enforces the autonomous monitoring of smart appliances and the smart contract-based charging of electricity use. To enable transaction execution in the smart community without the intervention of a third party, Solidity smart contracts are used. The adoption of smart contracts and blockchain technology allows participants to exchange energy and build trust among users or organizations. The presented results show that the total cost of energy consumption for the entire community, as well as the individual cost for each user, is reduced. The management and control of smart buildings cannot be possible without the use of an IoT platform.

2.1.2. Open-Source Software for IoT

Open-source software is usually easy to adopt for implementing IoT solutions [28]. Because it is licensed as open and free, it can be used without restrictions to monitor devices and automate actions in an IoT environment [28]. Open-source software usually, has its own documentation, and with practice, it is possible to create solutions to daily problems [28]. If there is an extensive community behind an open-source software, it usually supports everyone with any issue and provides an answer, which helps to accelerate the resolution of a problem [28]. This does not apply if there is no large community. In order to choose the best open-source software for IoT, the following parameters should be considered: cost effective, system security, sufficient clear documentation, and restrictions [28].There are many different IoT platforms adapted for automation in Smart Home and Smart Building applications. Some examples include OpenHAB[3], Home Assistant[4] [29], IBM Maximo Application Suite[5], which may be useful to integrate with Smart Contracts in an energy management system. On table 1 all the relevant IoT platforms found where summarized, from the analyzes that has been made for the present dissertation.

So, for this work, the decision to be taken was between OpenHAB and Home Assistant . OpenHAB is an open-source solution and stands for “Open Home Automation Bus”. It is a mature software

[3]https://www.openhab.org/

[4]https://www.home-assistant.io/

[5]https://www.ibm.com/products/maximo

Table 1. Comparison between IoT platforms.

	OpenHAB	Home Assistant	IBM Maximo Application Suite
Open-Source	Yes	Yes	No
Interoperability among devices	Yes	Yes	Yes
Web-Based GUI	Yes	Yes	Yes
Community	Large	Large	Medium

that is extremely stable, packed with functionality and offers a lot of features. In April of 2010, the first build was released [30]. Because of its age and the number of developers who have worked on it, OpenHAB has high quality and complete documentation [30]. Due to the constant migration of developers and users to newer software, the OpenHAB community is relatively stagnant [30]. Because OpenHAB has such extensive and rich documentation, users rarely need to seek assistance. However, if a problem not described in the documentation arises, the community is still available to assist [30]. This platform allows to support multiple protocols of communications and ensures connectivity between things from different manufacturers [29]. As an independent home automation platform, OpenHAB allows to integrate intelligent devices and other automating systems in one user-friendly interface easy to understand [29]. It provides a flexible solution and automation rules for the desired system [29]. This platform delivers an IoT-based infrastructure [23] that will help to monitor the energy consumed and produced, and manage the functions of smart buildings, among many other things.

Home Assistant is an open-source home automation software that prioritizes local control and privacy. The first build was released in January 2014 [30]. At the beginning of the dissertation it was noticed that this software lacks complete and detailed documentation due to its early development stage, but today, this is not true. As a direct consequence, users may have difficulty understanding some concepts and must often rely on a hit-and-trial approach to achieve their goals [30]. Due to its early development stage, Home Assistant, currently, has a growing and active community, which compensates for the lack of documentation. Because the community is so active, users can get help quickly if they have any problems [30]. When the two IoT platforms are compared, it is clear that OpenHAB has more documentation and features to explore, making it more appealing to implement. As a result, OpenHAB was chosen as the smart building management platform for this dissertation. When the dissertation began, the home assistant solution was not as stable or mature as OpenHab. Today, Home Assistant would be an equally viable alternative to OpenHAB. The next paragraph demonstrates an example of OpenHAB's usability.

The authors of [28] propose an energy monitoring system using an open-source IoT platform and a Raspberry Pi 3. This project intends to measure the energy consumed in one month in a school building. OpenHAB platform allows to connect the things, used to measure the energy consumed, to each other and to the platform, in order to manage them. Through a user interface it is possible to add new things, monitor, and control them.

The system proposed in [29] intends to improve interoperability between devices that make up the IoT, in a smart building environment. OpenHAB provides a solution that allows multiple protocols of communications and ensures connectivity between objects from different manufacturers. This system enables remote control of connected objects via a voice assistant or a Web application and also has a view of the status of each connected object via a web application.

OpenHAB platform is used as a communication and integration technology for monitoring systems. The OpenHAB project aims to provide a universal integration platform for all things home automation [31]. Figure 3 depicts the OpenHAB platform's communication architecture.

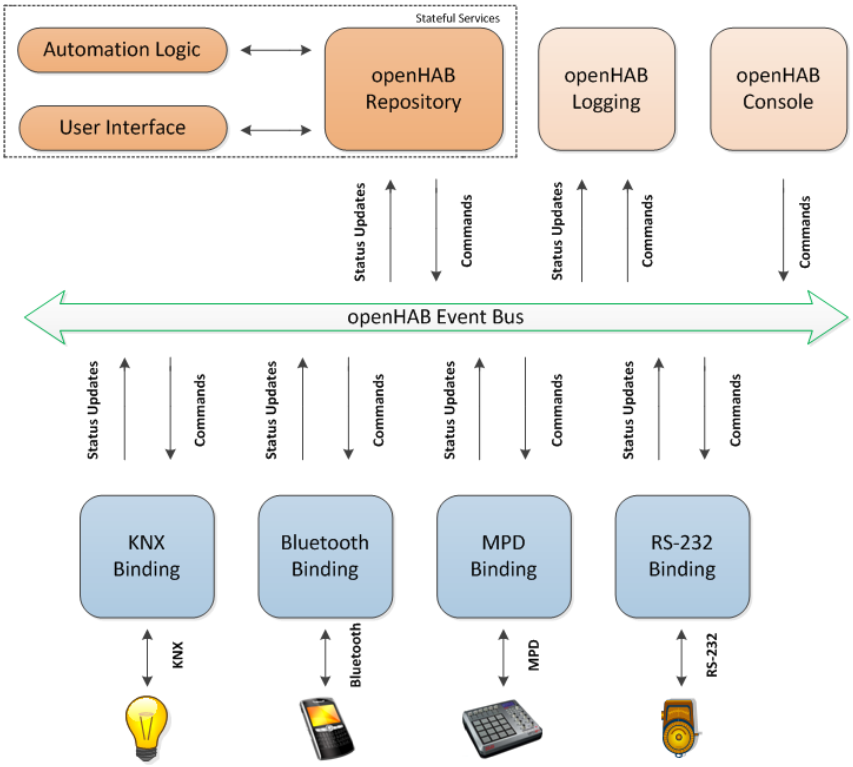


Figure 3. OpenHAB Architecture [31].

Bindings can be used to implement the interaction of OpenHAB with various actuators or systems. Bindings are optional packages that can be utilized to extend OpenHAB's functionality [31]. Bindings allow devices to communicate with OpenHAB. Because bindings may be created for each device's communication method, we can say that OpenHAB is scalable. Figure 4 illustrates the OpenHAB protocols implemented in this dissertation.

The HTTP bindings can be used to communicate with end stations. This binding can also control the conditions and status of items in OpenHAB [31]. OpenHAB uses the Rest API (Representational State Transfer Application Programming Interface) mainly for system communication. It allows to access required information or status updates, but also to send the commands to remote nodes [31]. The OpenHAB runtime uses REST API to communicate via HTTP protocol.

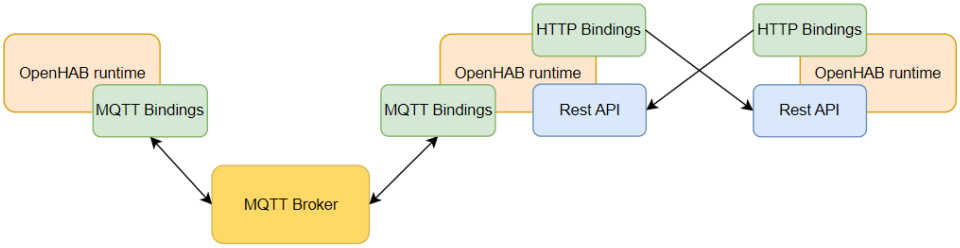


Figure 4. OpenHAB Communication Protocols Architecture.

The MQTT binding does not provide a broker. Mosquitto was the broker chosen for this work. The Broker was configured in a particular ip and port and authentication was included for security reasons. This binding was used to communicate the status of the items in OpenHAB. The OpenHAB runtime uses Mosquitto Broker to communicate via MQTT protocol. These are just two possible ways of communication, OpenHAB is flexible enough to use other approaches.

2.2. Smart Contracts

The concept of ”using computer code in order to automate legal contracts while using cryptography to make them secure and tamper-proof” was firstly introduced by Nick Nazbo in 1994 [17]. This technology consists of programmable and self-executed scripts [10] that control transactions under particular conditions [32]. The terms and conditions of the smart contract are written into lines of code [10]. The execution of the contract is controlled by the code, and transactions are trackable and irreversible. The smart contract auto executes itself when the terms of the contract, agreed upon two parties, are achieved [10]. Smart Contracts, ”a computerized transaction protocol that executes the terms of a contract,” aim to reduce the necessity of trusted middlemen between transacting parties and the occurrence of malicious or unintentional exceptions [10]. The current trend of decentralization is motivating the search for cutting-edge solutions that enable reliable and tamper-proof data and energy exchange to boost self-consumption in local energy communities and assist the implementation of more distributed control solutions [17]. These challenges might be handled by smart contracts. When deployed to a blockchain, this technology enables trusted transactions between anonymous parties without the need for a central authority. Some of its applications scenarios are represented in figure 5.

The authors of [33] propose an autonomous decentralized framework for managing electrical consumption in smart building clusters. Smart contracts enabled the collaborative decision-making of participating nodes. This solution can autonomously allow the sharing of residue power in the smart building cluster and reduce the cost of distributed energy.

A decentralized framework to manage the daily energy exchanges within a Smart Building community is described in [26]. Through a series of local optimization processes, Smart Contracts enabled participants to collaboratively decide on a planning profile that minimizes the overall aggregated cost.

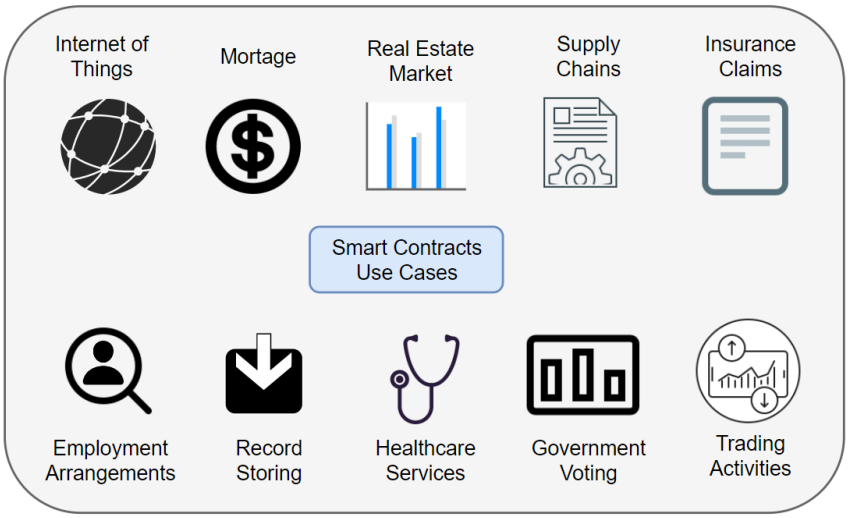


Figure 5. Smart Contracts Use Cases.

In [10] the topic of blockchains and smart contracts for the IoT is addressed. It is discussed some advantages and issues of this integration. The continued integration will result in significant transformations across several industries, create new business models and review the implementation of existing systems and processes.

In [2] the potential of integration between blockchain and digital building twins for performance-based smart contracts is demonstrated. It investigates how this integration could support a transition to a more performance-oriented built environment.

The system proposed in [34] aims to enhance and integrate hospital healthcare applications with blockchain technologies and smart contracts to provide immutable secure storage.

Programming Language

There are many different Smart Contract Languages adapted for automation in Smart Building applications. On table 2 some relevant Smart Contract languages where summarized.

Table 2. Comparison between Smart Contract Languages.

	Solidity[6]	**Vyper**[7]	**Yul**[8]	**Daml**[9]
Level of Programming	High	High	Low	High
Blockchain Portability	Yes	No	No	Yes
Database Integration	No	No	No	Yes
Community	Large	Small	Small	Large
Open-Source	Yes	Yes	Yes	Yes

Solidity is an object-oriented, high-level programming language used to implement smart contracts. The most noticeable use of Solidity has been in the development of Ethereum smart contracts.

[6] https://docs.soliditylang.org/en/v0.8.17/
[7] https://vyper.readthedocs.io/en/stable/
[8] https://docs.soliditylang.org/en/latest/yul.html
[9] https://daml.com/

[35]. Solidity, as a curly-bracket language, is heavily influenced by other languages such as JavaScript, C++, and Python [35]. One of the benefits for smart contract developers is the similarity of Solidity to modern programming languages [35]. Another one of its key advantages is that Solidity smart contracts can be easily transferred to other blockchain networks [35].

Vyper is a Python-influenced programming language designed specifically for smart contract development. Vyper's three fundamental design principles and goals lay the groundwork for its efficiency in smart contract development [35]. Auditability ensures that the code is human readable and that malicious code is difficult to write. Simplicity implies that the compiler implementation is straightforward and simple to comprehend and the last one Security.

Yul is a programming language used as an intermediate language in the compilation of Ethereum smart contracts written in Solidity [35]. Programs written in Yul can be read even if the code was produced by a Solidity compiler. High-level structures like loops, function calls, and if and switch expressions are available in Yul [36].

Another top contender among blockchain smart contract languages is Daml (Digital Asset Modelling Language). Daml is the leading platform for developing, deploying, and managing complex multi-party applications. It is an open-source programming language used to create distributed applications [35]. Daml is a private, secure, and scalable language for DLT, Blockchain, and Databases. It is supported by a growing number of platforms like Corda, VMWare, PostgreSQL, Hyperledger Fabric, Amazon Aurora and many others.

Daml 2.0, which now includes Canton Ledger, makes it possible for multi-party applications to synchronize with conventional IT systems and different blockchains. This new version guarantees the privacy needed by organizations to safeguard sensitive data and comply to laws like the GDPR.

First, a quick summary of Canton is provided. A single virtual global ledger can be created by connecting many Daml Ledgers together using the Canton Daml Ledger interoperability protocol [37]. Participant nodes and domain nodes, each of which has a private contract store, make up a Canton network. Each participant node is associated with one or more synchronization domains, which enables it to communicate with all other parties whose participants are also associated with a certain domain [37]. A party may be a physical person, a legal entity, or just one of several accounts for a person or business. Using Daml smart contracts and the Canton protocol, parties hosted on several participant nodes can conduct transactions [37].

Canton complies faithfully with the authorization and privacy standards set by Daml for its transactions. Workflows can be composed on a virtual global ledger created by this infrastructure. These visibility and authorization guidelines are upheld by the Canton synchronization protocol, ensuring that data is transmitted securely and dependably even when malicious actors are present [37].

The Canton network is easily expandable, allowing for the inclusion of additional parties, ledgers, and applications that build upon existing ones. Extensions don't require a global network consensus or a centralized controlling entity [37].

Programmers use Daml to explain the contract formation, involved parties, and parties who authorized the contract formation [35]. Daml allows developers to focus on the business logic rather than how to convert ideas into code. This language expresses all parties, contracts, obligations, rights, and authorization directly [35].

All languages are open source, but only Daml supports database integration. Daml is the language selected for writing smart contracts in this work, due to its compatibility with open-source databases.

2.3. Combination of IoT and Smart Contracts

IoT has been gaining a lot of popularity due to the wide range of its application domains. It is expected that billions of intelligent devices will be connected in emerging industries [8]. Automation and security for smart devices will be crucial for IoT networks in the future, which will require complex solutions. As a result of their inherent automated and decentralized character, smart contracts will address many of these challenges in future IoT systems [8]. More specifically, IoT concepts are being applied more frequently in the energy sector for monitoring and controlling remote assets and smart cities [17]. There is growing concern about the security and control of the data collected by IoT devices, particularly when these are centrally controlled by a single system [17]. Many of the security concerns of the IoT context can be satisfied via smart contracts [8]. Some of the Smart Contract applications in IoT context are presented in figure 6 and summarized below.

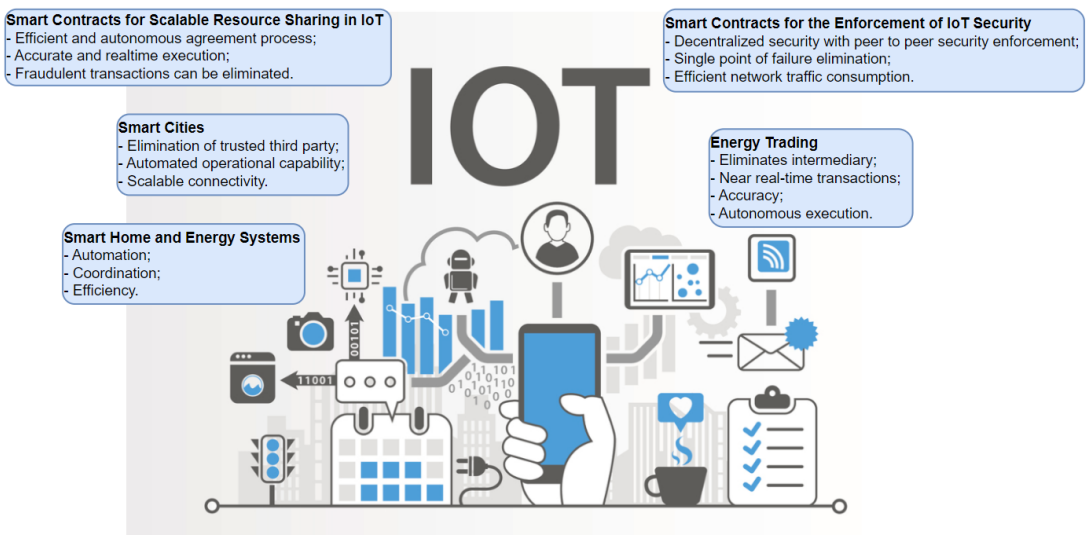


Figure 6. Smart Contract applications in Internet of Things.

Smart contracts for scalable resource sharing of IoT

In the IoT network, resource sharing is essential. The resource-constrained infrastructure requires optimal resource-sharing service. This service cannot be a resource intensive process because it will add another overhead to the system, increasing network traffic and computing costs [8]. Peer-to-peer resource sharing, which is ideal in terms of computation and network traffic, will be made possible by

smart contracts. As a result, smart contracts have the potential to be the next-generation resource sharing approach in the IoT context.

In [38] the SmartEdge, an Ethereum-based smart contract for edge computing, is introduced as a low-cost and low-overhead tool for compute-resource management. The smart contract has five different states that can be transitioned through during its lifetime. The solution enables nodes to offload computation in a verifiable manner to edge computing devices owned by third parties in exchange for payment.

