UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO ACADÊMICO EM SISTEMAS E COMPUTAÇÃO

Orthus: A Blockchain Platform for Smart Cities

Stefano Momo Loss

Natal-RN, Brasil
2020

Stefano Momo Loss

Orthus: A Blockchain Platform for Smart Cities

Dissertação de Mestrado apresentado ao Programa de Pós-Graduação em Sistemas e Computação do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Mestre em Sistemas e Computação. Área de Concentração: Engenharia de Software

Orientador: Nélcio Alessandro Azevedo Cacho
Coorientador: Frederico Araújo da Silva Lopes

Natal-RN, Brasil
2020

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Loss, Stefano Momo.

Orthus: a blockchain platform for smart cities / Stefano Momo
Loss. - 2020.
95f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande
do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-
Graduação em Sistemas e Computação. Natal, 2020.

Orientador: Nélcio Alessandro Azevedo Cacho.

Coorientador: Frederico Araújo da Silva Lopes.

1. Computação - Dissertação. 2. Blockchain - Dissertação. 3.
Cidades inteligentes - Dissertação. 4. Escalabilidade -
Dissertação. I. Cacho, Nélcio Alessandro Azevedo. II. Lopes,
Frederico Araújo da Silva. III. Título.

RN/UF/CCET

CDU 004

STEFANO MOMO LOSS

"Orthus: A Blockchain Platform for Smart Cities"

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Presidente: Dr. NÉLIO ALESSANDRO AZEVEDO CACHO
(Orientador - UFRN)

Prof. Dr. Anne Magaly de Paula Canuto
(Coordenadora do Programa)

Banca Examinadora

Examinador Externo: Dr. DANILO CURVELO DE SOUZA
(UFRN)

Examinador Externo: Dr. FREDERICO ARAÚJO DA SILVA LOPES
(Coorientador - UFRN)

Examinadora Interna: Dra. THAIS VASCONCELOS BATISTA
(UFRN)

Examinador Externo: Dr. WILSON DE SOUZA MELO JUNIOR
(INMETRO)

Fevereiro, 2020

I dedicate this work first to God who gave me the strength to complete this journey. I also dedicate it to my mother Sadolange Momo (in memorian) who with very caring and effort helped me to reach another stage of my life.

Dedico primeiramente a Deus que me deu forças para completar esta jornada. Dedico também, a minha mãe Sadolange Momo (in memorian), que com muito carinho, não mediu esforços para que eu chegasse até esta etapa de minha vida.

Acknowledgements

Often saying "thank you" is not enough to thank the people who, at the most difficult times of our lives, reached out and offered us help without expecting anything in return. To those people who somehow contributed directly or indirectly to me writing these words, I leave my most sincere thanks. You were very important for me to get here.

In this way, I thank God first for allowing all of this to come true. I am grateful to Him for giving me perseverance, strength, health and intelligence to overcome all difficulties and get to where I am today.

To my friends and family, especially my brother Gabriel Loss and my aunts Denise Momo and Mariangela Momo for always being supportive and helping me at all times. To my father Mario Loss, for trusting me and supporting me in life's moments.

To the Postgraduate Program in Systems and Computation (PPGSC) and the Department of Informatics and Applied Mathematics (DIMAp) of the Federal University of Rio Grande do Norte (UFRN) for all their support and for providing an enabling environment for the development of my project search.

I also thank my fellow students of PPGSC for exchanging ideas and mutual help. In particular, Adelson Araújo Júnior, Camila de Araujo, Guilherme de Freitas, Douglas Rolim, Larysse da Silva, Savio Rennan Melo, João Marcos do Valle, Arthur Souza and Juliana Oliveira. Together we can move forward and overcome all obstacles.

To my teachers for the high quality of the teaching offered, mainly my supervisor Nélcio Cacho and my co-supervisor Frederico Lopes, for their support, corrections and incentives. Without your help, I would not have finished this master's thesis.

Thanks also to the colleagues at Distributed Systems Lab (LASID) for their many afternoons spent in the lab researching and developing new solutions for Smart Cities.

To the institutions that provided the financial support and encouraged the execution of this project is fundamental for the development of this work: to the Brazilian Ministry of Science, Technology, Innovations and Communications (MCTIC), and; to the Brazilian National Council for Scientific and Technological Development (CNPq).

To infinity and beyond...

Agradecimentos

Frequentemente, dizer "obrigado" não é suficiente para agradecer as pessoas que, nos momentos mais difíceis das nossas vidas, nos estenderam a mão e nos ofereceram ajuda sem esperar nada em troca. Para essas pessoas que, de alguma forma, contribuíram direta ou indiretamente para que eu pudesse escrever estas palavras, deixo meu mais sincero agradecimento. Vocês foram muito importantes para eu chegar até aqui.

Desta forma, agradeço primeiramente a Deus que permitiu que tudo isso se concretizasse. Sou grato a Ele por ter me dado perseverança, força, saúde e inteligência para superar todas as dificuldades e conseguir chegar onde hoje estou.

Aos meus amigos e familiares, especialmente meu irmão Gabriel Loss e a minhas tias Denise Momo e Mariangela Momo por estar sempre me apoiando e me ajudando em todos os momentos. Ao meu Pai Mario Loss, por confiar em mim e estar me apoiando os momentos da vida.

Ao Programa de Pós Graduação em Sistemas e Computação (PPGSC) e ao Departamento de Matemática e Informática Aplicada (DIMAp) da Universidade Federal do Rio Grande do Norte (UFRN) por todo apoio e por proporcionaram um ambiente propício para o desenvolvimento do meu projeto de pesquisa.

Agradeço também, meus colegas discentes do PPGSC pelas trocas de ideias e ajuda mútua. Em particular, Adelson Araújo Júnior, Camila de Araujo, Guilherme de Freitas, Douglas Rolim, Larysse da Silva, Savio Rennan Melo, João Marcos do Valle, Arthur Souza e Juliana Oliveira. Juntos conseguimos avançar e ultrapassar todos os obstáculos.

Aos meus professores pela elevada qualidade do ensino oferecido, em especial meu orientador Nélcio Cacho e meu co-orientador Frederico Lopes, pelo suporte, pelas suas correções e incentivos. Sem essa ajuda, não teria terminado essa dissertação de mestrado.

Também agradeço aos companheiros do Laboratório de Sistemas Distribuídos (LASID), pelas inúmeras tardes passada no laboratório pesquisando e desenvolvendo novas soluções para Cidades Inteligentes.

As instituições que forneceram o apoio financeiro e incentivaram a execução deste projeto sendo fundamentais para o desenvolvimento deste trabalho: ao Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTIC) da República Federativa do Brasil, e; ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Ao infinito e além...

Confidence is contagious and so is lack of confidence.

Vince Lombardi, American football coach (1913-1970)

A confiança é contagiosa e a falta de confiança também.

Vince Lombardi, treinador de futebol americano (1913-1970)

Abstract

Currently, blockchain has been widely used to store decentralised and secure transactions involving cryptocurrency (for instance, Bitcoin and Ethereum solutions). On the other hand, smart city applications are concerned about how data and services can be safely stored and shared. In this regard, this research investigates the use of blockchain and pinpoints a set of essential requirements to meet the needs of blockchain for the context of Smart Cities. Based on that, a platform named Orthus was proposed to support the use of blockchain in Smart City initiatives focused on scalability. This master's thesis demonstrated a case study about how to use the proposed platform in the context of the Natal Smart City Initiative, in Brazil, to handle land registration. Moreover, it also compares this platform with other implementations that use blockchain in different domains. Finally, this research confirms that use of blockchain technology has much to contribute to Smart City solutions once it enables the creation of solutions in distributed networks being able to meet the demand of the entire population.

Keywords: Blockchain. Smart Cities. Scalability.

Resumo

Atualmente, o blockchain tem sido amplamente usado para armazenar transações descentralizadas e seguras envolvendo criptomoeda (por exemplo, soluções Bitcoin e Ethereum). Por outro lado, os aplicativos de Cidades Inteligentes preocupam-se com a forma como os dados e serviços podem ser armazenados e compartilhados com segurança. A este respeito, esta pesquisa investiga o uso da blockchain e identifica um conjunto de requisitos essenciais para atender às necessidades da blockchain no contexto de Cidades Inteligentes. Com base nisso, foi proposta uma plataforma chamada Orthus para apoiar o uso de blockchain em iniciativas de Cidades Inteligentes focadas em escalabilidade. Esta tese de mestrado demonstrou um estudo de caso sobre como usar a plataforma proposta no contexto da Iniciativa Cidade Inteligente de Natal, no Brasil, para lidar com o registro de terras. Além disso, ele também compara essa plataforma com outras implementações que usam blockchain em diferentes domínios. Finalmente, esta pesquisa confirma que o uso da tecnologia blockchain tem muito a contribuir para soluções de Cidades Inteligentes, uma vez que permite a criação de soluções em redes distribuídas, capaz de atender à demanda de toda a população.

Palavras-chave: Blockchain. Cidades Inteligentes. Escalabilidade.

List of Figures

Figure 1 – Block structure and connection in a Blockchain	31
Figure 2 – Blockchain Process	31
Figure 3 – Orthus Logo	57
Figure 4 – Medical scenario integration through Orthus	58
Figure 5 – Orthus components integration	59
Figure 6 – BPMN of creation of new transaction process	61
Figure 7 – BPMN of add a transaction process	62
Figure 8 – BPMN of create and mine a new block process	63
Figure 9 – BPMN of add and mine a block process	64
Figure 10 – Folder Structure	66
Figure 11 – Model Layer	67
Figure 12 – Controller layer of reference platform for Blockchain usage in the Smart Cities	68
Figure 13 – Business Process Model and Notation (BPMN) of Case Study	75
Figure 14 – Case Study Integration Diagram	76
Figure 15 – Case Study Class Diagram	77
Figure 16 – Test environment	81
Figure 17 – Effect of number of users variation in the analysis of transaction execu- tion by platforms upon average response time (milliseconds)	83
Figure 18 – Effect of number of users variation in the analysis of transaction execu- tion by platforms upon analysis throughput (transaction per second)	84

List of Tables

Table 1 – Comparison between Consensus Mechanisms in Decentralized Database vs Permissioned and permissionless Blockchain	34
Table 2 – Systematic Review Summary	47
Table 3 – Selected papers list	47
Table 4 – Requirements Taxonomy	51
Table 5 – Comparison Orthus vs Hyperledger Fabric	82

List of abbreviations and acronyms

API	Application Programming Interface
BPMN	Business Process Model Notation
CQL	Cassandra Query Language
EC	Exclusion Criteria
GE	Generic Enablers
HTTP	Hypertext Transfer Protocol
IC	Inclusion Criteria
ICT	Information and Communication Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MIS	Management Information System
MVC	Model-View-Controller
NGSI	Next Generation Service Interfaces
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
RQ	Research Question
SQL	Structured Query Language
SRRQ	Systematic Review Research Questions
URL	Uniform Resource Locator