Smart contracts for the enforcement of IoT security

In the context of IoT, security is a vital requirement. It is very difficult to enforce security by increasing key sizes with many cryptographic operations on devices with limited resources [8]. The devices are limited in terms of memory and processing capacity. As a result, IoT devices will raise costs when using public-key certificates [8]. It would increase the network traffic due to verification requests from PKI systems to the cloud servers [8]. The centralized servers access control and privilege definition will be open to attack [8]. In general, the blockchain enables the stakeholders to decentrally install smart contracts with access control policies embedded in them. In contrast to the cloud computing environment, the code is immutable and permanent. When security is implemented, no extensive network traffic is generated due to the decentralized operation.

In [39] it is illustrated how IoT devices and data will become a trading commodity in the near future, as well as the infeasibility of a centralized trading platform. According to the authors, blockchain-based smart contracts will eliminate the need for a trusted third party. It is also demonstrated the use of smart contracts and blockchain to establish trust and enable end-to-end trading.

Smart Cities

Smart cities are a significant IoT innovation that will be applied in the future construction of cities and countries infrastructures and also improve the quality of life [8]. The scalability of the operating platforms is a major concern because smart cities contain thousands of connected devices. Additionally, centralization will come with additional risks and high overhead costs. Future smart cities will heavily rely on blockchain technology and smart contracts [8]. With guaranteed service availability, the decentralized model's operational capability of smart contracts will be more valuable. Smart contracts' peer-to-peer functionality will cut down on the need for network resources consumption, increase network efficiency, and decrease latency [8].

In [40] a security framework is presented to provide a secure communication platform in a smart city. This solution integrates the blockchain technology with smart devices.

A contract-based energy blockchain is suggested in [41] for the smart community's electric vehicle charging. The authors used smart contracts to implement secure charging services once the necessary

trading conditions were met, as well as for cryptocurrency exchanges.

Smart contracts in energy trading

The blockchain-based smart contracts have a lot to offer to the energy sector. The important characteristics of smart contracts, such as accuracy, autonomous execution, and peer-to-peer operations, enable the energy market to support peer-to-peer energy trading, smart metering, effective renewable energy production, etc [8]. The applications of smart contracts in the context of smart energy, highlight the adaptability of blockchain-based smart contracts for the energy business [8].

The conceptual design, as well as the energy grid prototype and control layer running on the Ethereum platform, are presented in [42]. A middleware application that connects the grid and the smart contract was suggested by the authors as a way to simplify communication between two parties.

The framework proposed in [43] is based on blockchain technology and includes pricing methods, the architecture of the power transaction system, and a few modules in the energy trading system. When there is a lack of trust between trading entities, smart contracts are incorporated to enable the system for decentralized trading.

Smart Home and Energy systems

Home energy management systems (HEMS) are currently using smart contracts to coordinate variable loads and assets, such as scheduling heating and cooling in homes [17]. In order to reduce costs and minimise the user's carbon footprint, household appliances are coordinated with the help of smart contracts because of their security [17]. Smart contracts are used in Smart Home applications to coordinate assets, to automatically perform control decisions (turn appliances on or off) based on the state of particular variables, and to ensure the communication channel is secure [17].

In [44] is proposed a system that uses three different types of smart contracts to enable access control, assess asset misbehavior, and register new access control policies in a Smart Home.

A decentralized system for controlling electrical usage in a group of Smart Buildings is presented in [26]. Participants can choose a planning profile that reduces the total aggregated cost through the use of smart contracts.

This dissertation fits in this last application and it will be possible to design an architecture that allows this combination, while also addressing the trust and energy problems identified in section 2.1. In this manner, an architecture will be suggested and discussed in the following chapter.

CHAPTER 3

Proposed Architecture

The main goal of the current work is to solve the problem of trust in the delegation of building management functions. A system for distributed management in smart buildings has been developed as a solution to this problem. A distributed system may lead to more efficient management of smart buildings, including other benefits such as energy management. This section intends to explain the proposed design of that system. This chapter is structured as follows, section 3.1 presents in detail the architecture of the system proposed, the section 3.2 explains the smart contract life-cycle and the storage of the system, and the section 3.3 illustrates the architecture flow control.

3.1. Architecture Building Block

The problem that this work intends to solve encompasses on two main entities, that for clarity of explanation we refer as Bob, the entity that manages the smart building, and Alice, the external entity that will control part or parts of Bob's smart building. There must be a communication between these two entities so that one can give control to the other. A smart building has several control and management functions that can be assigned to an external entity. Thus, there may exist several external entities that can control certain functions of an smart building. Therefore, there must be a communication between the Bob and any off the existing external entities. Figure 7 represents the bidirectional communication between stakeholders.

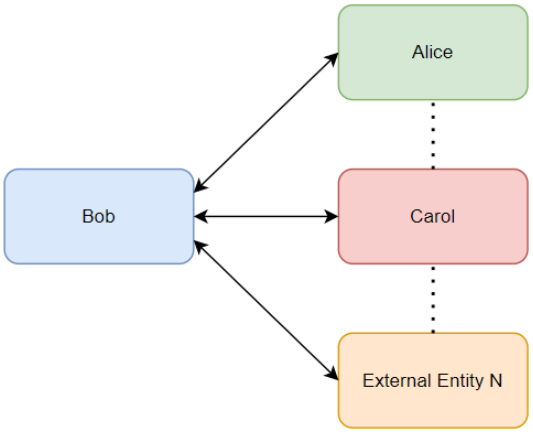


Figure 7. Communication between stakeholders.

The proposed system architecture is based on two nodes, one for each entity. Therefore, one node represents Bob and the other one represents Alice. These two nodes will communicate with each other. In order for Bob to provide control of certain functions of its smart building to Alice, the external

entity, it will have to create a smart contract and then share it with Alice. This technology enables trusted negotiations between unknown entities. Each entity has a private identifier that enables identification. The contract shall contain the entities interested in the exchange of services. The smart contract created by Bob must include Alice's id as well as his own. To control the functions that Bob has provided, Alice will have to execute the contract. For instance, a functionality could be Bob granting Alice the ability to control a light, depending on its present state. As a result, when Bob provides control over an off light, Alice can only turn it on. This control is not of course limited to control only one light. Other options for potential controls will be discussed later. The domain node will enable the communication between the two nodes. The domain node represents the architecture component controls and ensures coordination of the distributed system. The domain node is responsible for tasks such as connecting all smart contract storage and allow them to be reconciled. Either a database or a blockchain might be used to achieve its goals. Some of the characteristics/requirements of the domain node are referred bellow:

- Synchronization: The domain must facilitate the synchronization of the shared ledger among participants.
- Transparency: The domain must inform the designated participants timely on changes to the shared ledger.
- Finality: The domain must facilitate the synchronization of the shared ledger in an append-only fashion.

Figure 8 illustrates the proposed architecture based on all the above described nodes. The next paragraphs provide a description of the components and requirements that constitute this architecture.

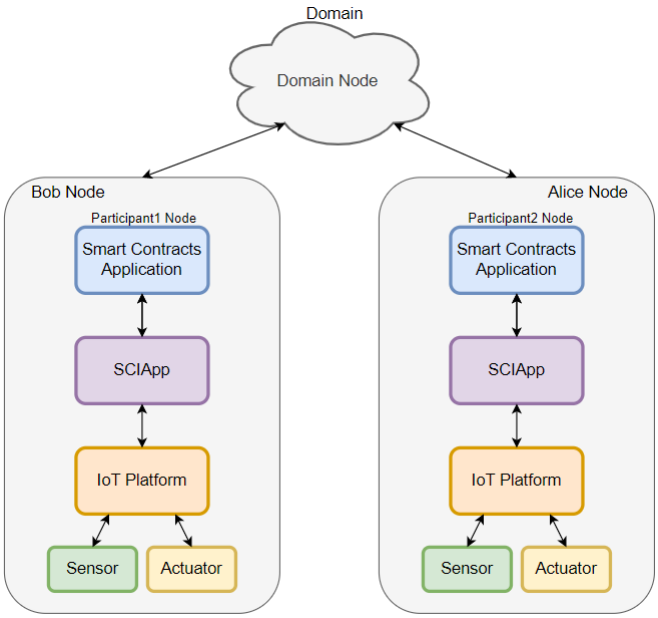


Figure 8. Architecture proposal.

Both Bob and Alice will have a control interface contained in an IoT platform. This interface will serve to visualize and control the smart building and to execute external entity operations. This IoT platform shows the items that are available to be controlled in the smart building and in the external entity. Here it is possible to check the state of an item and change it, as well as other functionalities regarding other items. Some system requirements are listed below:

- The item that can be controlled by Alice will only be visible if Bob has first created a specific smart contract. This contract specifies the item to be controlled and which external entity is intended for. The IoT platform interface of Alice will not display any potential controls if no smart contract has been created.
- An external entity can only control the item in the smart building once. If further control is required, a smart contract must be created again. If the external entity agrees to control the smart building, the necessary controls will be visible on the IoT platform of the external entity. When it performs a control, such as changing the state of a light, the new state should be updated first on the IoT platform of the external entity, then on the smart building entity's IoT platform, and finally on the light itself, which is in the smart building. Some system requirements are listed below:

In order to achieve this solution it is necessary to consider a Smart Contract Integration Application (SCIApp) in each node. The several applications that make up this solution will be able to connect with one another thanks to this SCIApp. It will act as a link between the Smart Contracts application and the IoT platform.

The two operational flows in this architecture are upstream and downstream. In the Upstream connection, starting in Bob's node, sensors and actuators send data to the IoT platform. The platform forwards data to the SCIApp for processing, which creates the Smart Contracts through the Smart Contract Application. Via domain node, Alice has access to the smart contract created. The data is transmitted to SCIApp by the Smart Contract Application at Alice's node. Based on the data obtained from SCIApp, OpenHAB then updates the sensors and actuators.

In Downstream, the flow is the opposite of upstream. In the Downstream connection, starting on Alice's node, it is performed a control in the IoT platform. This information is then sent to SCIApp for processing. Next, SCIApp sends the information received from IoT platform to the Smart Contract Application. The contract is exercised, and this information is transmitted to Bob's node via domain node. The information that the contract was exercised is sent to the SCIApp for processing by the Smart Contract Application. The SCIApp then transmits the contract data to the IoT platform and it updates the sensors and actuators.

Bob grants the control of a specific function to Alice via this IoT platform. The IoT platform receives the states and changes of the items. The controls that are performed here will be sent to the SCIApp for processing.

The SCIApp provides connectivity between the IoT platform and the Smart Contract Application. It sends the information received from IoT platform to the Smart Contract Application. The SCIApp also processes the contract data and sends it to the IoT platform.

Along with creating the contracts, the smart contracts application will also store them. It sends the contract information to the external entity and transmits the contract information to the SCIApp.

3.2. Smart Contract Integration

In this system, the smart contracts can be created by Bob and Alice's nodes. The contract creation/exercise process is automated, i.e. when a user, such as Bob or Alice, interacts with the IoT platform, a contract is created. Figure 9 shows the contract formation cycle in detail and is described in the following paragraphs.

First, when Bob gives Alice permission, to control an item in the building, via the IoT platform, a contract will be created. The permission will be granted by clicking a button on the IoT platform of Bob's node (1).

Once permission to control an item is granted to Alice, its IoT platform will update its own UI with a new control functionality. After Alice controls the item, it will exercise the contract, previously created by Bob (2). The contract exercise will be triggered by a click of a button on Alice's IoT platform.

Finally, after the contract has been exercised, it will be archived, and a new contract will be created automatically by Alice's node, and shared with Bob (3).

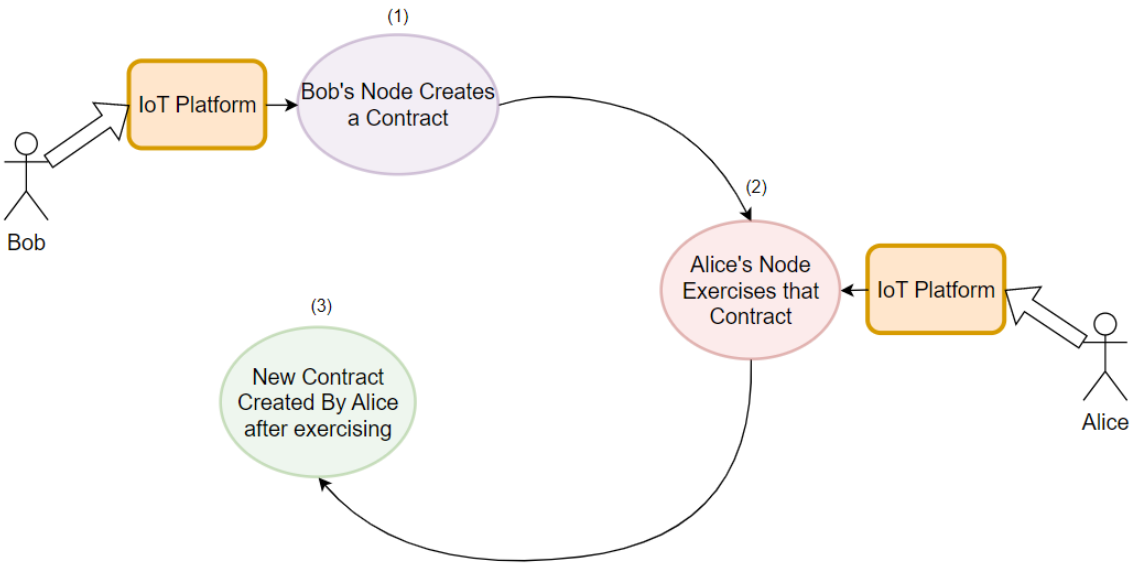


Figure 9. Smart Contract life-cycle.

The contract serves to transfer control of an smart building item to an external entity. The contracts created can be deployed on both blockchain and database platforms. This system could support a wide variety of contracts.

Some of the following contracts could be created, depending on the goals of the entity in charge of managing the smart building:

- Checking the state of a certain light.
- Checking the state of a certain temperature sensor.
- Interactive control of a light. (ON/OFF, Color, Intensity, etc)
- Interactive control of an air conditioner. (ON/OFF, Temperature, etc)
- Interactive control of a power outlet.

In the Implementation Chapter, one of these contracts will be selected to be incorporated in the system.

The smart contracts used in this solution must be stored. Each entity involved in a control negotiation contains its own storage. If the control of a smart building is transferred to a external entity, the corresponding smart contract is retained in each entity's storage. In order to manage contracts among the different participants, a number of methods are possible, including blockchain and databases. When a control negotiation occurs between two entities, the storage of these entities becomes reconciled after communication between their nodes. The dotted line connecting the two entities' storage represents exactly that. Figure 10 depicts the storage architecture and how nodes communicate with one another.

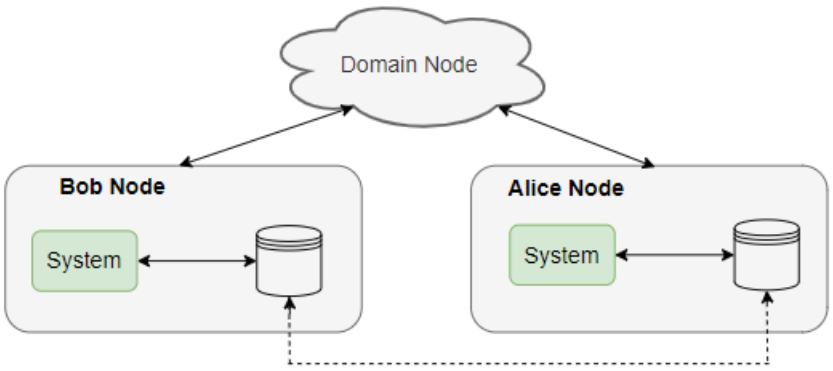


Figure 10. Database architecture.

3.3. Architecture Flow Control

The interaction between the two nodes of the suggested solution is shown in figure 11. This communication shows how to delegate the control of an smart building light, to an external entity. It is also demonstrated that just after this control assignment, the external entity changed the state of the light. This action leads to an update of the light status in the smart building. As previously mentioned, the nodes communicate via a domain node.

The upstream flow represents the transfer of control of an item to Alice, as well as all of the processes involved. Bob is the one who triggers this flow. In the case of the downstream flow, it is Alice who triggers it by changing the state of the item granted by Bob. In this flow all the steps are

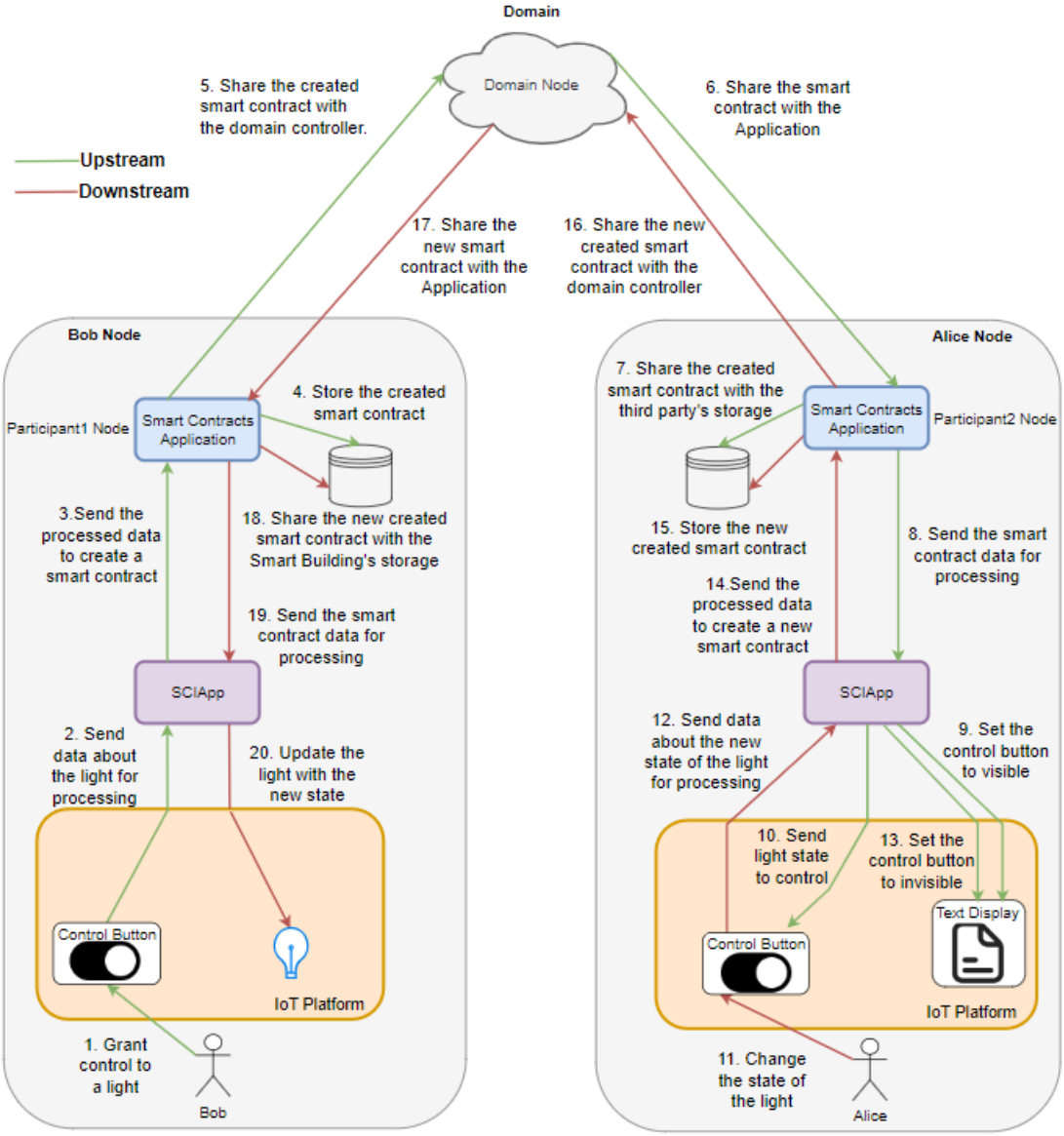


Figure 11. Interaction between the two nodes.

specified until the item is updated in the smart building. The item chosen for this interaction was a light.