Contents

1	Introduction	23
1.1	Problem Statement and Limitation of Current Approaches	24
1.2	General Research Questions and Goals	26
1.3	Main Contributions	26
1.4	Methodology	27
1.5	Thesis Organization	28
2	Background	29
2.1	Distributed Database	29
2.2	Blockchain	30
2.2.1	Blockchain Types	32
2.2.2	Consensus Mechanisms	33
2.3	Smart Cities	34
2.3.1	Smart Cities System Requirements	36
2.4	Reactive Design Patterns	37
2.4.1	Akka	38
2.4.2	Play Framework	41
2.5	Other Technologies for Smart Cities	41
2.5.1	Spark	41
2.5.2	Cassandra	42
2.5.3	Orion Context Broker	42
3	Related Works	45
3.1	Systematic Review	45
3.1.1	Planning	45
3.1.1.1	Systematic Review Research Questions	45
3.1.1.2	Search Strategies	45
3.1.2	Inclusion and Exclusion Criteria	46
3.1.3	Execution Results	47
3.1.4	Requirements Taxonomy	48
3.1.5	Evaluation Results	50
3.1.6	Threats to Validity	54
3.2	Others Blockchain Solutions	54
4	Orthus Platform	57
4.1	Architecture Overview	57
4.2	Requirements Fulfillment	59
4.3	Components Communication Process	60
4.3.1	New Transaction Creation	61

4.3.2	Add a Transaction	61
4.3.3	Create and mine a new Block	62
4.3.4	Add and mine a Block	63
5	An Orthus Implementation	65
5.1	Technologies chosen	65
5.1.1	Model Layer	66
5.1.1.1	Extension Points	67
5.1.2	Controller Layer	68
5.1.3	View Layer	69
5.2	Implementation details	69
5.2.1	Actors Instantiation	70
5.2.2	Orion Subscription	71
5.2.3	Block Hash Method Calculation	72
6	Platform Evaluation	73
6.1	Study Setting	73
6.1.1	Study Hypothesis	73
6.1.2	Target Systems	74
6.1.2.1	Case Study	74
6.1.3	Study Phases	75
6.2	Study Results	81
6.2.1	Average Response Time	82
6.2.2	Throughput	83
6.3	Threats to Validity	84
7	Conclusion	85
7.1	Contribution	85
7.2	Limitations	86
7.3	Threats to Validity	86
7.4	Future Works	87
	Bibliography	89
	APPENDIX A List of Publications	93
A.1	Publications resulted from this work	93

1 Introduction

Money usage predates history since there are historical records of government control of money in the ancient Egyptian economy (2750-2150 BC) (VANTIEGHEM, 2012). Up to the beginning of the 17th century, the monetary system was still controlled by government institutions, when the first central banks were created. Since then, central banks have been monitoring all bank transactions involving paper money. As an alternative form, the digital currency, a type of currency available only in electronic form and is not tangible, was introduced by David Chaum in 1983 (CHAUM, 1983). An interesting concept to create digital currency is cryptocurrency (NARAYANAN et al., 2016). Cryptocurrency is a digital asset that uses cryptography to secure its transactions and use decentralized control as opposed to centralized electronic money and central banking systems.

The world's first utterly decentralized cryptocurrency was the Bitcoin (BRITO; CASTILLO, 2013). Bitcoin definition was created in 2008 by an unknown person using the pseudonym Satoshi Nakamoto (AIXA, 2018). One of the main actors responsible for Bitcoin success is Blockchain technology. The creation of Blockchain is inherent in the creation of Bitcoin.

Blockchain can be defined as a distributed database that registers an ordered list of transactions records that are immutably linked together through a chain, on blocks (AIXA, 2018). Blockchain is a disruptive technology because it digitally creates a trusted entity with the potential to replace transaction certifying and centralizing entities such as banks, governments, notaries. With Bitcoin popularization, several applications began to use Blockchain to store data in a distributed way. According to David Aixa (AIXA, 2018), some of the potential applications of Blockchain are supply chain, digital identity, voting, healthcare and government (AIXA, 2018, p. 37). Some of them are aimed at solving social problems in the most diverse areas, contributing to the construction of Smart Cities.

In the context of this work, Smart Cities infuse "information into its physical infrastructure to improve conveniences, facilitate mobility, add efficiencies, conserve energy, improve the quality of air and water, identify problems and fix them quickly, recover rapidly from disasters, collect data to make better decisions and deploy resources effectively, and share data to enable collaboration across entities and domains" (KANTER; LITOW, 2009). This infusion occurs when using the information and management system to solve the social problem.

Currently, most municipal public services already use some Management and Information System (MIS) to store their data and, in some cases, provide some assistance to the population. However, those systems can be developed by different private companies or public agencies and in most cases, are designed in isolation and without communication standard, which makes difficult the interoperability with other systems. Interoperability

between systems is the ability of a set of systems to share information and services aims to use the information or services exchanged, in a given context (BROWNSWORD et al., 2006).

For example, suppose a Smart City has the following Management Information System:

- Hospital Information System. This system provides, among other functions, information about a patient's health history through the electronic health record, available internally only.
- Medlab. A Medical laboratory system that manages and store information about patient's medical exam. It is used only in a private medical labs network.
- Pharmacy Control. The government system to control the number and type of medicines sold by drugstores to population.

As each of those systems operates in isolation and independently, the patient must take a printed prescription to get medical checkups or buy medicine. This process occurs because there is no interoperation between them. If these systems had a reliable, secure distributed decentralized database to store and share information between these systems, it would be possible, for example, the doctor to prescribe a controlled medicine to a patient directly in this database that is accessed by the pharmacy system, without the need for a printed prescription, and justifies this prescription with a medical exam stored into the electronic health record. Furthermore, as a result of the integration between these systems, the number of frauds would certainly decrease since all operations could be verified and confirmed.

Thus, in this perspective, Blockchain can be used to generate interoperability between systems in addition to sharing data and services in a transparent, secure and reliable way, without centralizing entities, enabling the integration of city services to improve the quality of life of the population (SUN; YAN; ZHANG, 2016).

1.1 Problem Statement and Limitation of Current Approaches

There are many ways to integrate systems for share data between them in decentralized form, most of which are done ad hoc to meet a specific customer need. There are already some ways to integrate system data such as using middleware or frameworks to compose systems. The problem with sharing data between different systems is the reliability of data exchanged. Blockchain can be used to ensure decentralized and secure data sharing with security through transaction storage. However, Blockchain was initially created to store cash-only transactions (BRITO; CASTILLO, 2013).

Therefore, the first problem addressed in this work can be summarized as :

Problem I:

Blockchain, as conceived initially by Bitcoin, is not generic enough to store any data transaction to meet the needs of being used in non-cryptocurrency contexts. For example, it cannot store files, images or other types of media.

Into Blockchain, data are stored as a transaction. A challenge for Blockchain technology used by Bitcoin refers to scalability in terms of the number of transactions processed per second. The current Bitcoin's Blockchain technology processes only seven transactions per second (SWAN, 2015).

Another problem related to latency with Blockchain implementation is the processing time needed to create and verify new transactions and create blocks with them. That process uses consensus algorithms to verify new transactions, as shows in 2.2.2, and some algorithms are more expensive than others.

There are several types of consensus algorithms, depending on the Blockchain type and Blockchain platform implementation. For example, Bitcoin uses a proof-of-work algorithm that by project decision takes about ten minutes to process each block by multiple computers before they can be stored in a Blockchain block (ALVES; CAVALCANTE; BATISTA, 2018). This time is necessary to calculate the computational challenge and avoid forks in the Blockchain. On the other hand, EOS platform¹ uses the proof-of-stake algorithm that takes about half second to process each transaction (ALVES; CAVALCANTE; BATISTA, 2018). Regardless of the consensus algorithm used, all will have some strengths and weaknesses, and it will always have a computational cost involved in processing and time to verify each transaction.

The second problem addressed in this work can be described as follows:

Problem II:

Blockchain has a high computational cost for creating, verifying and storing new transactions.

Even though problems I and II are being resolved, another limitation of Blockchain is related to its use in the context of Smart Cities. Blockchain systems such as Bitcoin and Ethereum, which have been developed using conventional technologies such as sequential processing tools and/or relational databases (Ethereum uses a Merkle Patricia Trie to save its state), cannot handle large amounts of diversified data produced by Smart Cities. Since a Smart City application has particular requirements as shows in 2.3.1. Thus, the Blockchain implementation must use some Big Data technology to store and manipulate a large and varied amount of data at the required velocity.

Finally, the third problem addressed in this work can be stated as:

¹ <https://eos.io/>

Problem III:

Blockchain, as originally conceived, cannot store and process a large amount of data with the required speed, which is required in the context of Smart Cities.

1.2 General Research Questions and Goals

Based on the context and limitation of current approaches, according to elicited in Section 1.1, four general research questions (RQ) were proposed in order to guide this work. They are:

RQ1. How can Blockchain be used in Smart Cities environments? This RQ aims to investigate the use of Blockchain in Smart Cities context, find out in which cases it can be used and how it was used.

RQ2. What are the requirements to use Blockchain in the context of Smart Cities? This RQ aims to investigate what are the requirements for the efficient use of Blockchain for Smart City applications.

RQ3. How to architect a Blockchain platform for Smart Cities context that meets with these requirements? This RQ aims to ensure that the specified platform architecture meets criteria elicited by RQ2 since the platform can be implemented with different Big Data technologies following the reference architecture.

RQ4. Is the proposed platform capable of promoting interoperability between information systems based on a Blockchain approach? Finally, this RQ aims to define how the proposed platform will be validated, what will be the metrics and benchmark to verify their actual effectiveness.

To answer the proposed research questions, the main goal of this thesis is (i) conduct a systematic review to allow a better understanding of how Blockchain could contribute to developing reliable Smart City solutions and your requirements; (ii) propose a reference architecture to use Blockchain in Smart City. The justification for creating a new platform is that competing platforms do not fully meet the requirements found in the systematic review; (iii) develop a case study that uses Blockchain in the context of Smart Cities applications to validate this architecture, and (iv) confirm this case study comparing with other Blockchain approaches.

1.3 Main Contributions

This work comes up with four main contributions, each one summarized in the following.

A systematic review of the use of Blockchain in the context of Smart Cities. The first contribution brought by this work is the systematic review that elicited

a set of requirements needed to use Blockchain in Smart Cities context and discover the leading platforms used in this context.

A Blockchain platform for Smart Cities named Orthus. A platform is proposed for share and storing transactions of Management Information Systems in a distributed and secure way to support the amount of data generated from city devices in the Smart Cities context where these transactions can meet any data type with velocity and veracity. The justification for creating a new platform is that competing platforms do not fully meet the requirements found in the systematic review.

A land registry case study using Orthus. The Orthus platform was used to develop a case study involving land registration in a city to prove its applicability. The transfer of land between people of this city involving three government agencies was simulated.

A comparative test between Orthus and Hyperledger Fabric. The same case study was performed in the same computing environment reasonably to be able to compare the Orthus platform with Hyperledger Fabric. Average response times and transactions per second were compared.

1.4 Methodology

To achieve the proposed objectives, a set of methodological procedures were adopted. The first activity was a systematic review, performed as described in Chapter 3, where the main requirements of Orthus platform were elicited.

After the systematic review, the architecture for the platform was discussed and defined. To implement the architecture design, some technologies were chosen following the Reactive Manifesto.² For this, some techniques have been studied to meet the requirements, and the Reactive Manifesto and the technologies are chosen were: Akka with Play Framework was selected to be the core of the platform, Cassandra as a database, Orion as context broker, as will be described in Chapter 2.