In the upstream flow, it is possible to see Bob's node granting control of a light to Alice (1). This control is carried out in the IoT Platform, which sends this data to the SCIApp (2). The SCIApp then processes this data and sends it to the Smart Contract Application in order to create a smart contract (3). Finally, this contract is stored on Bob's node and transmitted to Alice's node via the domain node (4)(5)(6).

As soon as Alice's node notices that a new contract has been created, the Smart Contract Application notifies its storage and also communicates with the SCIApp to process the data of the smart contract (7)(8). After receiving the data from the SCIApp, the IoT Platform will show a switch in its interface along with the current state of the light (9)(10).

After illustrating the upstream flow, downstream flow will now be discussed. The control button is now visible. Alice will change the state of the light by pressing the control button (11). The SCIApp will communicate to the SCIApp the new state of the light and that the control was executed (12). The IoT Platform will make the control button invisible (13). When the SCIApp has finished processing the data, the Smart Contract Application will be notified (14). It will create a new contract, store it, and communicate with Bob's node via domain node (15)(16)(17).

Finally, the last step of the communication occurs in Bob's node. The Smart Contract Application contacts the storage and the SCIApp as soon as it learns about a new contract (18)(19). The SCIApp will receive this data and process it. After that, using the information obtained from the SCIApp, the IoT platform will update the light state (20).

CHAPTER 4

Implementation

This chapter covers all of the technological options that were considered, as well as the steps for implementing the solution. Whenever possible, open-source solutions were used in the system's implementation. Section 4.1 discusses the choices for the Virtual Machines (VM), Operating System and their Synchronization as well as the specification assigned to the host and the virtual machines. The action cycle of the implemented contract is covered in section 4.2. Section 4.3 introduces the smart contract language and storage that were chosen, as well as how this application was developed and how it connects to a storage. The SCIApp's methods for integrating the IoT platform and the Smart Contract Application are described in section 4.4. The IoT platform and the procedures that must be established on it for the implementation of the solution are presented in section 4.5. Section 4.6 contains a description of the communication protocols that are used to transmit information in the system. Section 4.7 discusses the steps to be taken in order to implement other controls.

Figure 12 depicts the major technological options chosen to implement the solution as well as the protocols that can be used between the connections. It serves as an introduction to the themes that will be discussed in greater depth in the following sections. This figure also indicates the sections in which each system component is described.

4.1. Testbed Setup

Choosing the virtual machine software to use was the first step in putting the solution into practice. It was essential to select an open-source option in order to keep the cost of the solution low. VMware Workstation Player was chosen after considering several options and taking into account that it had previously been used during the academic life. It's simplicity makes it an ideal tool for development and testing. The virtual machines have been configured in NAT (Network Address Translation) mode. Regarding the system hardware, table 3 indicates the specifications of the host and the virtual machines.

Table 3. Specifications of Host and Virtual Machines.

Node Type	Virtualization Platform	CPU	Cores	RAM	Storage	OS
Host	-	Intel(R) Core(TM) i7-8565U CPU 2.00GHz	4	16GB	512GB	Windows 11
Bob Node	VMware	Intel(R) Core(TM) i7-8565U CPU 2.00GHz	4	5GB	80.1GB	Kali Linux
Domain Node	VMware	Intel(R) Core(TM) i7-8565U CPU 2.00GHz	4	2GB	80.1GB	Kali Linux
Alice Node	VMware	Intel(R) Core(TM) i7-8565U CPU 2.00GHz	4	5GB	80.1GB	Kali Linux

Regarding the operating system, Linux was chosen in order to maintain the same open-source approach. It is known for being a more stable and secure system and consumes very little resources

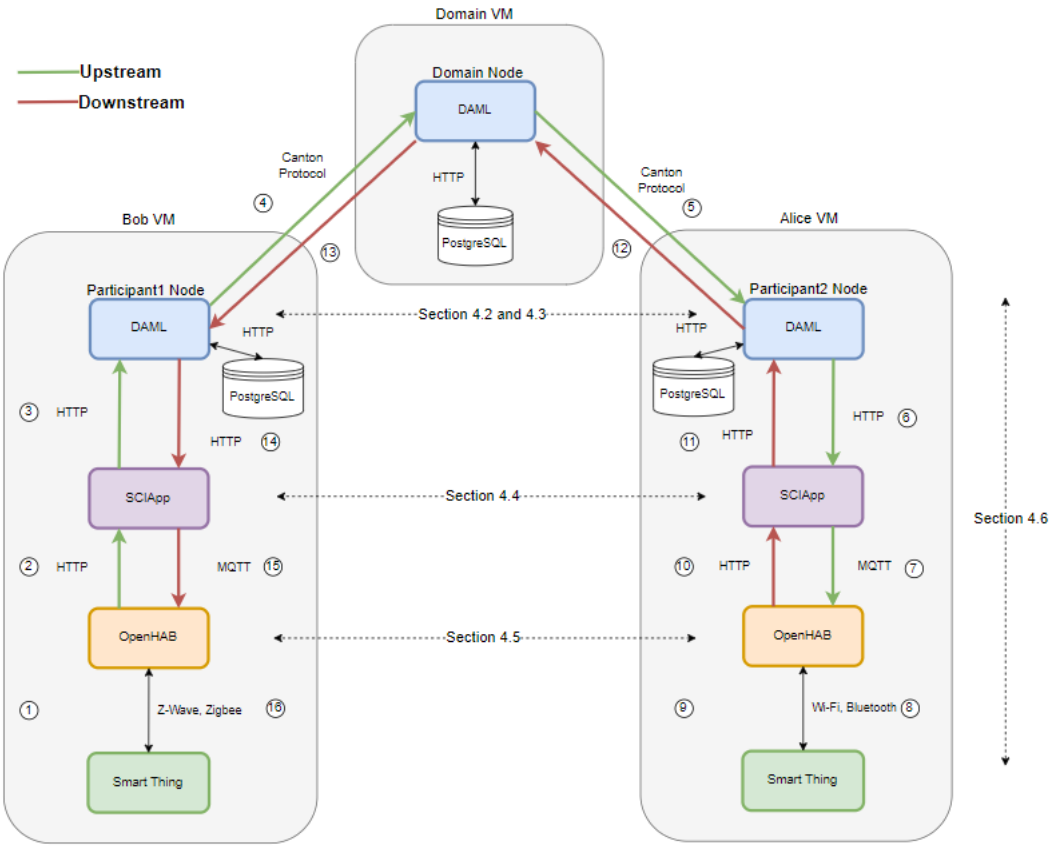


Figure 12. Technological Options of the Implementation.

and space unlike Windows. Although there are many Linux distributions, Kali Linux[10], a distribution focused on cyber-security and penetration testing attacks, was the one selected to carry out this solution. Not only is Kali Linux better suited for development, but it is also open-source, secure, reliable, customizable and has continued support from developers and the community.

The virtual machines must be synchronized with the same clock in order to later run performance tests in the system. As reaction and response times will be measured, if the clocks of the virtual machines are not synchronized, the times will not be coherent. The Network Time Protocol (NTP), a networking protocol for clock synchronization, was set up on the three VMs in order to accomplish this. The Domain VM was configured as an NTP server, while the other two VMs were configured as NTP clients. Appendix A contains the instructions on how to configure the NTP in each VM. It was possible to check whether the NTP Clients are synchronized with the NTP Server by conducting fifty tests that involved getting the time of each virtual machine. These results allow to conclude that the difference between the acquired times is very small, i.e. the results of the performance and functional tests will be reliable. Table 4 displays the medium delays between the clients and the server.

[10]https://www.kali.org/

Table 4. Delay time between VMs.

	Bob VM	Alice VM
Domain VM	7.37 ms	7.48 ms
Standard deviation	1.92 ms	1.23 ms

4.2. Smart Contracts

The contract selected for this system implementation consists on the iterative control of a light and it is described in figure 13. This contract was one of the contracts mentioned in the Architecture section. If a different type of smart contract had been chosen for this implementation, it would have to be reprogrammed to achieve the characteristics of this new solution. The contract selected will be composed of two parties: the entity that created the contract, i.e. Bob, and the entity to whom Bob wishes to delegate control, i.e. Alice. This agreement will also specify the reference in the building of the light which is intended to grant the control, as well as its current state at the time the control is given. The contract's structure is depicted in Figure 13.

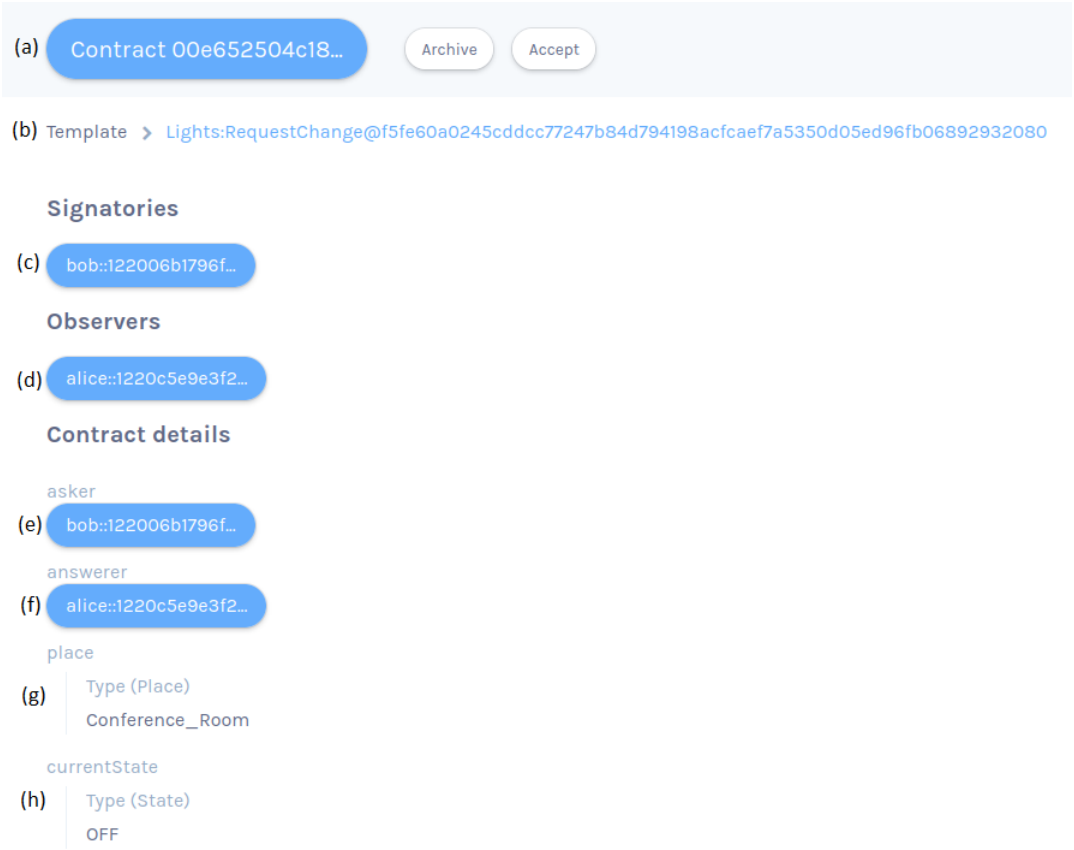


Figure 13. Smart Contract Architecture.

Each new contract is given a unique contract id that is assigned at random (a). Contracts are created from blueprint called template. In other words it corresponds to the id of the written code (b). The signatories of a contract are specified by the signatory keyword. These are the parties who must have authority to create or archive the contract, just like in a real contract (c). At least one

signatory is required for every contract. Signatories of a contract are guaranteed to see the creation and archival of that contract. An observer is a contract party. Being an observer allows them to see that instance and all the information about it (d). They do not have to consent to the creation. The contract is also visible to the observers because they are additional stakeholders. In this system, Bob is the signatory because he created the contract, and Alice is the observer because Bob determined so.

In the contract details section, the asker field designates the entity that decides to grant control (e), and the answerer field indicates the entity to which the asker grants control (f). In this system, the asker is defined as Bob, while the answerer is Alice. The reference and current state of the light, respectively, are represented by the fields place and currentState, respectively (g)(h). In this implementation the place field corresponds to the reference of the item light, but could be altered for the name of the item instead.

When the observer Alice receives this contract, it may then be executed by her. Given the example that the light was OFF when Bob grants control, Alice can only control the light by turning it to ON. Alice by changing the state of the light, will exercise a choice in the previous contract. This leads to the creation of a new contract, turning Alice as Signatory of this new contract. As a result the previously created contract will be archived. The new contract's structure is depicted in Figure 14.

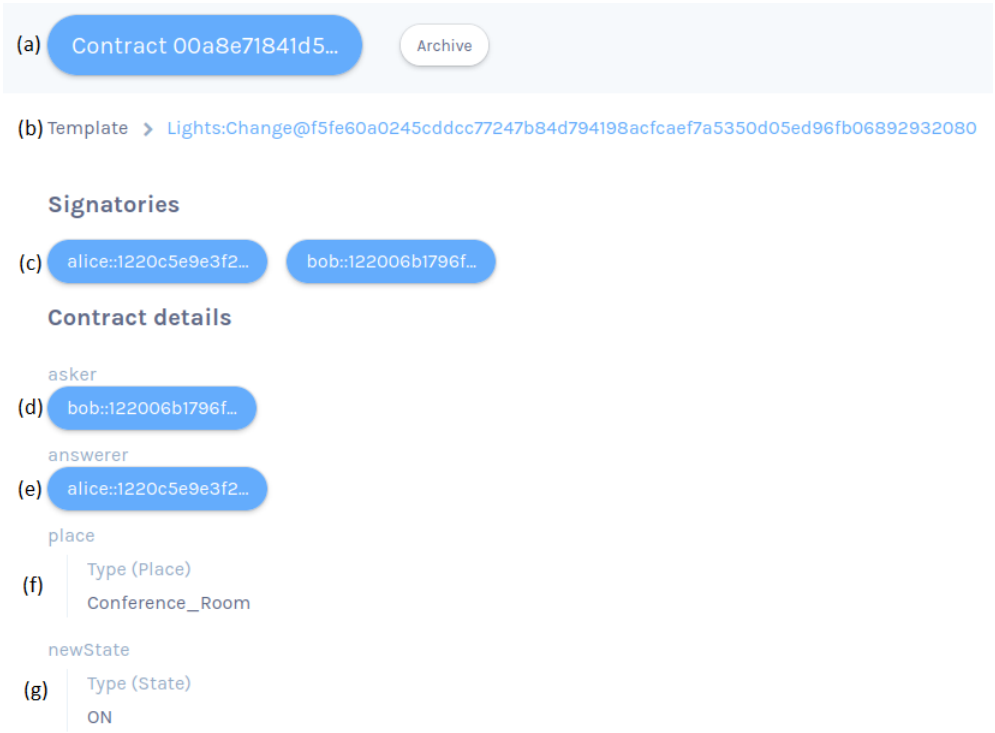


Figure 14. Smart Contract Architecture.

A new id is generated for this new contract, distinct from any others that have already been created (a). The template remains unchanged (b). However, Alice joins Bob as a signatory because she has exercised the previous contract (c). There is no change in the asker, answerer, or place

(e)(f)(g). Alice's updated state is now defined, replacing the current state (g). This allows Bob to process the data and modify the building's light state. The newState field in this new contract is the only modification to the previous contract details section (g). This field, which represents the new state of the light defined by Alice, can never match currentState.

A scenario also possible to occur, after Bob delegates control of a light to Alice, would be for Bob, to change the state of the light before Alice does so. There may be several options for dealing with this situation in this particular case, such as:

- Bob's control of the lamp is blocked;
- The contract is revoked when Bob modifies the light's state;
- Do nothing, and Alice turns on what is already on.

The third option was selected for this solution implementation. However, this problem would not have occurred if a contract to verify only the condition of a light had been implemented.

4.3. DAML

Daml[11] was the Smart Contract language chosen for this solution's implementation. Visual Studio Code and JDK 11 or above must be previously installed in order to install the Daml SDK. In Appendix B is described the installation of Daml SDK as well as VS Code once JAVA is already installed. Following the installation, the smart contract code had to be written. Because Daml is a relatively new language that needed to be studied, this phase took a long time. The code developed for the smart contract can be accessed in the Git repository[12]. To connect the entities of this solution, a Canton network was required. Daml employs this nomenclature to identify participants who communicate via the Canton protocol. The Canton network's setup will be described in the following paragraphs.

This solution will be implemented with three nodes: the Bob node, the Domain node, and the Alice node. In Daml's nomenclature, Bob will be participant1, Alice participant2, and the domain will be assigned equally. Bob and Alice are represented by the participant nodes, while the domain is the node that allows participants to communicate with one another. Figure 15 displays each node's setup.

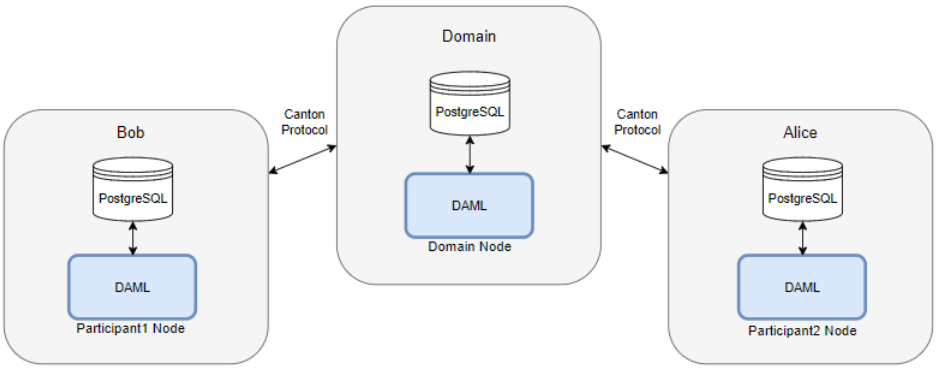


Figure 15. Canton Architecture.

[11] https://daml.com/
[12] https://github.com/bclse/dissertation/

In terms of storage for the canton network, the only open source database that has integration with daml is postgresql. Daml supports several blockchains, including Hyperledger Fabric and Ethereum, but this integration is not open source. Therefore, postgresql was chosen as the synchronization technology that will implement the ledger. The postgresql installation is detailed in Appendix C.
The steps to implement this solution are as follows:

- A - Create the configuration file; Used for each node, and the file for the Domain connection for Bob and Alice.
- B - Generate the RSA key; Used for Authorization of Bob and Alice.
- C - Create a signed JWT Token; Used for Bob and Alice.
- D - Launch the canton network; Execute the commands on each node.
- E - Create a dar file; Used for Bob and Alice.
- F - Create the parties; Used for Bob and Alice.
- G - Lauch JSON API; Used for Bob and Alice.