Once the platform architecture was defined, and the technologies were chosen, the implementation process began. The Orthus platform was implemented using the Java language according to the Model-View-Control pattern. A case study involving land registry in Smart Cities context was chosen to validate the platform, with the result of this test it was possible to compare with other solutions that use Blockchain as a secure and decentralized database.

² <https://www.reactivemanifesto.org>

1.5 Thesis Organization

The remainder of this master's thesis is organized as follows. Chapter 2 presents the background with essential concepts applied during the thesis development. Chapter 3 presents related works through a systematic review to elicit the necessary platform requirements in Smart Cities context, and present and compare other Blockchain platforms used in this context. Chapter 4 introduces and describe the Orthus architecture, a Blockchain platform for Smart Cities.

Chapter 5 justifies the choices of technologies to meet the platform architecture and presents details of the platform's implementation. Chapter 6 presents a case study involving a land registry in Smart Cities context implemented using Orthus and in this chapter also performs an Orthus validation comparing it with another Blockchain approach. Chapter 7 revisits the achieved contributions and presents perspectives for future work. Appendix A contains a list of publications that resulted from this work.

2 Background

This chapter presents some essential concepts and techniques to provide a better understanding of this work. Section 2.1 define a distributed database and shows its strong aspects and weaknesses. Section 2.2 gives an overview of Blockchain, the core of this research, presents and compares Blockchain types, consensus mechanisms often found in the literature. Section 2.3 presents some concepts on Smart Cities as well as their main requirements for systems in this context. Section 2.4 introduces the Reactive Design Pattern created from the Reactive Manifest and introduces some technologies that follow this pattern. Finally, Section 2.5 presents the technologies Spark, Cassandra and Orion Context Broker responsible for analyzing, store and disclose data, respectively, that comply with the Smart Cities requirements.

2.1 Distributed Database

Özsu and Valduriez (2011) define a distributed database as "a collection of multiple, logically interrelated databases distributed over a computer network." A distributed database system is the join of the distributed database with the distributed database management system that "permits the management of the distributed database and makes the distribution transparent to the users" (ÖZSU; VALDURIEZ, 2011).

Information in a distributed database is replicated or divided in different places, and it is necessary to ensure data consistency in different areas of the network. In relational databases, where data is strongly related in tables, to maintain consistency when a record is edited it is necessary to lock all related files, which makes replication difficult because it is required to lock multiple tables. To facilitate this process, there are already several tools such as master-slave algorithms where it has a master node that controls replication and, only in it, is possible to make changes in the data, and several slaves nodes where the data is read. If the master node goes down, a slave node becomes the new master node to keep the central management.

To try to make the replication process even more accessible, not only related databases have emerged that synchronization may be momentary as the data is not strongly related, so it is not necessary to lock more than one structure.

However, in both distributed database approaches, they are most often not concerned with the use of this data by more than one system. If so, they are not concerned about the veracity of the data generated by the systems that have access to the distributed database, as these systems are reliable. On the other hand, if more than one system shares the same distributed database and one of them forges false data, the other systems will not verify the truth of the data and will use it as if it were true. It was precisely to fill

this gap that Blockchain emerged.

2.2 Blockchain

The creation of Blockchain is inherent in the creation of Bitcoin. The Bitcoin definition was created in 2008 with the publication of the article "Bitcoin: A Peer-to-Peer Electronic Cash System" (NAKAMOTO, 2008), by an unknown person using the pseudonym Satoshi Nakamoto (AIXA, 2018).

The main difference between the distributed database system and Blockchain is related to your purpose. In contrast, the decentralized database is concerned to "provide data storage and access services to business systems and operational-support, mainly for business products and development projects, with the data being stored with a focus on supporting analysis and retrieval" (LI, 2018).

On the other hand, Blockchain is concerned to "build a new ecosystem of digitized data assets and automated trust services. The global Blockchain updates its state autonomously, and data is traceable to its source" (LI, 2018). Besides, Blockchain is decentralized management and uses consensus mechanisms to verify data veracity while the decentralized database is centralized targeting and does not care about the data source.

Thus, Blockchain is a technology that enables distributed transactions and data sharing between unreliable participants without the need for a centralizing authority for validation. It introduces the new consensus paradigm, where the network node set agrees with the order in which transaction blocks are aggregated into the Blockchain, providing a reliable, scalable, unchanging, and secure distributed environment for the realization and storage of all transactions. Blockchain should only be used when you do not have a trusted third party or when the cost of having a trusted party is more expensive than using Blockchain.

The purpose of Blockchain technology is to eliminate a single centralized entity to validate the addition, editing or deletion of data transactions, for example. For this to occur, more than half of network participants must agree on transaction data to be considered valid and be entered into the Blockchain database, so that the operation is denied without consensus. The definition of the correct term for a transaction may vary by the context of the Blockchain system, where each defines the validation protocol.

When new data need to be inserted in the database, an original transaction must be created to contain this data. Transactions are validated and grouped into blocks. This block is added to the Blockchain database as a permanent record. Once in the chain, transactions stored in one block cannot be changed or deleted without interfering with other blocks in the chain, which makes the Blockchain immutable.

The Blockchain structure is made up of infinite growing blocks, as shown in Figure 1, which contain stored data linked to each other where are created from the first block

(genesis block). The following blocks are formed by (i) a generated hash to represent a unique identifier of the data within that block; (ii) the hash of the previous block; (iii) a time stamp; and (iv) block data that contains some transactions.

Figure 1 – Block structure and connection in a Blockchain

Source: Alves, Cavalcante and Batista (2018)

The generation of the hash consists of encrypting and compressing the block data into a fixed-size string of characters, disregarding the amount of data inside the block (SWAN, 2015). The hash value can be recalculated to confirm that data inside the block were not altered, but the reverse process of generating the block's data from the hash is impossible to perform.

Figure 2 – Blockchain Process

Source: PwC's Digital Pulse

<https://blockchainhub.net/blog/infographics/look-blockchain-technology/>

Figure 2 shows the steps for securely creating a transaction using Blockchain. The process starts when someone requests a transaction; the requested transaction is broadcast

to all nodes. In turn, the network of nodes checks if the transaction is valid and verify the user's status using a consensus algorithm. After the transaction is validated, the transaction is combined with other transactions to create a new block. After that, the block is inserted into the Blockchain in a permanent and unalterable way. Finally, after this process, the transaction is finalized.

Blockchain technology uses some consensus algorithms to randomize block processing across the network of nodes as well as to avoid service abuses. Some types of consensus algorithms will be detailed and compared in Subsection 2.2.2. For example, Bitcoin and initially, Ethereum uses the proof-of-work implementation scheme based on hash-cash. Hash-cash technology (BACK et al., 2002) was initially proposed as a mechanism to throttle systematic abuse of unmetered Internet resources such as e-mail and anonymous re-mailers. It works by creating a proof-of-work (a cost function that generates a token) to be computed by users to establish a connection with a server. The server will check the value of the token using an evaluation function, and it proceeds with the connection only if the token has the required value.

As blocks are chained in the database, changing data within a block requires redoing the proofs-of-work in all succeeding blocks (NAKAMOTO, 2008). Nevertheless, if a node can modify blocks in the chain or it starts accepting invalid transactions, its database would not match the others, and it would not be accepted as correct record thanks to the distributed consensus mechanism. In this way, Blockchain ensures the immutability, reliability and security of the information stored within the database.

In addition to these essential requirements, Blockchain technology must meet other characteristics to be used in the context of Smart Cities, since initially the Blockchain was not designed to achieve it. To build a Blockchain platform for Smart Cities, firstly, it is necessary to understand their concepts and requirements.

2.2.1 Blockchain Types

The creation, access, and validation of a transaction stored in the Blockchain database depend on the type of permission present on the Blockchain network. It could be that anyone can have a node with a database copy, but no new transactions will be added. Alternatively, where only authorized persons have access to the stored data, but anyone can add new transactions. Moreover, besides, as with Bitcoin, where anyone can participate, and all transactions are publicly available. Conceptually, the authors Mattila (2016) define two Blockchain types of permissions:

1. **Public or Permissionless:** Anyone can create new transactions and have access to the ledger and can participate in consensus mechanisms. The consensus mechanism is required, as described in 2.2.2.

2. **Private or Permissioned:** Restriction on access and who can transact. Only the owner determines who can participate in the consensus mechanism. In this case, as there is a certain level of trust between the systems/nodes that is part of the platform, but still can not have confidence in the data generated by them. Usually uses Byzantines Fault Tolerance as consensus mechanisms as described in 2.2.2.

2.2.2 Consensus Mechanisms

Consensus mechanisms are used in Blockchain to ensure that a particular network node is honest since only by using digital certificates, it is not possible to have this guarantee. Singhal, Dhameja and Panda (2018) defines "a digital certificate is a public key, bundled with the device ID and the certificate expiration date". It allows verify and confirm that the public key belongs to the legitimate node of the network, not someone pretending to be it. However, the digital certificate does not guarantee that this node is trusted.

According to Singhal, Dhameja and Panda (2018), consensus mechanisms originate from game theory, where it rewards honest players and punishes rogue players who engage in fraudulent activity. However, in public Blockchains, such as Bitcoin and Ethereum, it is difficult to punish rogue nodes because new nodes can be recreated without punishment. The best solution is to reward the nodes that validate transactions honestly. The answer is to create some algorithm to find a trusted node as it is not possible to determine in advance whether a randomly selected node is a malicious or honest node.

The goal of consensus is also to ensure that the network is robust enough to sustain various types of attacks. If nodes need permission to participate in the Blockchain network (private Blockchain), it can fit the Byzantine fault-tolerant consensus model. Enumerated below are the most commonly used algorithms in public and private Blockchain:

1. **Proof-of-work** is a process where new transactions are transmitted to the network, miners must find proof-of-work, a computational challenge to generate hash-based on transactions that will be part of a new block, to receive some reward in return. This reward is inserted into the block as if it were a new transaction generating more coins. In this approach, you need to ensure that the entire network has delays sufficiently in maintaining synchronization between mining events, slowing down the mining of blocks (ALVES; CAVALCANTE; BATISTA, 2018).
2. **Proof-of-stake.** Unlike what happens in proof-of-work, in proof-of-stake, there are no due mining rewards, there are only rewards transaction fees for validators. Validators need to sign up to be able to participate in transaction validation, which depends on your amount of cryptocurrencies. The probability is proportional to your bet, the higher the number of cryptocurrencies, the greater the chance of validating a new transaction block. The decision of who will create the new transaction block is

decided and varies based on the PoS algorithm used. Because this algorithm uses the amount wagered by each validator to determine the winner, it becomes deterministic and works faster than a proof-of-work. (SINGHAL; DHAMEJA; PANDA, 2018, p. 253)

3. **Byzantines Fault Tolerance.** According to Alves, Cavalcante and Batista (2018), it is a consensus mechanism "which allows for large-scale participation in consensus through proxy voting while reducing the time to process and validate blocks." However, user nodes (for example, companies, bank users, notaries, city halls) must be restricted and allowed to participate in this voting process, that is what exactly is restricted in the private type Blockchain (described in 2.2.1).