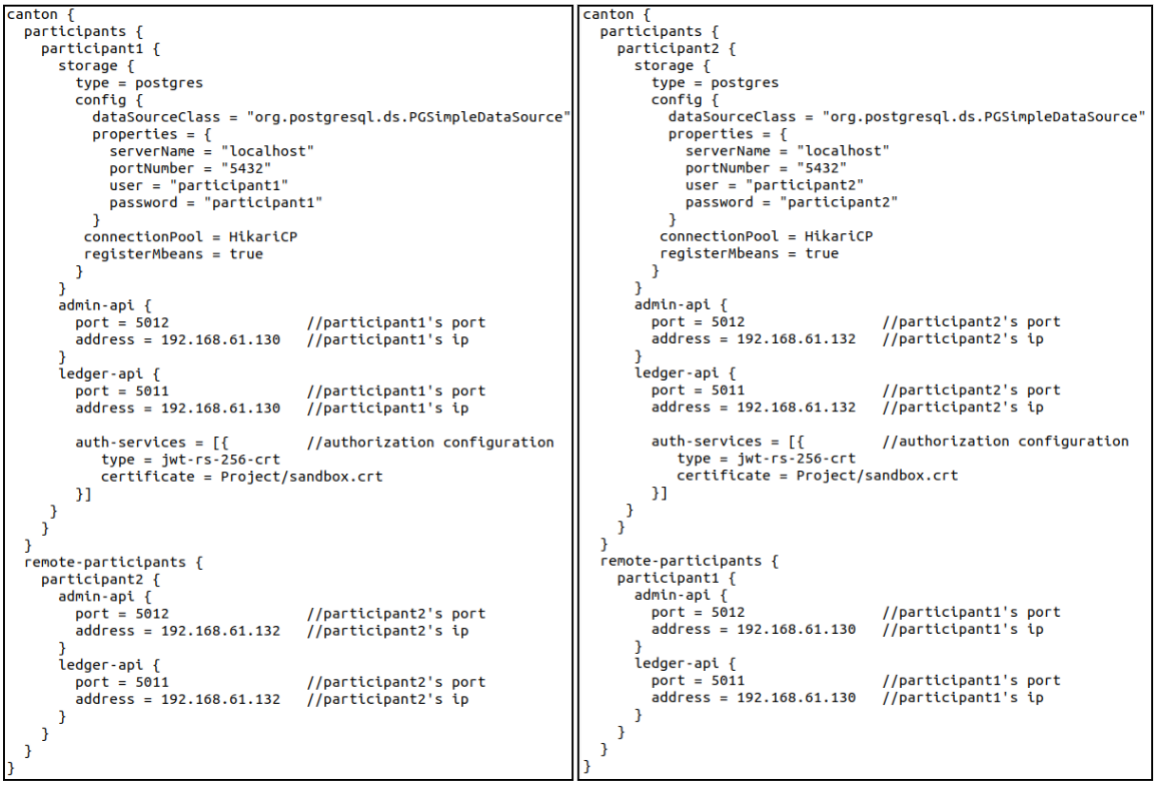
A - Creation of the configuration files.

To configure each node, the release package v2.1.1[13] was downloaded and extracted and it contains the scripts for running Canton. The configuration files for the three nodes were then created. These files were inserted in a folder called Project placed inside the downloaded package. The configuration file for Bob and Alice is illustrated in Figure 16. In the Project folder the source code for the smart contract, was also placed, which comes in a .dar file.

The configuration file specifies the storage access, the participant APIs, the configuration of other Canton network participants, and the authorization services, which will be discussed later. The connection to PosgreSQL is established using a previously created user and password. The created contracts will be stored in this local database. Each node's configuration file contains a connection with a unique user and password for that node. Each participant node exposes an Admin API and an Ledger API. The IP address and port number assigned to these APIs are specified in the configuration file. The Admin API allows the administrator to manage the participant node's connections to domains, add or remove parties to be hosted at the participant node, upload new Daml archives, configure the operational data of the participant, such as cryptographic keys and run diagnostic commands. The Ledger API enables each participant's parties to access the Ledger. It is also possible to use the HTTP JSON API Service to access the Ledger API. The developed SCIApp communicates with the ledger via the HTTP JSON API. This service, at its core, provides a simplified view of the active contract set as well as additional primitives for querying and exposing the contracts via a well-defined JSON-based encoding over a standard HTTP connection.

The domain configuration file is shown in Figure 17. This file defines the database connection and the configuration of the remote participants, previously described, as well as the tokens for

[13] https://github.com/digital-asset/canton/releases/tag/v2.1.1



```
canton {
  participants {
    participant1 {
      storage {
        type = postgres
        config {
          dataSourceClass = "org.postgresql.ds.PGSimpleDataSource"
          properties = {
            serverName = "localhost"
            portNumber = "5432"
            user = "participant1"
            password = "participant1"
          }
         connectionPool = HikariCP
         registerMbeans = true
        }
      }
      admin-api {
        port = 5012                //participant1's port
        address = 192.168.61.130   //participant1's ip
      }
      ledger-api {
        port = 5011                //participant1's port
        address = 192.168.61.130   //participant1's ip

        auth-services = [{         //authorization configuration
           type = jwt-rs-256-crt
           certificate = Project/sandbox.crt
        }]
     }
    }
  }
  remote-participants {
    participant2 {
      admin-api {
        port = 5012                //participant2's port
        address = 192.168.61.132   //participant2's ip
      }
      ledger-api {
        port = 5011                //participant2's port
        address = 192.168.61.132   //participant2's ip
      }
    }
  }
}
```

```
canton {
  participants {
    participant2 {
      storage {
        type = postgres
        config {
          dataSourceClass = "org.postgresql.ds.PGSimpleDataSource"
          properties = {
            serverName = "localhost"
            portNumber = "5432"
            user = "participant2"
            password = "participant2"
          }
         connectionPool = HikariCP
         registerMbeans = true
        }
      }
      admin-api {
        port = 5012                //participant2's port
        address = 192.168.61.132   //participant2's ip
      }
      ledger-api {
        port = 5011                //participant2's port
        address = 192.168.61.132   //participant2's ip

        auth-services = [{         //authorization configuration
           type = jwt-rs-256-crt
           certificate = Project/sandbox.crt
        }]
     }
    }
  }
  remote-participants {
    participant1 {
      admin-api {
        port = 5012                //participant1's port
        address = 192.168.61.130   //participant1's ip
      }
      ledger-api {
        port = 5011                //participant1's port
        address = 192.168.61.130   //participant1's ip
      }
    }
  }
}
```

Figure 16. Configuration file for Participants 1 and 2.

each participant and the Domain's Public and Admin API. The domain node provides a Public API that participant nodes can use to communicate with the synchronization domain. This must be accessible from where the participant nodes are hosted. A domain node, like a participant node, exposes an Admin API for administration services. It can be used to manage keys, set domain parameters and enable or disable participant nodes within a domain. The console provides access to the Admin APIs of the configured participants and domains.

Furthermore, participant and domain nodes communicate with one another via the Public API. The participants do not communicate with each other directly, but are free to connect to as many domains as they desire. There is nothing in the configuration file that says participant1 and participant2 should connect to domain. Canton connections are added dynamically rather than statically. To accomplish this, at the start of each participant node, a file is executed to connect the participants to the domain. Listing 1 displays the information in the participant 1 and participant 2 files.

```
participant1.domains.connect("mydomain","http://192.168.61.131:10018")
participant2.domains.connect("mydomain","http://192.168.61.131:10018")
```

Listing 1. Configuration files for connecting Participant 1 and 2 to Domain.

```
canton.remote-participants.participant1.token = "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOm51bGwsInN1YiI6InBhcnRpY2lwYW50X2FkbWluIiwiZXh
wIjoxOTYzMDI3ODAzLCJzY29wZSI6ImRhbWxfbGVkZ2VyX2FwaSJ9.0DTTAfGRLnzoLNkFM0wsEn8AZqtmXSx3Mqxj3FVD2v9qC5lVYb94DMV3L3me3TAFmrf2xtF3jRmugvMtbdlqwL7
CM1R6QQQ0x6Ubfm_fEJGHJqbYyzWcBnAqFpEumgpI7P8ocRH5jIu6W3RuLqb97rkRKhzRL4JgXh1D-jrtmFm0-DFyOtfUaCxY4Bpca73-0eLIpAW-XRtQ1IeIy0mwrJ_I0anI7bY2FJEY
TE__AFaN7x1gKGp3GAPTWeXT-YH2rQ9Hz5JUdx-YNUWvVI_jtb3Szlbtz7zCSeY-B1Qe1F47q-yA_sIB5tGaJ6DtKfjZfXlVH1DbhSrJTm9k-5EyQ"

canton.remote-participants.participant2.token = "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOm51bGwsInN1YiI6InBhcnRpY2lwYW50X2FkbWluIiwiZXh
wIjoxOTYzMDI3ODAzLCJzY29wZSI6ImRhbWxfbGVkZ2VyX2FwaSJ9.B0sw_hMy4ISIqgfbQGdsmek-HRR-UOmN5V2XALJso6wCXNZ-R8SV2247tzrxXWRQ4NM9b7I9tcaJCOf-_izTlp9
IheC8ScJiR-upLSSPWBGxMtVla76i3Jg0e4k-jjmhXw_goRV0DfoSFJfP4ESOaee66M5pm1DGHDTbuOOUQ0obyyMu29gxQQQIFtf0x7G7IGKOA8GEIAWW-yxbpNxtlBisfxElMK-koM7S
UP5cevdbRZAJmEfcyN0J2dXzeW5ulHamI9u-pyRJqa92B5dXo2XTqsdodD_ZeyjXPSW-FHf1uniEE6B8rQ0-HIih3rVv5J6xaSpwijithTzcjuh_4w"

canton {
  domains {
    mydomain {
      storage {
        type = postgres
        config {
          dataSourceClass = "org.postgresql.ds.PGSimpleDataSource"
          properties = {
            serverName = "localhost"
            portNumber = "5432"
            user = "domain"
            password = "domain"
          }
         connectionPool = HikariCP
         registerMbeans = true
        }
      }
      admin-api {
        port = 10017                        //domains's port
        address = 192.168.61.131            //domains's ip
      }
      public-api {
        port = 10018                        //domains's port
        address = 192.168.61.131            //domains's ip
      }
    }
  }

  remote-participants {
    participant1 {
      admin-api {
        port = 5012                         //participant1's port
        address = 192.168.61.130            //participant1's ip
      }
      ledger-api {
        port = 5011                         //participant1's port
        address = 192.168.61.130            //participant1's ip
      }
    }
  }

  remote-participants {
    participant2 {
      admin-api {
        port = 5012                         //participant2's port
        address = 192.168.61.132            //participant2's ip
      }
      ledger-api {
        port = 5011                         //participant2's port
        address = 192.168.61.132            //participant2's ip
      }
    }
  }
}
```

Figure 17. Configuration file for Domain.

B - Generate the RSA keys.

Regarding the authorization services, these are configured in each participant node. In order to generate the RSA keys, each participant must run the following command in listing 2.

```
openssl req -nodes -new -x509 -keyout sandbox.key -out sandbox.crt
```

Listing 2. Generate the RSA keys.

Two files are created as a result of this command. The first file, sandbox.key, contains the private key in PEM/DER/PKCS#1 format. The second file is sandbox.crt, which is a self-signed certificate in PEM/DER/X.509 Certificate format that contains the public key. The sandbox.crt file is defined in each participant's node configuration, as seen in figure 16, and the private key contained within the

sandbox.key file is used to sign the JWT token. This signed JWT Token is used to add authorization to Ledger API requests. Figure 18 displays a signed JWT Token.

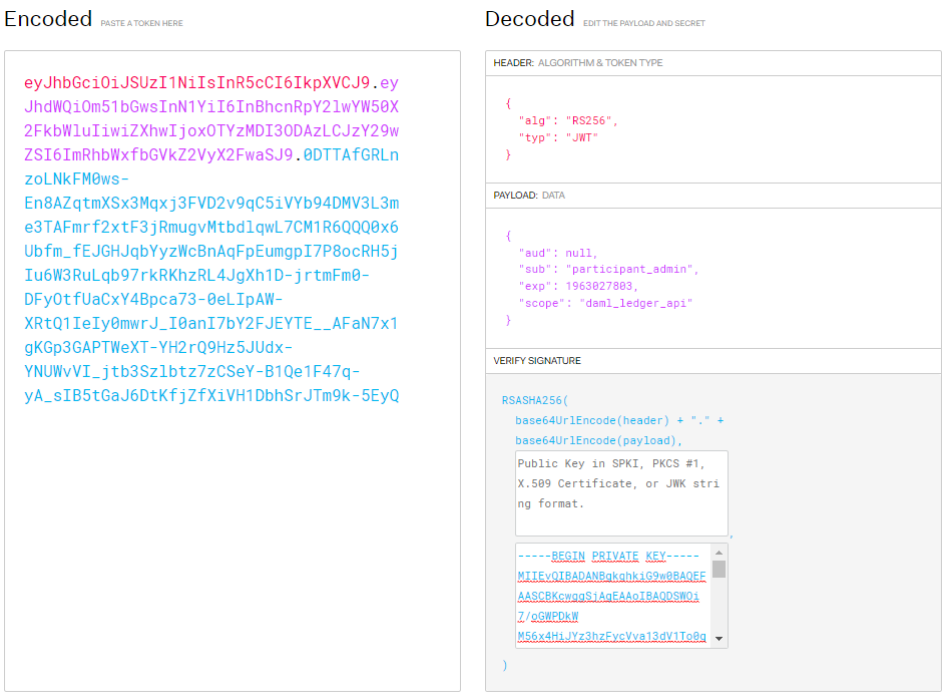


Figure 18. JWT Token.

C - Create a signed JWT Token.

The domain configuration file also defines the tokens used by each participant to access the Ledger API. JSON Web Token[14] (JWT) technology is used to generate these tokens. JWT is an open standard (RFC 7519) that defines a compact and self-contained method for securely transmitting information as a JSON object between parties. Because it is digitally signed, this information can be verified and trusted. JWTs can be signed using a secret (HMAC) or a public/private key pair (RSA or ECDSA).

D - Launch the canton network.

Finally, in order to launch the Canton network, it is necessary to perform the following commands. First the domain was launched with the command in listing 3, with the configuration file shown in figure 17.

```
bin/canton -c examples/Project/domain.conf
```

Listing 3. Launch Domain Node.

[14] https://jwt.io/

Next, the following command was executed in each participant node, using the files shown in figure 16 and listing 1, respectively. The command in listing 4 is meant to launch the Bob's node and connect to the domain. For the launch of Alice's node, it will be necessary to change the designation of participant1 to participant2.

```
bin/canton -c examples/Project/participant1.conf --bootstrap examples/Project/init.canton
```

Listing 4. Launch Bob Node.

E - Create a dar file.

When a Daml project is compiled, the compiler produces a Daml archive. These are platform-independent packages of compiled Daml code that can be uploaded to a Daml ledger or imported in other Daml projects. To generate the.dar file that will be used to create smart contracts, it was executed the following command. The command in listing 5 will generate the .dar file in the .daml/dist folder in the project root folder. For example, running daml build in the folder Project, with version 0.0.1 will result in the Daml archive .daml/dist/Project-0.0.1.dar.

```
daml build
```

Listing 5. Create .dar file.

F - Create the parties.

The next step was to create the Parties that each participant would use to access the Ledger. A script was written within the smart contract code to accomplish this. Each participant ran the command in listing 6 to create their own party, which will be saved in a text file (party1.json). Once more, to run the command on Alice's node would require changing the ip, the output file, and also the participant's name.

```
daml script --dar .daml/dist/Project-0.0.1.dar --ledger-host 192.168.61.130 --ledger-port
    5011 --output-file party1.json --script-name Lights:initializeUser --access-token-file
    token_file
```

Listing 6. Create the party.

The token file refereed in the listing 7, contains the encoded User Access Token with the RSA key created earlier. Figure 18 shows both encoded and decoded tokens, with the decoded token having the following format:

```
{   "aud": "someParticipantId",
    "sub": "someUserId",
    "exp": 1300819380
    "scope": "daml_ledger_api"   }
```

Listing 7. JWT token format.

G - Launch HTTP JSON API

To launch the HTTP JSON API Service, which will be the connection between Ledger API and the SCIApp, the command in listing 8 was executed in each of the participants. Again, it is necessary to change the ip to run this command in a different participant. This command specifies the JSON API port, which is the port to which the SCIApp connects to send HTTP messages. The domain does not require a HTTP JSON API because it only serves as a link between the two participants.

```
daml json-api --ledger-host 192.168.61.130 --ledger-port 5011 --http-port 7575
```

Listing 8. Launch JSON API.

The SCIApp will send contract creation requests as HTTP messages to the JSON API in order to interact with the Ledger via the Ledger API. If the Party from Participant 1 creates a contract that includes the Party from Participant 2, that contract will be available in Participant 2's Ledger. The canton network enables a virtual global ledger in which contracts with both parties are visible in both Participant Ledgers.

4.4. SCIApp

It was necessary to develop an SCIApp to connect the IoT platform to the Smart Contracts Application. The programming language of choice was JAVA, which was already installed, and IntelliJ IDEA[15] was chosen as the integrated development environment (IDE) for the development of this SCIApp. The IntelliJ installation procedure is described in Appendix D.

The upstream and downstream connections of Bob and Alice will be demonstrated. This demonstration is numbered in figure 12. Bob will create a contract that allows Alice to change the state of a light in his smart building. Once Alice changes the state of the light, the new state will be updated in Bob's smart building. The focus of this example is on SCIApp in both Bob and Alice's nodes.

Now, the connection between OpenHAB and the SCIApp of Bob's node will be described. When Bob clicks on the Control Button of the OpenHAB UI, he grants control of a light to Alice. This information is published in the Broker, via the upstream connection, using MQTT protocol. Once it is published, the SCIApp will receive this publication (2).

[15] https://www.jetbrains.com/idea/

In order to further clarify the upstream connection, after the control is granted to Alice, it is the responsibility of the SCIApp to provide this data to the Smart Contracts Application so that a contract can be created. The HTTP protocol is used to submit the request for the contract generation(3). With the help of the domain node the information that Bob has created a contract is transmitted to Alice (4)(5).

Moving on to the Alice's node upstream connection. The SCIApp of the Alice's node is in a state of passive waiting until the contract reaches the database, at which point the data will be processed and sent to the OpenHAB. After receiving the notification that the contract has been created by an HTTP message (6), the SCIApp of the Alice's node updates the OpenHAB UI (7), also using the HTTP protocol, allowing the control of the light.

The light can now be managed by Alice. The downstream connection starts when Alice modifies the state of the light in OpenHAB. An MQTT message will be published in the Broker and received by the SCIApp of the Alice's node (10). This message will inform that the light state was changed by Alice. The OpenHAB UI will be updated, denying a new control of the light. The SCIApp is responsible for notifying the Smart Contracts Application once it receives notification that the light has changed state (11). With the help of the domain node, the information that Alice has changed the state of the light, creating a new contract and exercising the previous one in the process, is transmitted to Bob (12)(13).

Moving on to the Bob's node downstream connection. The Bob's node will likewise be placed on passive hold until the Alice's contract is exercised. As soon as the contract exercised is received (14), the SCIApp of the Bob's node will update OpenHAB with the Alice's node modified state of the light (15), finishing the downstream connection.

Appendix E contains the libraries used to implement this system. All the code developed can be accessed in the Git repository[16]

4.5. OpenHAB

OpenHAB[17] was chosen as the IoT platform that will pass the data between the SCIApp and the items. The decision was simple because I was already familiar with the platform and felt comfortable implementing various features. This IoT platform is one of several home automation platforms available in the market. It is open-source, highly customizable, and supports the integration of multiple technologies. JAVA[18] must initially be installed in order to run this application. Appendix F contains instructions on how to install OpenHAB and Java.

User Interface

Once the installation was completed, a user interface was created for Bob and Alice, simulating the various floors of a smart building, as well as some of the items in each floor. This interface has both

[16] https://github.com/bclse/dissertation/
[17] https://www.openhab.org/
[18] https://www.java.com/

visualization and control purposes. Certain rules have been created that shape the user interface. An example can be found in the transition from figure 20 to figure 21. The code developed for the UI is available in the Git repository[19] and in Appendix G.

Figure 19 demonstrates the Bob's user interface. The item that will be controlled will be located within the Conference Room in First Floor. There are two significant items in the Conference Room. While the Lights item is the light itself, the Control Lights item is what gives control of the Light to Alice. If Bob wishes to transfer control of a light in his building to Alice, he needs to click on the "Control the Lights" button. This action will automatically trigger the creation of a contract.

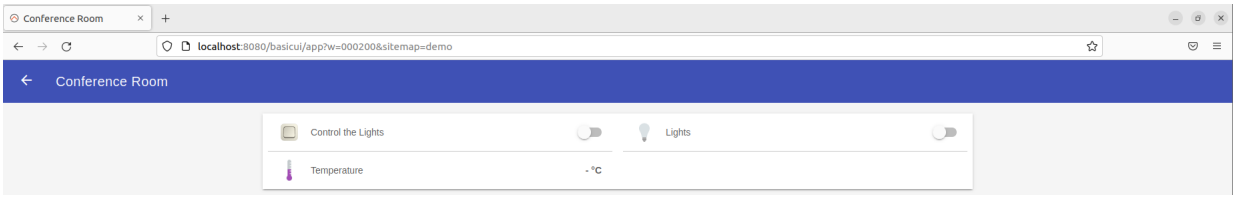


Figure 19. Bob User Interface.

Figure 19 displays the Alice's user interface. Aside from the divisions, this UI includes a control section, as shown in Figure 20.

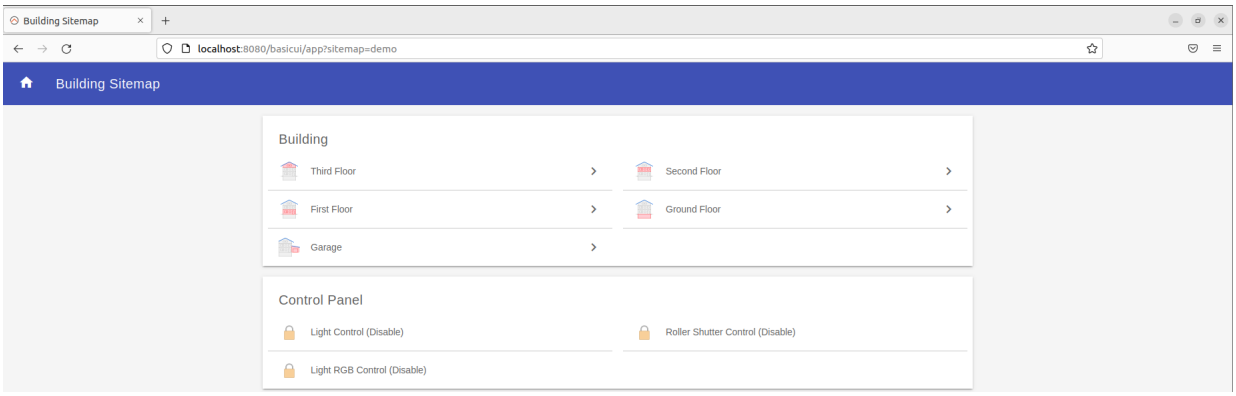


Figure 20. Alice User Interface Without Control.

If Bob wishes to transfer control of a light in his building to Alice, he must create a smart contract that specifies the entity Alice. If a smart contract is created that assigns control of a specific light to Alice, her Control Panel will be updated and the control functionality will be visible. Figure 21 depicts how the UI responds to the delegation of control to Alice. A switch with the smart building's light state is set to visible, for Alice to control, while the previous item is made invisible.

If Alice wishes to control the light, she needs to click on the "Lights Control (Enable)" button. This action will automatically trigger the exercise of the previous contract, creating a new contract with the required information, and archiving the old one.

[19] https://github.com/bclse/dissertation/

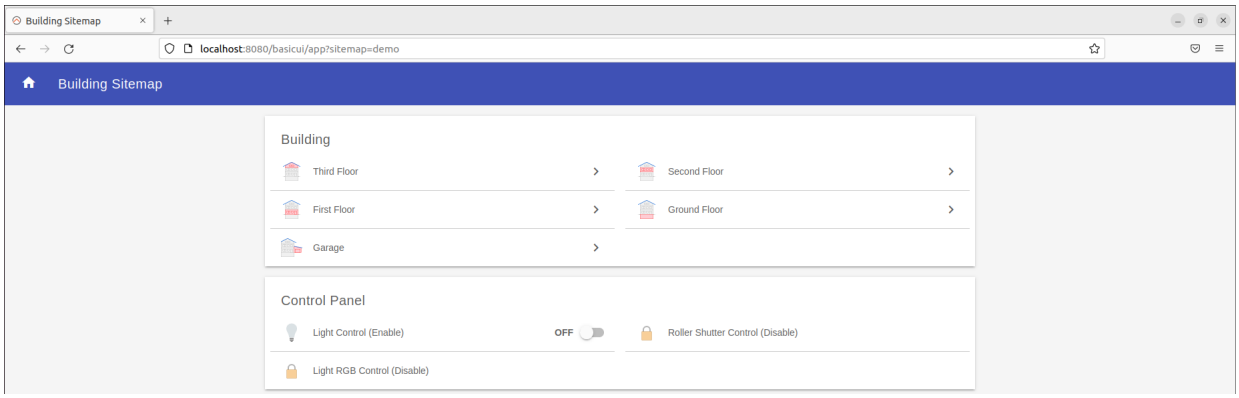


Figure 21. Alice User Interface With Control Granted.

4.6. Communications

The communication protocols used between the various systems when implementing the solution ultimately fall on HTTP and MQTT. The HTTP protocol was chosen to be used in the majority of the solution because the smart contracts application uses it as its communication model. It is also quite simple to create a HTTP request and read a HTTP response using the SCIApp. However, MQTT technology is also used due to its uniqueness. It allows the SCIApp not to fall over an active wait until it receives the message coming from the IoT platform. By doing this, this communication can be carried out with fewer computational resources. The HTTP protocol could also be used for all system communications, but that would force the SCIApp into active waiting for the IoT platform's HTTP response, which would consume more computational resources. Next, the various communication protocols used throughout the system are discussed.

Smart Thing <-> OpenHAB

Regarding the connection between the IoT Platform and smart things, it can be achieved using different protocols, depending on the characteristics of the device, the environment, etc. Protocols like ZigBee, Z-Wave, Bluetooth and Wi-Fi can be used to perform this communication. In this system as there were no physical objects to connect, these protocols will not be used.

OpenHAB <-> SCIApp

In order to transport upstream data from the IoT platform to the SCIApp (1), the MQTT protocol was chosen. MQTT stands for Message Queuing Telemetry Transport, and it is one of the most popular messaging protocol for the IoT. MQTT is used to send and receive messages and data between devices. This protocol uses the publish/subscribe pattern to connect the devices (3)(4). Topics are used for communication between the sender (Publisher) and the receiver (Subscriber), who are separated from one another (2). The connection between them is handled by the MQTT Broker. The MQTT Broker filters all incoming messages and distributes them correctly to the subscribers. Figure 22 presents the MQTT connection between OpenHAB and the SCIApp.

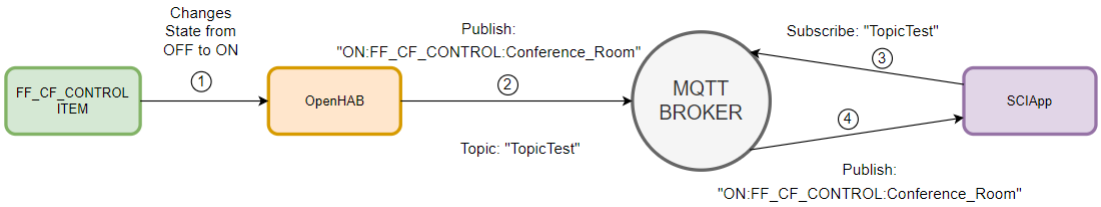


Figure 22. MQTT Connection.

The goal of the proposed solution was to change the ON/OFF state of a light between two entities by means of a smart contract. An example of a different implementation, would be a contract to check only the state of a light. In this case, a specific message containing the light's state and other information would probably be generated and published in the Broker by OpenHAB, where the SCIApp would subscribe. This example could only be accomplished in this manner. A different approach would also have to change the message sent by OpenHAB to the SCIApp. The configuration for such implementation is described in the following paragraphs and pictures.

Installing a Broker is required before configuring an MQTT connection. The open-source nature of the Mosquito[20] Broker led to its selection. How the Broker was set up and authentication added are described in Appendix H. Before adding the Broker to OpenHAB, the MQTT Binding, which is available in the OpenHAB application, must be installed. Bindings integrate physical hardware, external systems and web services in OpenHAB. The configuration of the MQTT Broker in OpenHAB is displayed in the figure 23 and figure 24.

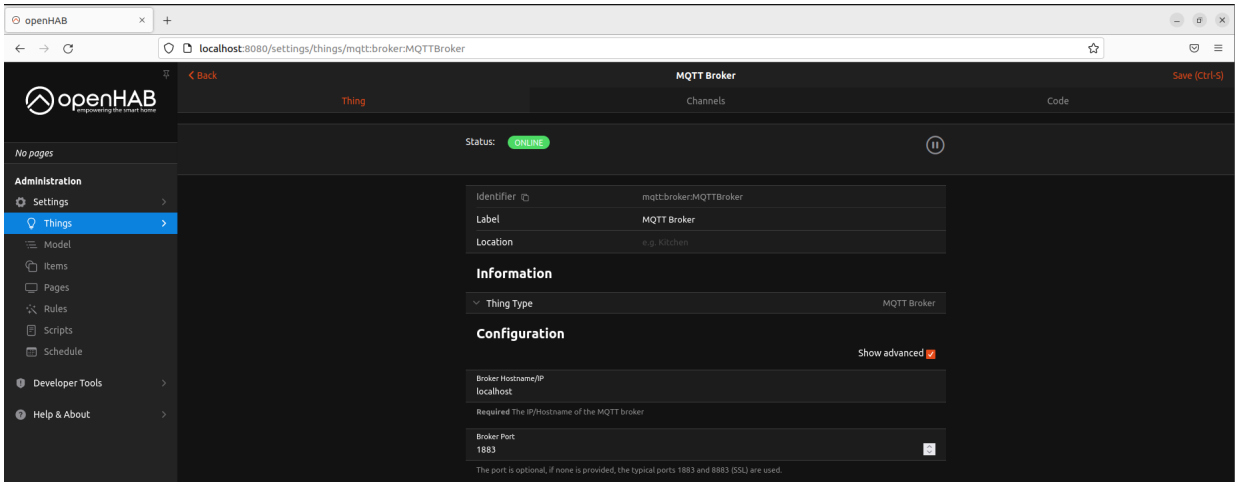


Figure 23. Configuration of the MQTT Broker in OpenHAB.

After the Broker has been set up, a Thing must be created, in this case an MQTT Thing, which must also have its configurations set up as seen in figure 25. The physical layer of an OpenHAB system is represented by Things. Things specify to OpenHAB which physical entities (devices, web services, information sources, etc.) should be managed by the system from a configuration perspective. Things

[20] https://mosquitto.org/

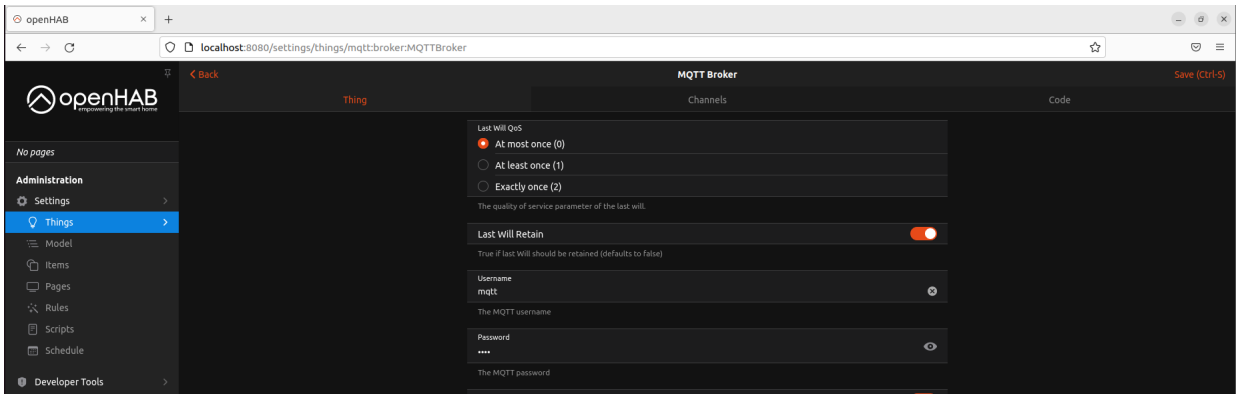


Figure 24. Configuration of the MQTT Broker in OpenHAB.

are connected to OpenHAB through bindings. Each Thing provides one or more Channels to access its functionality. These Channels can be linked to items. Items are used to control Things and consume their information. Ultimately, when Items are linked to Channels on a Thing, they become available to the various user interfaces and to the rules engine.

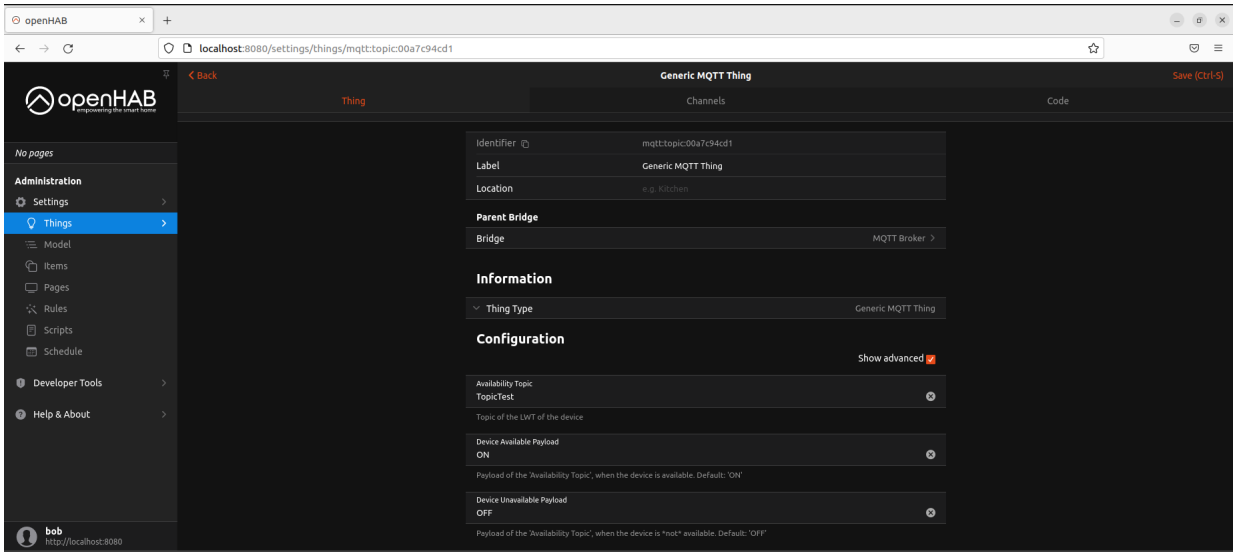


Figure 25. Configuration of the MQTT Thing in OpenHAB.

An item must be connected to the previously created MQTT Thing in order for the SCIApp to be able to subscribe to it. This connection is accomplished via a channel that must be properly configured as demonstrated in figure 26.

The item FF_CF_CONTROL is the one that must be linked to the Channel Thing because it corresponds to the state change that the SCIApp wishes to subscribe to. The item FF_CF_CONTROL could also be identified in the field place, within the smart contract. This configuration is shown in figure 27. This message exchange's topic is referred to as TopicTest. Whenever the item FF_CF_CONTROL changes its state to ON, the following message will be published in the Broker: "ON:FF_CF_CONTROL:Conference_Room" if it is set to OFF, only the first argument will change, i.e. "OFF:FF_CF_CONTROL:Conference_Room". The SCIApp will later subscribe to these messages.

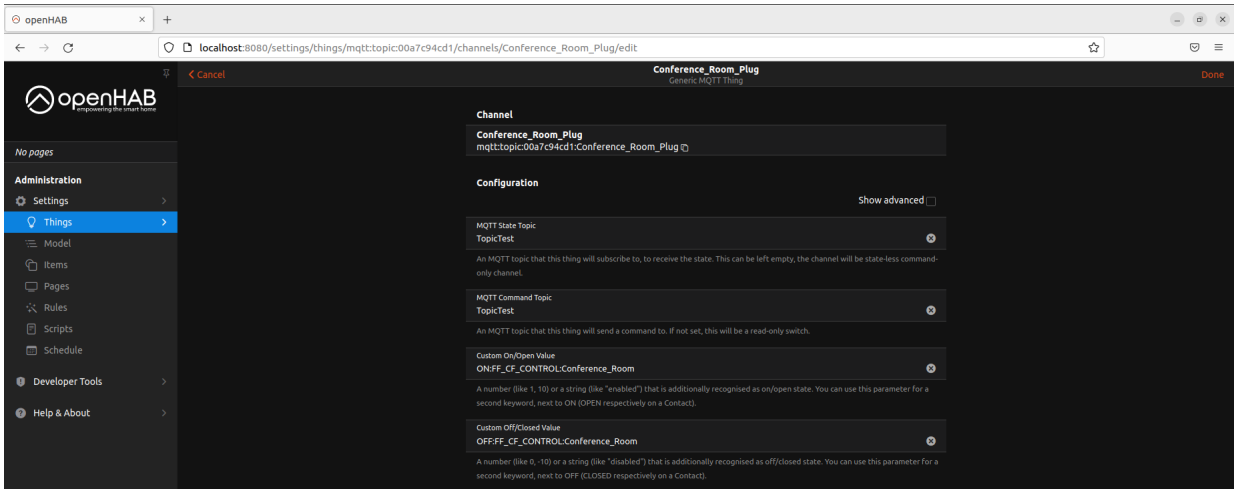


Figure 26. Configuration of the Item linked to the Thing.