Table 1 summarises and compares decentralized Blockchain database with and without permission on the following criteria, namely: i) geographically distributed if data is allocated to more than one network location; ii) heterogeneous concurrent access, whether data can be accessed in more than one location and by more than one user or system concurrently; iii) internal communication protocol, names the types of internal communication protocol; iv) trusted parties if it has a centralized entity capable of verifying the stored data; v) consensus mechanisms, names the type of consensus mechanism, and vi) data access permission if permission is required to access stored data.

Table 1 – Comparison between Consensus Mechanisms in Decentralized Database vs Permissioned and permissionless Blockchain

	Geographically Distributed	Heterogeneous Concurrent Access	Internal Communication Protocol	Trusted Parties	Consensus Algorithm	Data Access Permission
Distributed Database	Yes	No	Replication Protocol	Yes	Not required	Not required
Permissionless Blockchain	Yes	Yes	Gossip Protocol	No	Proof-of-work Proof-of-stake	Not required
Permissioned Blockchain	Yes	Yes	Gossip Protocol	No	Byzantines Fault Tolerance	Required

2.3 Smart Cities

For Kanter and Litow (2009), a Smart City integrates its physical infrastructure with information and communication technology (ICT) to improve people's quality of life, urban mobility and government management, saving energy, improving air and water quality, identifying urban problems, collecting data to make better decisions and fix them quickly, among others.

However, merely integrating intelligence into every public service in a city (transportation, energy, education, health care, buildings, physical infrastructure, food, water,

public safety) alone is not enough to have a Smart City. A Smart City must be seen as a complex organism, as a network of integrated and interoperable services. In a Smart City, attention must be given to connections as well, not just parts. The improvement in cities stems from the proper integration of its constituent parts, making it a Smart City focused on improving the quality of life of the population.

Thus, ICTs can be the integrative link between the services of a Smart City. They can provide higher intelligence in city management (BLAIR et al., 2015) generating economy, efficiency and improving the quality of life. The intelligence vision of cities comes from the integration between society and the digital city.

While ICTs are required in Smart Cities, Blair et al. (2015) report that the presence of digital components does not guarantee intelligence for a city. A digital city that uses communication technologies and promotes broad access to tools, content and management systems to meet the needs of the public authorities and their servers, their citizens and organizations, not necessarily it is smart.

At the same time, there is no unique way to have a Smart City, because the problems of cities can be varied and the forms of the solution too. For example, traffic can be a big problem for a crowded city and not so much for other cities. Thus, depending on the investment, the use of ICTs may or may not be present in some cities.

Gama, Alvaro and Peixoto (2012) create a model of technological maturity to grade cities based on the integration of varied (conceptual and physical) devices. There are five maturity levels defined by them:

0 - Chaotic: The start phase of most cities, in which cities do not use ICTs to assist in their management. Residents may use social networks, but there is no use of systems aimed at improving living conditions or services.

1 - Initial: phase of planning and modelling of information systems that will assist in a given domain, as well as the identification of existing systems that could potentially be integrated into the Smart City solution. There may be sensor data capture.

2 - Managed: data collected (traffic data, power consumption, etc.) and accessible through information systems.

3 - Integrated: Smart City with integrated systems that can use the Cloud Computing model, being available as services for both citizens and third-party applications. It presents the intensive use of ubiquitous and autonomic computation. The government can function as a Smart City facilitator and foster of a service ecosystem.

4 - Optimized: A more efficient city seeking to innovate and pioneer ICT solutions. Use of data obtained from the various domains of the city to support decision making for both the population and the rulers.

Looking at the levels described by the authors, it can be stated that levels 0, 1 and 2 of the maturity model are still related to a digital city, whereas levels 3 and 4 already characterize a city as smart. In this case, what differentiates a digital city from a Smart

City is that in the former ICTs exist to collect and store data, while in the latter the use of ICTs aims to improve societal problems.

At the same time, it is necessary to have a digital city for a Smart City to exist, and achieving a higher level of quality of life, governance and sustainable development is no longer easy to implement without the help of an essential element such as ICTs. This way, Blockchain technology can be used to reliably facilitate the process of integrating Smart Cities systems without a central entity. Even though various solutions help improve the lives of the population, some requirements are common to all computing systems in this context.

2.3.1 Smart Cities System Requirements

The recurring requirements for Smart City systems are related to handling large amounts of data. Since real-life objects become part of the Internet where their information is stored, giving rise to the term internet of things (IoT). Coetzee and Eksteen (2011) states that these objects must be able to access their position and status through the network using a unique identifier.

There are a large number of devices used to collect city data that forces platforms for Smart Cities to use IoT technologies. Data collected from these devices must be securely transmitted over computer networks to be grouped and processed to provide more advanced Smart City services. To handle this massive amount of data, using tools to store and manipulate large data sets is required. This set of techniques is known as Big Data tools that the main requirements are commonly referred to as the Big Data four Vs (PATGIRI; AHMED, 2016):

- **Volume.** Data volume refers to the size of data sets that need to be analyzed and processed, being larger than terabytes and petabytes in some cases. Requiring different processing technologies and different from traditional storage and processing technologies. In other words, this means that data sets in Big Data are too large to be processed with a normal desktop or laptop processor.
- **Velocity.** Data velocity is related to the speed with which data is generated. High-speed data is generated at such a rate that it requires distinct (distributed) processing techniques.
- **Variety** Variety of data makes Big Data have a wide variety of sources and types that often require distinct processing resources and specialized algorithms.
- **Veracity** Veracity is related to the quality of the data being analyzed. Highly accurate data has many records that are valuable to analyze, and that contribute significantly to overall results. Low truth data, on the other hand, contains a high

percentage of meaningless data. What is not valuable in these datasets is called noise.