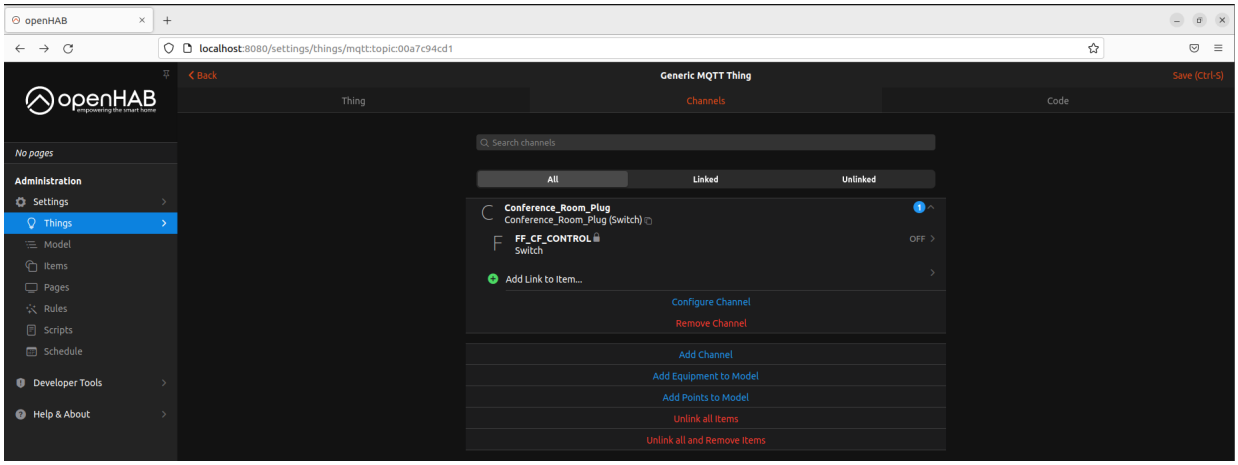


Figure 27. Item linked to the Thing.

In order to transport downstream data from the SCIApp to the IoT platform, the HTTP protocol was chosen. The purpose of this communication is to acquire the state of items on the IoT platform as well as to update them. To that goal, SCIApp creates an HTTP message with a particular function.

SCIApp <-> DAML

As already mentioned, DAML uses the HTTP protocol as its communication model so it was used in the communication between SCIApp and DAML. SCIApp generates HTTP messages to communicate with DAML and also processes HTTP messages that DAML sends to SCIApp.

DAML <-> DAML

The communication between the various Daml applications on the three nodes is carried out via the Canton Protocol. The Canton protocol is the technology which synchronizes participant nodes across any Daml-enabled blockchain or database. The Canton protocol enables transactions between Daml

applications as well as their portability between various underlying synchronization systems.

4.7. Control Modifications

To summarize, this implementation was a concrete case for controlling a light in a smart building, but if other services for controlling or monitoring through smart contracts were necessary, several steps would be required. The procedures listed below would need to be followed in order to build a new control:

1. Bob's OpenHAB: In the upstream connection, the MQTT message to be sent to SCIApp should be restructured to include the information needed to create the contract. In the downstream connection, it would be necessary to review the items that will be updated.
2. Bob's SCIApp: In the upstream connection, the processing in SCIApp must be reviewed and changed. SCIApp has a dedicated processing for the light control's MQTT message. When changing the MQTT message to implement a new control, the processing in SCIApp must also be changed. The same thing happens in the downstream connection. The processing of the HTTP message from DAML will need to be reviewed because a different contract with different characteristics will be implemented.
3. Bob and Alice's Daml: To accomplish the control's goal, the contract's structure needs to be altered. This will necessarily require changing the contract code.
4. Alice's SCIApp: In the upstream connection, the processing in SCIApp has to be revised. The HTTP message coming from Daml will be different from what is implemented, so to perform the updates correctly, the processing of the HTTP message, in SCIApp, will have to be changed. In the downstream connection, the processing of the MQTT message will also need to be changed. The MQTT message of the new control will differ from the one implemented. In order to send the correct information to the Daml, to exercise the contract, the processing of the MQTT message in SCIApp will need to be modified.
5. Alice's OpenHAB: In the upstream connection review the items that will be updated. In the downstream connection, the MQTT message to be sent to SCIApp must be restructured to contain the necessary information to exercise the contract.

CHAPTER 5

Results and Tests

The results gathered from the tests performed are presented and discussed in this chapter. This chapter is organized as follows, section 5.1 presents the results of functional tests performed on the proposed system. In section 5.2 the results of performance tests conducted on the system are presented. The setup displayed in table 3, detailed in the previous chapter, was implemented for running these tests.

5.1. Functional Test

This section presents the functional tests conducted on the proposed system. The goal of this test is to determine if the proposed system is decentralized and if can implement interactivity functionalities. In this test, Bob grants Alice the control over a light. Then, Alice changes the state of the light, which is then updated in the smart building. Were extracted timestamps for all system applications since the control of a light is given to Alice, until the new state defined by Alice is updated in the smart building. The time instants of this test were collected in log files in the case of OpenHAB and DAML and implemented in strategic places in the SCIApp code. Three different tests were performed: a functional test (which has already been described), a performance test of the SCIApp connection to OpenHAB, and a test in which the same functional test was run ten times with average times recorded. Figure 28 illustrates the time spent on each system application, from the start to the end of the test. An analysis of this figure will be discussed in the following paragraphs.

Validation of the integration with the time spent in each system block

The figure 28 illustrates the time spent on each application that runs in Bob's node (in green), the Domain node (in blue), Alice's node (in orange). The upstream connection will now be discussed. The first block of time in the test corresponds to OpenHAB and lasts 19 ms. Pressing the button that will give the third entity control corresponds to the zero instant. The time when the button is pressed is recorded in a text file after 19 ms.

The next period of time, known as the SCIApp, has a duration of 276 ms. This is where the information is processed and forwarded to the DAML in order to create a smart contract. The information that the control has been granted to a third entity, sent through the MQTT protocol, takes 32 ms to reach the SCIApp. At 51 ms, the SCIApp receives the MQTT message containing the location of the light in the smart building and the reference of the item represented in OpenHAB. These two parameters, along with the current state of the light, are required to create the contract.

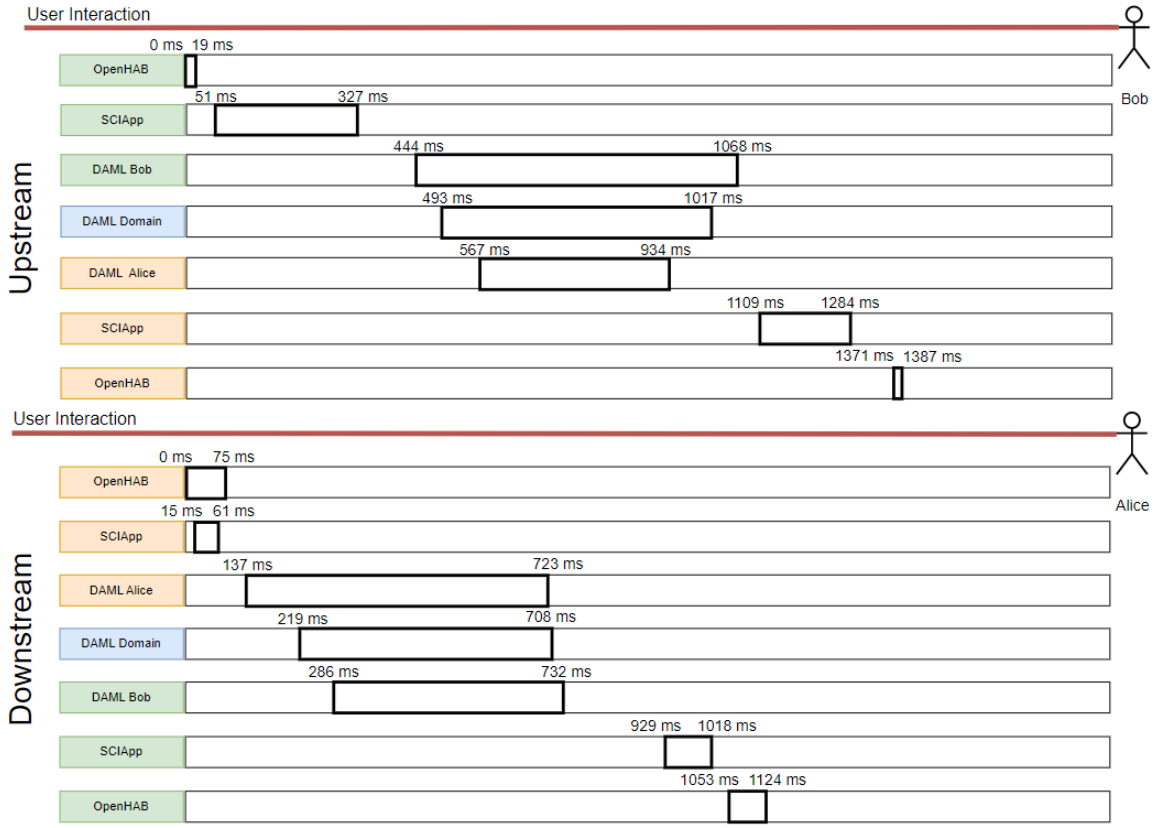


Figure 28. Time spent in each system block.

276 ms after receiving the MQTT message, i.e. at instant 327 ms, SCIApp transmits the information required to create the contract to DAML through HTTP. This process takes some time because the SCIApp needs the current state of the light that the third entity will control to send this message. To acquire this state, during this period, an HTTP message must be sent to OpenHAB which leads to additional time spent on the SCIApp.

A Smart Contract is created at instant 444 ms, i.e, 117 ms after the SCIApp has sent the information to the DAML via HTTP.

This information is delivered to the node of the domain 49 ms later, i.e. at time 493 ms through the Canton Protocol.

Next, the node of the domain communicates the contract to Alice's node, also through the Canton Protocol, 74 ms later, i.e. at time 567 ms. The contract processing is not immediate. Figure 28 shows exactly that, each node varies in the duration of processing the contract.

At time 1109 ms, the SCIApp of Alice's node receives the information that a contract has been created. It takes 175 ms to process the information coming from DAML, through HTTP, and to update the corresponding items in OpenHAB also through an HTTP connection. This process ends at time 1257 ms.

After 87 ms, i.e. at time 1371 ms, OpenHAB receives the information to update its items, which will then be visible to Alice to control the light. The next 16 ms elapsed in OpenHAB are due to the

time it takes to write in a text file when the items were updated. The total time spent on the upstream connection was 1387 ms.

The downstream connection will now be discussed. The user will now interact with the system at this point. Alice can change the state of the light from the OpenHAB platform, once the control is available. The button click that causes the controlled light to change its state corresponds to the zero instant. The OpenHAB block of Alice's node has a duration of 75 ms. It takes a little longer to complete this process because in addition to communicating with the SCIApp via MQTT that the light state has changed, this timestamp is also stored in the text file. Moreover, in addition these two actions, it is also necessary to set the control buttons to invisible in order to remove the Alice's control capability. This last action is triggered by the SCIApp so it is expected that OpenHAB process will last longer than the SCIApp one.

The SCIApp receives the data from OpenHAB over MQTT at instant 15 ms. SCIApp processing takes 46 ms, and, at instant 61 ms it sends the information, that Alice controlled the light, to DAML, via HTTP. Additionally it requests an update of the item from OpenHAB, via HTTP, in order for it to become invisible.

After 76 ms from the moment the SCIApp has sent the information to the DAML via HTTP, the previous smart contract is archived and a new smart contract with new information will be created from instant 137 ms.

This information is delivered to the node of the domain 82 ms later, i.e. at time 219 ms through the Canton Protocol.

Next, the node of the domain communicates the contract to the Bob's node also through the Canton Protocol, 67 ms later, i.e. at time 286 ms. The contract processing is not immediate.

The SCIApp in Bob's node learns that the contract was exercised at time 929 ms. This data is processed and submitted to OpenHAB via HTTP after 89 ms, i.e at instant 1018. The light that has been granted control will then be updated with the state defined by Alice, and the button that grants this control will also be updated to OFF.

OpenHAB updates the desired items at time 1053 ms, i.e. 35 ms after the SCIApp has sent it this information through HTTP. It takes 71 ms to update the two items and to write to a text file the instant of this update. The total time spent on the downstream connection was 1124ms. The system's total processing time, which is calculated as the sum of the upstream and downstream connections, is 2511 ms.

The presented results demonstrate that the proposed system is decentralized, in other words, it does not require an entity at the top of a hierarchy to give orders to everyone. Bob can give instructions to Alice or to any other external entity, and the inverse can also occur. Regarding the interactivity functionality, the system times suggest that to control devices like radiators, washing machines or even the watering of a garden, i.e. devices that do not require immediate response, and feedback to users, the system is acceptable. The 2511ms test duration serves this purpose. However if it is

necessary to control devices that require immediate response, such as turning a light on or off, this system is not appropriate. The system takes too long for this particular case, where delay only up to a few hundred of milliseconds is tolerated.

A closer examination of the test leads to the conclusion that the Daml application times are independent and cannot be controlled. On the other hand, certain times that correspond to the SCIApp block can be controlled. Then, two situations were verified to understand where more time was spent. Two tests were performed ten times to determine how long it takes for the SCIApp to acquire the status of an item to OpenHAB (in this case, the light) and, also how long it takes for an item to be updated in OpenHAB by the SCIApp. In the first test (GET), it is necessary to acquire the time when the HTTP message is sent to OpenHAB and when the SCIApp receives the HTTP message with the item state. This situation occurs in Bob's SCIApp when it is necessary to acquire the current state of the light, to create a contract. In the second test (UPDATE), it is necessary to acquire the time when the SCIApp sends the HTTP message to OpenHAB and when OpenHAB actually updates the state of the specified item. This situation occurs in the Bob's SCIApp and the Alice's SCIApp when various items need to be updated in the OpenHAB control interface. The tests results are displayed in Table 5.

Table 5. Average time of the UPDATE and GET test.

	GET	UPDATE
Average Time [ms]	37.6	32.7

After analyzing the results, a different approach to reducing these times could be considered for future implementation.

Average time to reach next application

To further understand the average time it takes for the information to travel between applications, this functional test was repeated ten times. The goal of this test is to determine if interactivity functionalities can be implement and if the response time is too excessive for the control of electricity production and consumption in smart buildings. This test is the same as the functional test shown in figure 28, but it was performed more times and an average of the results was obtained. Now a different view of the results is presented. The average amount of time it takes for the information to go from one application to the next in the upstream and downstream message flow is shown in Figure 29.

When compared to other applications, the exchange of information from one to the next takes longer during the communication between DAML and the SCIApp, in both Bob and Alice's node. The SCIApp is connected to the Ledger SCIApp passively waiting for a new contract to arrive which makes this process time consuming. The exchange of information between the SCIApp and OpenHAB, also in the two nodes mentioned above, is another time-consuming process for the system. Due to the SCIApp's requirement to update the OpenHAB items via HTTP, this process also takes a significant time. Aside from these high delay when exchanging information between two applications, the remaining delays are quite reasonable for an iterative system, and fall under 200ms.

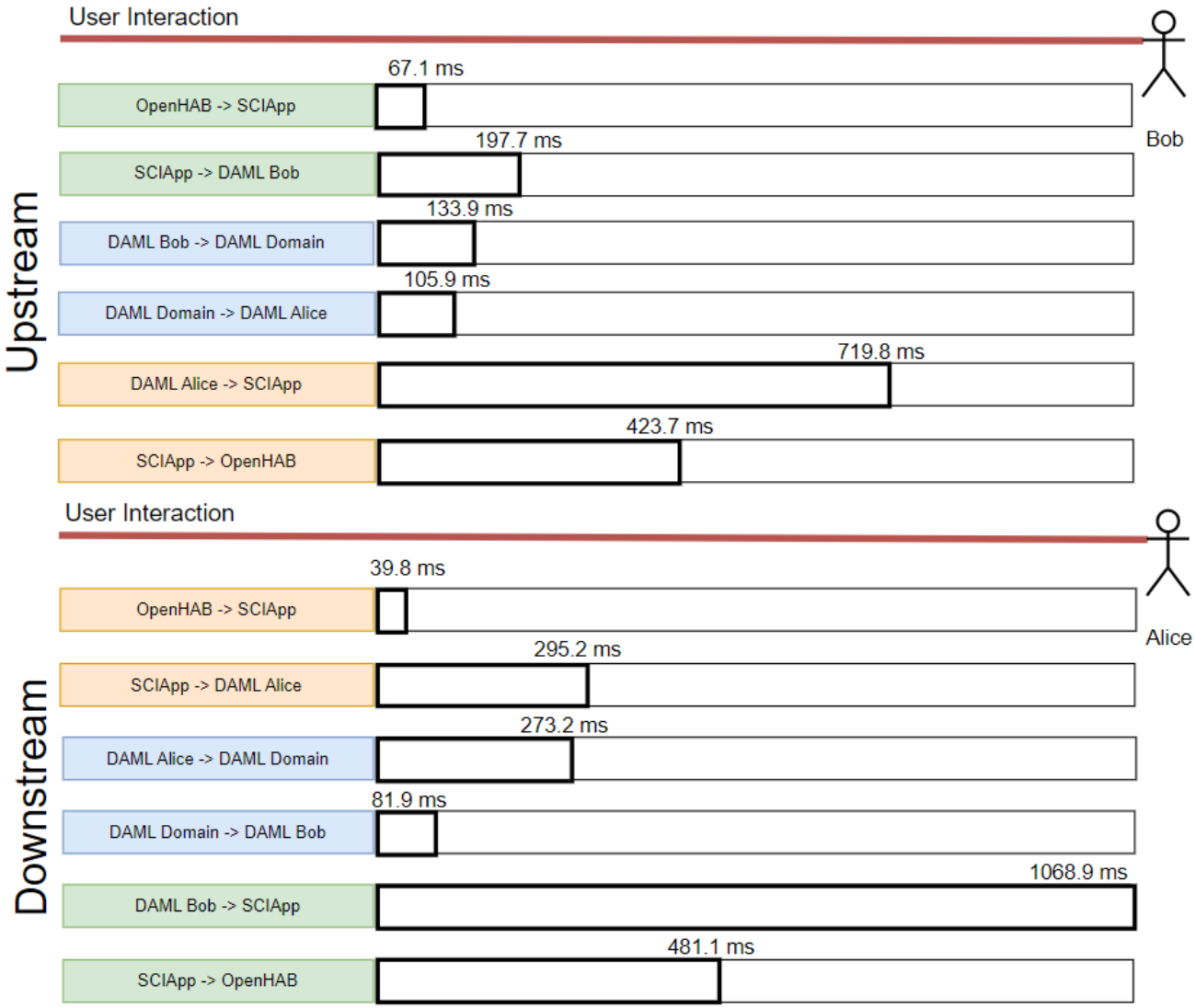


Figure 29. Average time to reach the next application.

The processing delay for each application was also collected by analyzing the results of the previous test, as shown in table 6.

According to the table's results, it is possible to conclude that the DAML applications are consuming the majority of time across all nodes. This occurs as a result of the intrinsic characteristics of the DAML application. In the first part of the test, it ranges from 360.1 ms to 892.2 ms, and in the second part, it ranges from 498.6 ms to 707.8 ms. Due to the HTTP message that was sent to OpenHAB in order to update the appropriate items, the times are longer in both the first and second parts of the test on the third party SCIApp and the smart building SCIApp, respectively. Another time that is a bit longer than expected and could be improved in a future work, is the smart building SCIApp in the first part of the test. This is due to the need to acquire the current state of the light by HTTP.

The results of the ten functional tests performed will also provide information about the upstream and downstream connection times. This information is illustrated in table 7.

Table 6. Average time spent on each application.

Application	Time Spent [ms]	Standard Deviation
Bob Interaction	–	–
OpenHAB Bob	19.3	7.0
SCIApp Bob	138.5	50.6
DAML Bob	892.2	280.7
DAML Domain	672.1	155.1
DAML Alice	360.1	130.1
SCIApp Alice	328.4	149.7
OpenHAB Alice	8.5	5.7
Alice Interaction	–	–
OpenHAB Alice	90.9	41.5
SCIApp Alice	40.0	22.7
DAML Alice	707.8	249.4
DAML Domain	527.8	98.6
DAML Bob	498.6	105.2
SCIApp Bob	424.5	278.7
OpenHAB Bob	74.6	40.3

Table 7. Average time of upstream and downstream connections.

Connection	Time Spent [ms]	Standard Deviation
Upstream	1747.2	413.1
Downstream	2314.7	793.3

The presented results demonstrate that the proposed system is suitable for functionalities that doesn't require an immediate response like turning on a washing machine or determining the status of an item. However, in the case of functionalities that demand fast reaction or feedback, such as turning on and off a light, the response time is too long for a user to be waiting for the light to change. One way to make the system more interactive is to decrease the time spent in the SCIApp by using a more effective implementation. Regarding the response times, to operate an intelligent building's functions, specifically the electricity production and consumption, the response times of the system are sufficient and adequate, where a reaction time of up to a few minutes is tolerated.

5.2. Performance Tests

This section contains the performance tests carried out in the system. The purpose of these tests is to understand how the system responds to the creation of successive contracts. This test tries to verify how scalable this solution is by observing if the response time degrades as more contracts are created. First is described the stress tests without network delay and then the stress tests with delay are presented.

Only the upstream connection will be tested in this test. In order to conduct this test it was necessary to automate the process to which Bob grants control of a light to Alice. The button that gives control to Alice is turned ON, at a specific time, according to a rule that was created in the Bob's OpenHAB. In this test, a contract was created for each guaranteed control. It was extracted

the time in only two situations. The first, in Bob's node, when the control of a light is given to Alice in OpenHAB, i.e, the moment when the control button is pressed. The second, in Alice's node, when Alice updates its control panel also in OpenHAB, i.e, the instant when the control button is visible to Alice control the light. The difference between the two instants will be measured.

Upstream Contract Creation Stress Tests without Network Delay

The purpose of these tests is to determine how the system reacts when an excessive number of contracts appear at once. The following scenario was simulated for the performance tests: Bob will create a smart contract giving control to Alice, and her control panel will be updated with the control option. Four tests were conducted, with the only difference between them being the contract creation time. Thus, fifty contracts will be created every 500ms, 1s, 2s and 20s. Figure 30 depicts the moving average time difference of those fifty tests, between the Bob's creation of a contract and the update of Alice's OpenHAB dashboard, over the last five executions. One sample contains five results.

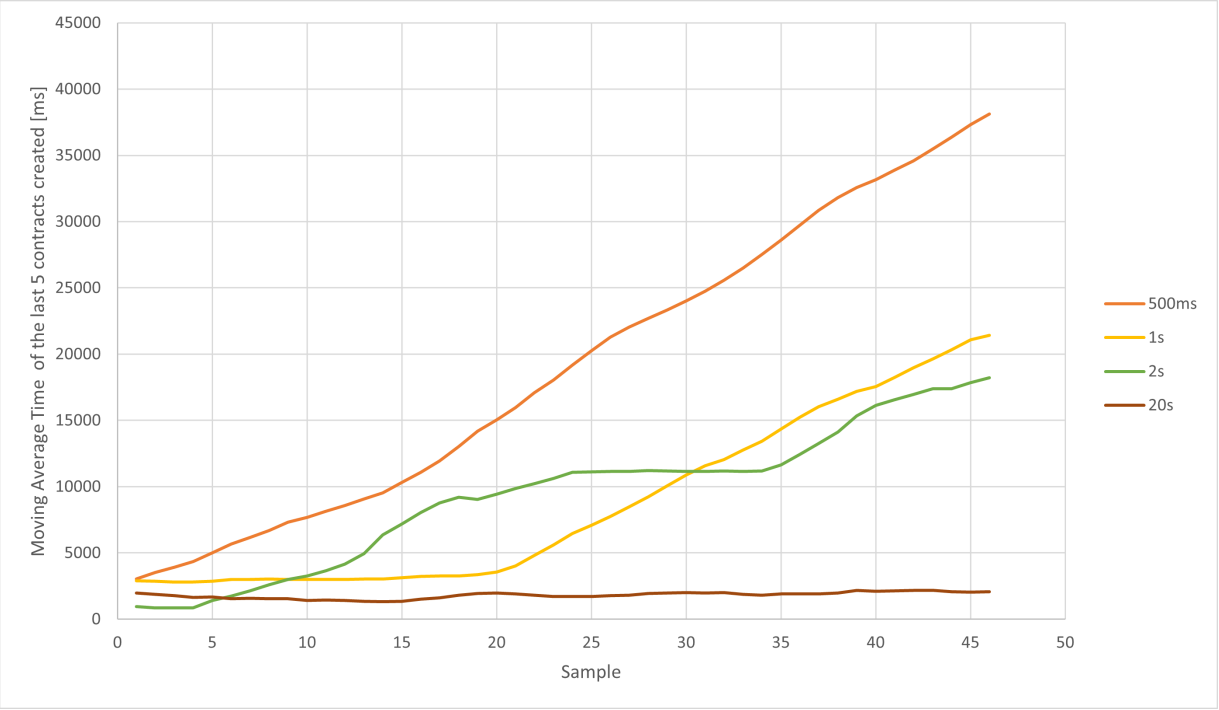


Figure 30. The moving average of the time it takes for a contract to become available on Alice's node, for the last 5 contracts, without delay.

For this test, time intervals smaller than 500ms were also tested. However it was observed that the contracts would not arrive with the right order, making it impossible to acquire coherent results. Therefore, the smallest time interval chosen for the tests was 500ms. The orange, yellow, green and brown colours represent the moving average time of the last five contracts when they are created every 500ms, 1s, 2s and 20s respectively. In figure 30, the 2s line over passes the 1s line between 10<Sample<30. The standard deviation can be used to explain why this is not the expected situation. The upstream connection's standard deviation is high enough that the lines could potentially cross

and overlap. This type of circumstance is possible due to the great variability of the results. Another reason is the moving average's sample size. Since there is a lot of variation in the results and there are only 5 results in one samples, having a highly distinct result can change the average. In addition to the reasons already mentioned, the fact that the 1s and 2s values are so near can also cause this event. Figure 30 illustrates what it would be expected: as more contracts are created in less time, information takes longer to get from one side to the other. This is due to the fact that more processing is required in a shorter period of time. In the opposite case, when contracts are created every 20s, the system has time to process all of the information because it is not overloaded. As a result, it takes less time for information to travel between nodes, and the times remain constant throughout the test.

It is then concluded that the more contracts are created in a shorter time, the more time degrades, i.e. the reaction time becomes gradually longer. In a normal usage scenario it is not expected that so many contracts would be created simultaneous, at least by Bob. Other tests, such as having several Bobs creating contracts with Alice, could be performed in a future work.

Upstream Contract Creation Stress Tests with Network Delay

The purpose of these tests is to determine how the system reacts when an excessive number of contracts appear at once. This test will provide a brief overview of the system's behaviour in a real world implementation. The added delay simulates how the system would operate in real life. The previous scenario was also simulated for the delay tests. Four tests were also performed using the previous times, however it was imposed to the domain node a 400ms delay. To be able to insert a certain delay or latency it is necessary to configure the virtual machine. The VMware software allows to configure this option very easily. Just go to Network Adapter Advanced Settings to configure this option. For this specific test it was added a latency of 400 ms for the Outgoing Transfer in the domain node. For future work it could also be configured other options like the percentage of lost packets. Figure 31a) shows the available options to configure and figure 31b) depicts where the delay is imposed on the domain node connections.

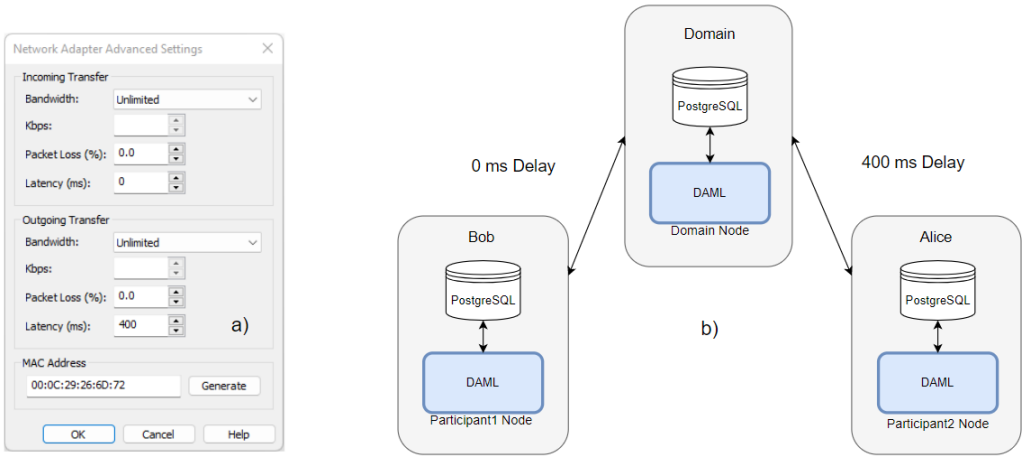


Figure 31. Delay Settings (a) and Delay Connections (b).

Regarding the tests results, figure 32 demonstrates the moving average time difference of fifty tests, between Bob's creation of a contract and the update Alice's OpenHAB dashboard, over the last five executions.

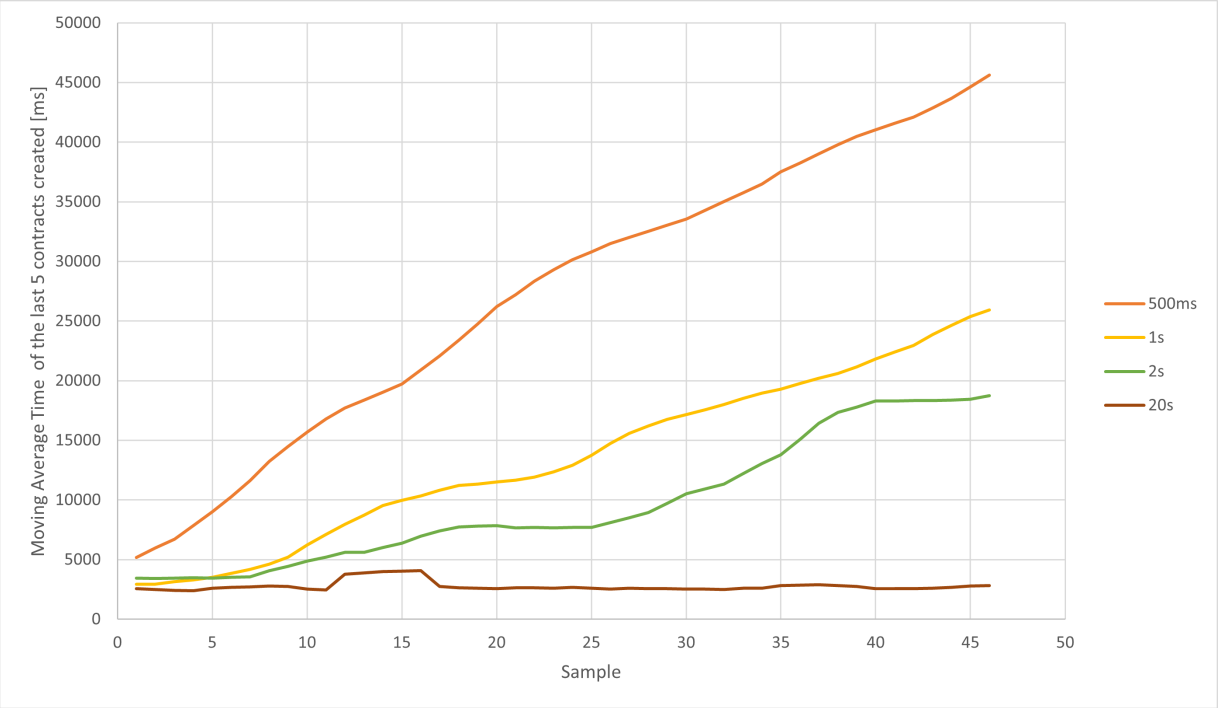


Figure 32. The moving average of the time it takes for a contract to become available on Alice's node, for the last 5 contracts, with delay.

The orange, yellow, green and brown colours represent the moving average time of the last five contracts created every 500ms, 1s, 2s and 20s respectively, with a 400ms delay in the domain node. When compared to tests without delay, the same effect occurs: if more contracts are created in a shorter period of time, the information takes longer to arrive from one side to the other. Also, when the system is not overloaded, in the case of the 20s test, the time between the creation of the contract and the update on the OpenHAB platform remains constant. However, because of the 400ms delay, the information takes even longer to perform the upstream connection. This is demonstrated in the figure 33, which compares the 500ms creation time of contracts with and without delay. Another test was conducted with a 200ms delay as a comparison between no delay and 400ms delay.

The same conclusion can be drawn from the testing without delay, namely that the more contracts that are created in a shorter time, the time degrades, resulting in longer reaction time. The only difference in the delayed tests is that they still take longer because of the delay.

If the performance tests had been carried out using the downstream connection, the results would not have led to very different conclusions. Additionally, the contracts implemented in this solution are expected to be performed in small communities, and thus the presented scalability is adequate in these scenarios.

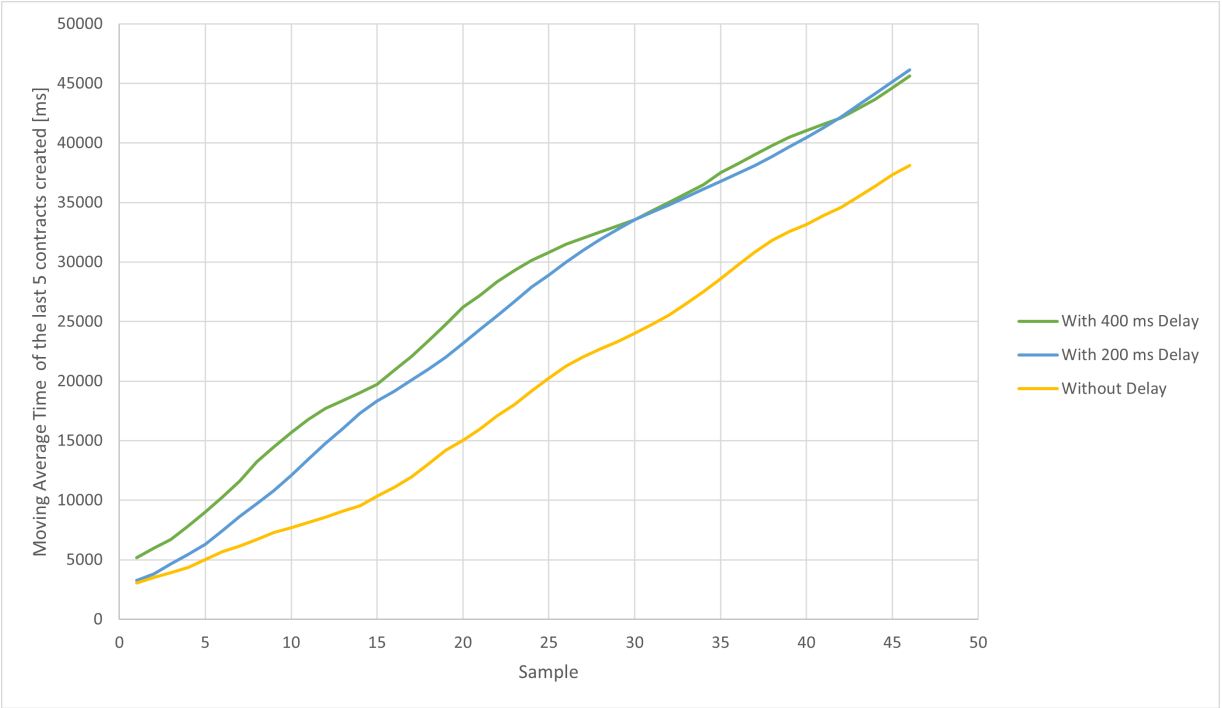


Figure 33. The moving average of the time it takes for a contract to become available on Alice's node, for the last 5 contracts, with and without delay.

CHAPTER 6

Conclusion

The main research question of this dissertation was whether it was feasible to integrate Smart Contracts in order to achieve decentralised management of smart buildings, regarding services such as energy management. With the results obtained, it is possible to conclude that it is feasible to build a system that integrates smart contracts with smart buildings, using open-source technologies, in order to achieve decentralised management of smart buildings.

The developed system allows an entity in charge of a smart building to assign an external party control over a specific component of the building, such as a smart light, through the use of smart contracts.

This system demonstrates that smart contracts can be implemented to provide a third party control over some smart building operations. The first test in section 5.1 clearly demonstrates that integrating smart contracts with IoT for distributed control of a smart building is feasible. This conclusion is supported by the results obtained in section 5.1 where the test duration time is 2511 ms.

The other research question was if the response time is too excessive, in a system that integrates Smart Contract technology with IoT. In response to this question, it is concluded that, depending on its application, the proposed system is either time excessive or not. When the various results are analyzed, it is found that the system does not have excessive response times for applications with response times between one and two seconds. However, this system won't be suitable for a real-time application. It is demonstrated in the tests carried out in section 5.2 that the response time grows when an increasing number of contracts are created at a predetermined frequency of time.

The third research question was if it was possible to implement functionalities that require interactivity in this system without worrying about the delay. In the case of the implemented solution, that aims to turn on and off an intelligent building light, the response times are a bit high. However, in the context of intelligent building remote control, the time spent controlling a light is perfectly acceptable. The response times are perfectly adequate in other situations, such as turning on a washing machine or an air conditioner. The control function is possible for operations that do not require a real-time response time. For functionalities that require a real-time response time, this system will not be the most suitable as it comes with a significant delay time. For distributed control of intelligent buildings, immediate feedback is not so relevant, as controls are supposed to be carried out remotely. This conclusion is supported by the results obtained in section 5.1 in table 7, where the average test duration time is 4061 ms.

The fourth and last research question was whether this type of integration is viable in small single-board computers with the complexity and processing that this system requires. The system was not implemented in small single board computers, but the virtual machines used for testing have properties similar to small single board computers. The characteristics of each VM are described in table 3. It is reasonable to assume that this system might work on small single board computers even without testing, considering the characteristics of the VMs used.

After reviewing the tests results, it was determined that the solution could be used for a variety of purposes other than granting control functionality. The proposed system, with appropriate changes can serve to check the status of an item in the smart building, to change the status of other items like air conditioner, power outlets etc. These system can improve item automation and trust in external entities.

6.1. Future Work

The future work is organized in two suggestion: Upgrades to the system; Deployment of the solution in a blockchain.

First, it is suggested how the system, more specifically the Application, might be optimized. Acquiring the current state of the light that will be controlled is a time-consuming process. This procedure involves sending an HTTP message to OpenHAB and waiting for an HTTP response indicating the current state of the light. One way to improve this process is to include the current status of the light in the previous MQTT message. Implementing this solution would result in shorter response times.

The goal of the final suggestion is to investigate the system's behavior if it were integrated into a blockchain rather than a database. In this manner, it would be possible to later compare the testing times for the system using the two approaches.

References

[1] M. K. M. Shapi, N. A. Ramli, and L. J. Awalin, "Energy consumption prediction by using machine learning for smart building: Case study in malaysia," *Developments in the Built Environment*, vol. 5, Mar. 2021, ISSN: 26661659. DOI: 10.1016/j.dibe.2020.100037.

[2] J. J. Hunhevicz, M. Motie, and D. M. Hall, "Digital building twins and blockchain for performance-based (smart) contracts," *Automation in Construction*, vol. 133, Jan. 2022, ISSN: 09265805. DOI: 10.1016/j.autcon.2021.103981.

[3] D. Ibaseta, A. García, M. Álvarez, B. Garzón, F. Díez, P. Coca, C. D. Pero, and J. Molleda, "Monitoring and control of energy consumption in buildings using wot: A novel approach for smart retrofit," *Sustainable Cities and Society*, vol. 65, p. 102 637, Feb. 2021, ISSN: 22106707. DOI: 10.1016/j.scs.2020.102637.

[4] S. S. K. Ali Dorri and R. Jurdak, "Blockchain in internet of things: Challenges and solutions," 2016. DOI: https://doi.org/10.48550/arXiv.1608.05187.

[5] S. Ashraf, "A proactive role of iot devices in building smart cities," *Internet of Things and Cyber-Physical Systems*, vol. 1, pp. 8–13, 2021, ISSN: 2667-3452. DOI: https://doi.org/10.1016/j.iotcps.2021.08.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266734522100002X.

[6] S. Sinha, "State of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot now surpassing 2 billion," 2021. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/%E2%80%9D.

[7] C. Plaza, J. Gil, F. de Chezelles, and K. A. Strang, "Distributed solar self-consumption and blockchain solar energy exchanges on the public grid within an energy community," IEEE, Jun. 2018, pp. 1–4, ISBN: 978-1-5386-5186-5. DOI: 10.1109/EEEIC.2018.8494534.

[8] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *Journal of Network and Computer Applications*, vol. 177, p. 102 857, 2021, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102857. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520303234.

[9] A. O. Philip and R. K. Saravanaguru, "Smart contract based digital evidence management framework over blockchain for vehicle accident investigation in iov era," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 4031–4046, 2022, ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2022.06.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157822001719.

[10] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016, ISSN: 21693536. DOI: 10.1109/ACCESS.2016.2566339.

[11] A. Alfrhan, T. Moulahi, and A. Alabdulatif, “Comparative study on hash functions for lightweight blockchain in internet of things (iot),” *Blockchain: Research and Applications*, vol. 2, p. 100 036, 4 Dec. 2021, ISSN: 20967209. DOI: 10.1016/j.bcra.2021.100036.

[12] S. Saxena, H. Farag, A. Brookson, H. Turesson, and H. Kim, “Design and field implementation of blockchain based renewable energy trading in residential communities,” 2019, pp. 1–6. DOI: 10.1109/SGRE46976.2019.9020672.

[13] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian, “A survey on the adoption of blockchain in iot: Challenges and solutions,” *Blockchain: Research and Applications*, vol. 2, p. 100 006, 2 Jun. 2021, ISSN: 20967209. DOI: 10.1016/j.bcra.2021.100006.

[14] C. Laneve and C. Sacerdoti Coen, “Analysis of smart contracts balances,” *Blockchain: Research and Applications*, vol. 2, no. 3, p. 100 020, 2021, ISSN: 2096-7209. DOI: https://doi.org/10.1016/j.bcra.2021.100020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096720921000154.

[15] R. Kumar and R. Sharma, “Leveraging blockchain for ensuring trust in iot: A survey,” *Journal of King Saud University - Computer and Information Sciences*, 2021, ISSN: 22131248. DOI: 10.1016/j.jksuci.2021.09.004.

[16] A. Daissaoui, A. Boulmakoul, L. Karim, and A. Lbath, “Iot and big data analytics for smart buildings: A survey,” *Procedia Computer Science*, vol. 170, pp. 161–168, 2020, ISSN: 18770509. DOI: 10.1016/j.procs.2020.03.021.

[17] D. Kirli, B. Couraud, V. Robu, M. Salgado-Bravo, S. Norbu, M. Andoni, I. Antonopoulos, M. Negrete-Pincetic, D. Flynn, and A. Kiprakis, “Smart contracts in energy systems: A systematic review of fundamental approaches and implementations,” vol. 158, 2022, p. 112 013. DOI: https://doi.org/10.1016/j.rser.2021.112013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032121012764.

[18] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of Management Information Systems*, vol. 24, pp. 45–77, 3 Dec. 2007, ISSN: 0742-1222. DOI: 10.2753/MIS0742-1222240302.

[19] E. Kaku, “Using blockchain to support provenance in the internet of things,” 2017. [Online]. Available: https://harvest.usask.ca/handle/10388/8099.

[20] A. Koohang, C. S. Sargent, J. H. Nord, and J. Paliszkiewicz, “Internet of things (iot): From awareness to continued use,” *International Journal of Information Management*, vol. 62, Feb. 2022, ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2021.102442.

[21] M. Laroui, B. Nour, H. Moungla, M. A. Cherif, H. Afifi, and M. Guizani, “Edge and fog computing for iot: A survey on current research activities future directions,” *Computer Communications*, vol. 180, pp. 210–231, Dec. 2021, ISSN: 1873703X. DOI: 10.1016/j.comcom.2021.09.003.

[22] O. Omar, "Intelligent building, definitions, factors and evaluation criteria of selection," *Alexandria Engineering Journal*, vol. 57, pp. 2903–2910, 4 Dec. 2018, ISSN: 11100168. DOI: 10.1016/j.aej.2018.07.004.

[23] L. Pocero, D. Amaxilatis, G. Mylonas, and I. Chatzigiannakis, "Open source iot meter devices for smart and energy-efficient school buildings," *HardwareX*, vol. 1, pp. 54–67, Apr. 2017, ISSN: 24680672. DOI: 10.1016/j.ohx.2017.02.002.

[24] P. Rocha, A. Siddiqui, and M. Stadler, "Improving energy efficiency via smart building energy management systems: A comparison with policy measures," *Energy and Buildings*, vol. 88, pp. 203–213, Feb. 2015, ISSN: 03787788. DOI: 10.1016/j.enbuild.2014.11.077.

[25] E. S. Kang, S. J. Pee, J. G. Song, and J. W. Jang, "A blockchain-based energy trading platform for smart homes in a microgrid," in *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*, Apr. 2018, pp. 472–476. DOI: 10.1109/CCOMS.2018.8463317.

[26] O. Van Cutsem, D. Ho Dac, P. Boudou, and M. Kayal, "Cooperative energy management of a community of smart-buildings: A blockchain approach," *International Journal of Electrical Power & Energy Systems*, vol. 117, p. 105 643, 2020, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2019.105643. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061519323087.

[27] M. Afzal, Q. Huang, W. Amin, K. Umer, A. Raza, and M. Naeem, "Blockchain enabled distributed demand side management in community energy system with smart homes," *IEEE Access*, vol. 8, pp. 37 428–37 439, 2020. DOI: 10.1109/ACCESS.2020.2975233.

[28] B. Balamurugan, G. A. Al-Abri, M. Abbas, B. Balusamy, and S. Kadry, "Open source iot monitoring and controlling system using raspberry pi 3 and openhab to measure and control the electricity power consuming," *International Journal of Engineering Technology*, vol. 7, pp. 6756–6759, 4 2018. DOI: 10.14419/ijet.v7i4.22926. [Online]. Available: www.emoncms.org..

[29] R. A. Abdelouahid, O. Debauche, and A. Marzak, "Internet of things: A new interoperable iot platform. application to a smart building," vol. 191, Elsevier B.V., 2021, pp. 511–517. DOI: 10.1016/j.procs.2021.07.066.

[30] S. Saxena, S. Jain, D. Arora, and P. Sharma, "Implications of mqtt connectivity protocol for iot based device automation using home assistant and openhab," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2019, pp. 475–480.

[31] D. Kovac, J. Hosek, P. Masek, and M. Stusek, "Keeping eyes on your home: Open-source network monitoring center for mobile devices," in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, 2015, pp. 612–616. DOI: 10.1109/TSP.2015.7296336.

[32] R. Kumar, F. Khan, S. Kadry, and S. Rho, "A survey on blockchain for industrial internet of things," *Alexandria Engineering Journal*, vol. 61, pp. 6001–6022, 8 Aug. 2022, ISSN: 11100168. DOI: 10.1016/j.aej.2021.11.023.

[33] S. Zhao, S. Zhu, Z. Wu, and B. Jaing, "Cooperative energy dispatch of smart building cluster based on smart contracts," *International Journal of Electrical Power Energy Systems*, vol. 138, p. 107 896, 2022, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2021.107896. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061521011091.

[34] A. A. Hilal, M. Badra, and A. Tubaishat, "Building smart contracts for covid19 pandemic over the blockchain emerging technologies," *Procedia Computer Science*, vol. 198, pp. 323–328, 2022, ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.12.248.

[35] G. Iredale, "Top 5 programming languages to build smart contracts," 2021. [Online]. Available: https://101blockchains.com/smart-contract-programming-languages/.

[36] E. Sunday, "Top 5 smart contract programming languages for blockchain," 2021. [Online]. Available: https://blog.logrocket.com/smart-contract-programming-languages/.

[37] D. Asset, "Introduction to canton," 2022. [Online]. Available: https://docs.daml.com/canton/about.html.

[38] K.-L. Wright, M. Martinez, U. Chadha, and B. Krishnamachari, "Smartedge: A smart contract for edge computing," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1685–1690. DOI: 10.1109/Cybermatics_2018.2018.00281.

[39] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 108–113. DOI: 10.1109/IVS.2018.8500488.

[40] K. Biswas and V. Muthukkumarasamy, "Securing smart cities using blockchain technology," IEEE, Dec. 2016, pp. 1392–1393, ISBN: 978-1-5090-4297-5. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0198.

[41] Z. Su, Y. Wang, Q. Xu, M. Fei, Y.-C. Tian, and N. Zhang, "A secure charging scheme for electric vehicles with smart communities in energy blockchain," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4601–4613, 2019. DOI: 10.1109/JIOT.2018.2869297.

[42] I. Kounelis, G. Steri, R. Giuliani, D. Geneiatakis, R. Neisse, and I. Nai-Fovino, "Fostering consumers' energy market through smart contracts," in *2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE)*, 2017, pp. 1–6. DOI: 10.1109/ES2DE.2017.8015343.

[43] S. Cheng, B. Zeng, and Y. Z. Huang, "Research on application model of blockchain technology in distributed electricity market," *IOP Conference Series: Earth and Environmental Science*, vol. 93, no. 1, p. 012 065, Nov. 2017. DOI: 10.1088/1755-1315/93/1/012065. [Online]. Available: https://doi.org/10.1088/1755-1315/93/1/012065.

[44] T. L. N. Dang and M. S. Nguyen, “An approach to data privacy in smart home using blockchain technology,” in *2018 International Conference on Advanced Computing and Applications (ACOMP)*, 2018, pp. 58–64. DOI: 10.1109/ACOMP.2018.00017.

APPENDIX A

Synchronization Setup

The NTP protocol was used to synchronise the virtual machines that make up this system. The VM of the Domain Controller will be the NTP Server while the Smart Building's and Third Party's VM will be NTP Clients. The NTP Server configuration is demonstrated below.

```
sudo apt-get install ntp
```

For switching to an NTP server pool closest to your location, it is necessary to edit the ntp.conf file with the chosen pools. The following command allows to edit the ntp.conf file.

```
sudo nano /etc/ntp.conf

Chosen pools:
server 0.pt.pool.ntp.org
server 1.pt.pool.ntp.org
server 2.pt.pool.ntp.org
server 3.pt.pool.ntp.org
```

Then it is required to restart the NTP service to apply the new settings.

```
sudo service ntp restart
```

The last step is to configure the firewall so that clients can access the NTP server.

```
sudo ufw allow from any to any port 123 proto udp
```

In the configuration of the NTP clients, it is necessary to install ntpdate, with the command below.

```
sudo apt-get install ntpdate
```

Then specify the IP and hostname of the NTP server in the host file with the following command.

```
sudo nano /etc/hosts

IP:               Hostname:
192.138.173.138   NTP-server-host
```

Next disable the systemd timesyncd service on the client.

```
sudo timedatectl set-ntp off
```

After that install NTP with the command below.

```
sudo apt-get install ntp
```

Next configure the /etc/ntp.conf file to add your NTP server as the new time server.

```
sudo nano /etc/ntp.conf

Add the line:
server NTP-server-host prefer iburst
```

Finally it is required to restart the NTP service to apply the new settings.

```
sudo service ntp restart
```

APPENDIX B

Daml

B.1. Installation

The following commands are required to install VS Code. The VSCode .deb package is first downloaded from the official website. Next, the editor is installed.

```
cd Downloads
sudo apt install ./code_amd64.deb
```

The command below selects the version of the Daml SDK that should be installed. In this dissertation it was installed the version 2.1.1.

```
curl -sSL https://get.daml.com/ | sh /dev/stdin ${SDK_VERSION}
```

B.2. Smart Contract's Code

The code developed for the smart contract can be found in the following link, in DAML CODE section: https://github.com/bclse/dissertation.

APPENDIX C

PostgreSQL Installation

The first command installs PostgreSQL and its dependencies. Once PostgreSQL has been installed, the second command starts the PostgreSQL service.

```
sudo apt install postgresql -y

systemctl start postgresql
```

The first command opens a PostgreSQL work interface for creating a database and user. The following commands create a database, a user, and a password. The final command gives the user full access to the database.

```
sudo su - postgres

CREATE USER user_example WITH PASSWORD 'password';

CREATE DATABASE db_example;

GRANT ALL PRIVILEGES ON DATABASE db_example to user_example;
```

APPENDIX D

IntelliJ Installation

The commands required to install IntelliJ IDEA are shown below.

Install required dependencies.

```
sudo apt install vim apt-transport-https curl wget software-properties-common
```

Install IntelliJ IDEA.

```
sudo apt install intellij-idea-community -y
```

APPENDIX E

SCIAPP Code

The code developed is presented in the following link, in SCIAPP section: https://github.com/b-clse/dissertation.

The following libraries were used to create the code:

```
<dependency>
    <groupId>org.eclipse.paho</groupId>
    <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
    <version>1.1.0</version>
</dependency>
```

```
<dependency>
   <groupId>com.daml.ledger</groupId>
   <artifactId>bindings-rxjava</artifactId>
   <version>100.13.56-snapshot.20200331.3729.0.b43b8d86</version>
</dependency>
```

APPENDIX F

OpenHAB Installation

The two commands required to install JAVA are shown below, where the first command installs the default Java Runtime Environment (JRE 11) and the second command installs the Java Development Kit (JDK 11).

```
sudo apt install default-jre
sudo apt install default-jdk
```

Regarding OpenHAB, this program can only be installed when JAVA has been set up. First, it is added the repository key.

```
curl -fsSL "https://openhab.jfrog.io/artifactory/api/gpg/key/public" | gpg --dearmor >
    openhab.gpg
sudo mkdir /usr/share/keyrings
sudo mv openhab.gpg /usr/share/keyrings
sudo chmod u=rw,g=r,o=r /usr/share/keyrings/openhab.gpg
```

Next, the HTTPS transport for APT is added.

```
sudo apt-get install apt-transport-https
```

Then it is added the repository.

```
echo 'deb [signed-by=/usr/share/keyrings/openhab.gpg] https://openhab.jfrog.io/artifactory/
    openhab-linuxpkg stable main' | sudo tee /etc/apt/sources.list.d/openhab.list
```

Finally the OpenHAB distribution package version 3.2.0 is installed.

```
sudo apt-get install openhab=3.2.0
```

APPENDIX G

Basic UI Code

The UI code is divided into three files: demo.sitemap, demo.rules, and demo.items. Both Bob and Alice have these three files. The demo.sitemap file contains what is intended to be displayed on the UI. The items that can connect to real things, in this case the API, are contained in the demo.items file. The demo.rules file contains predefined rules. The code developed is presented in the following link, in OpenHAB section: https://github.com/bclse/dissertation

The script shown below, takes two arguments, which are the exact time when the switch was pressed and this time in epochtime and writes them in a txt file. The script executed in the rules is the same, it simply changes the name of the txt file to which it is written.

```
#!/bin/bash
echo "Generated timestamp!"
echo $1 $2 >> /home/kali/Desktop/timestamp.txt
```

APPENDIX H

Mosquitto Setup

The commands for installing the mosquitto broker are shown below.

```
sudo apt -y install mosquitto
```

To add authentication to the broker it is first necessary to create a file. This file is called password.conf and has the format shown below.

```
allow_anonymous false
password_file /home/bob/mos/passwords.txt
```

The following commands must be entered in order to create the password file. The configuration file that was previously created is the first argument. The flag -c indicates that an existing file will be overwritten. The password file output is the second argument and the authentication username is the third argument. After running the command, a password must be assigned to the username. Then, after entering the password, the username and the encrypted password are both contained in the passwords.txt file as shown bellow.

```
password.conf -c passwords.txt mqtt
Password: mqtt
Reenter password: mqtt

cat passwords.txt
mqtt:$7$101$hiP1+p7hFtOgeCcm$1A9n7Nt1RHS77Yivc+
    cxIWFRAdtgNUCbTb3jpqJigiAMd6mAii2rCbUPTeWJxF5xztvPy/X5+W1YrlAO1xrtXw==
```

APPENDIX I

Article

Enabling Distributed Control of Buildings with Smart Contracts

Bernardo Chastre Lopes
Instituto de Telecomunicações
Instituto Universitário de Lisboa (ISCTE-IUL)
Lisboa, Portugal
bernardo_chastre@iscte-iul.pt

Rui Neto Marinheiro
Instituto de Telecomunicações
Instituto Universitário de Lisboa (ISCTE-IUL)
Lisboa, Portugal
rui.marinheiro@iscte-iul.pt

***Abstract*—There are some interesting, centralized solutions for managing smart buildings, whether open-source or not. However, there is a need for some of these functions to be decentralized. Decentralization calls for consideration of security and trust standards that enable a coordinated approach. This paper aims to fill the knowledge and research gaps that still exist in this field. As such, this paper intends to integrate smart contracts with an IoT platform for distributed control in smart buildings. The proposed integration aims the ability to grant control of functions or monitoring of data, in a given smart building, to an external entity that can remotely manage services. To achieve the goal of this paper, an integration was proposed with open-source technologies. Two of these technologies are Daml and OpenHAB, and the created SCIApp application enables their communication. Functional tests, confirm that it is possible to achieve the proposed integration. Response time in the order of seconds was obtained, with an average value of 4061ms. Performance tests allowed to verify response time for different loads. Results confirm that response time remains constant when new contracts are created every 20s. For the remaining frequencies, the response time increases. This paper leads to the conclusion that it is feasible to integrate smart contracts with IoT to control and manage functions of intelligent buildings. By analyzing the tests conducted on the developed system, it was observed that the control is possible for operations that do not require a real-time response time.**

***Index Terms*—smart contracts, internet of things, home automation.**

Work supported by Fundação para a Ciência e Tecnologia (FCT) of Portugal within the project UIDB/50008/2020