Data that is high volume, high velocity and great variety must be used advanced tools to be storage, processing, and analysis. Because of these characteristics of the data, the domain has been labelled Big Data. In this sense, this is very similar to what is expected from a platform for Smart Cities. One way to ensure that all these requirements are met is to use Reactive Design Pattern to develop a Blockchain platform.

2.4 Reactive Design Patterns

Reactive Design Patterns is a set of design principles for creating cohesive systems. It is a way of thinking about systems architecture and design in a distributed environment where implementation techniques, tools, and design patterns (KUHN; HANAFEE; ALLEN, 2017).

A Reactive System is based on an architectural style consisting of individual services, each with a specific purpose, that perform tasks that when combined, produce an expected result. The main driver of using this technology is the notion of responsiveness, and if the customer does not get value promptly, he will look for the answer elsewhere.

Some issues need to be resolved to have a responsive system: being resilient in the event of failure; be responsive under load, defined as elasticity. The Reactive Manifest describes that the system needs to be message-driven to meet these requirements (KUHN; HANAFEE; ALLEN, 2017).

The main benefits of Reactive Programming are the increased utilization of computational resources on multicore and multi-CPU hardware. A secondary benefit is developer productivity, as traditional programming paradigms have struggled to provide an easy-to-maintain and straightforward approach to dealing with asynchronous and non-blocking in-out. Reactive programming usually eliminates the need for explicit coordination between active components.

Reactive programming excels in component design and workflow composition because making the most of asynchronous execution, including backpressure, it is crucial to avoid overuse or very unlimited resource consumption. The Reactive Manifesto considers a reactive system to be:

- **Responsive** means that a system must always respond on time while maintaining low latency;
- **Resilient** means that a system must be responsive even when parts start to fail, keeping downtime to a minimum;

- **Elastic** means that if the workload increases the system must scale to remain responsive, but also with the decreasing workload it must scale down, using resources efficiently;
- **Message oriented**, sending messages to different parts of the system, which, according to Ledbrook states, is a defining characteristic of a Reactive system.

Some technologies follow this pattern that can be used. In the next subsections will present and explains the techniques used in the Orthus implementation to ensure compliance with these requirements.

2.4.1 Akka

Akka¹ was developed in 2009 by Jonas Bonér, inspired by Erlang's actor model. Its first public release, Akka 0.5, was announced on January 2010. This solution is currently maintained by Typesafe Inc., the same company that supports Scala and the Play Framework. Recently, Akka became part of the standard Scala API, replacing the original actor model of this language, and despite being written in Scala, it has an official API for Java as well.

By the team's definition, Akka is a "toolkit and runtime for building highly concurrent, distributed, and fault-tolerant JVM applications", it uses actors model to build systems, following the Reactive Manifest, that scale up, using the resources of a server more efficiently. Thus, applications created using Akka are born ready to operate in a distributed manner, as this feature is at the core of the Orthus platform.

To design and build highly concurrent, distributed, fault-tolerant, and event-oriented applications in the Java and Scala languages, Akka provides the following tools:

- **Actors:** An actor is an independent computational entity that performs specific tasks to respond to messages. There is no shared state between actors. To respond to messages, the actor can create new actors, update their state, or execute computational logic. Each actor has a mailbox in which incoming messages are queued and dequeued as they are processed. The arrival order of messages is determined at run time, but queuing is done considering the send order;
- **Supervisors:** To manage specialized actors, Akka uses the idea of supervisors (actors) who coordinate their subordinates' actors lifecycle. If a given actor cannot handle a proper message or behaves abnormally, the supervisor defines the type of recovery should be performed by the actor depending on the type of exception thrown.

¹ <https://akka.io>

- Futures: To perform callbacks and concurrent operations, Akka works with the Future framework, which needs an ExecutionContext, similar to Java Executor. When defining an actor system, you can access an ExecutionContext by default;
- Dispatcher: To define the executor where concurrent routines will be executed, Akka works with the notion of dispatchers that defines how a set of actors will be distributed in a thread pool;
- Routers: In Akka, it is possible to create more than one entity of a certain type of actor, so the routers route the messages to the recipients. The child actors who receive the messages are called routes.

To create an Akka actor in java, you need to extend from `AbstractActor` and override the `createReceive()` method. An example can be seen in Listing 2.1. This method checks the type of message received, where in line 8 of the Listing 2.1, it receives a message of type `String` and responds to the actor who sent it using method `getSender.tell()` where the response is passed by parameter, in this case, comes from the `processMessage()` method. In line 2 of the Listing 2.1, the `props()` method is defined and is used by the `SupervisorActor` to be instantiated, as can be seen in the Listing 2.2.

```
1 public class MyActor extends AbstractActor {
2     public static Props props() {
3         return Props.create(MyActor.class);
4     }
5     @Override
6     public Receive createReceive() {
7         return receiveBuilder()
8             .match(String.class, msg -> {
9                 getSender().tell(processMessage(msg), getSelf());
10            }).build();
11     }
12     private int processMessage(String msg) throws
13         ArithmeticException{
14         /* Some computational logic */
15     }
```

Listing 2.1 – MyActor Example Code

To create an Akka Supervisor in Java, which is also an actor, you need to extend from `AbstractActor` and override the `supervisorStrategy()` method that defines what to do with child actors for each exception type that can be thrown by them. An example can be seen in Listing 2.2, were into the `props()` method, see line 2, is created a `MyActor` actor with `RoundRobin` router with a pool of 50, this number can be changed as needed.

```
1 public class SupervisorActor extends AbstractActor {
2     public static Props props() {
3         ActorRef myActor = getContext().actorOf(MyActor.getProps().
4             withRouter(new RoundRobinPool(50)));
```

```

4  return Props.create(ControllerActor.class);
5  }
6  @Override
7  public SupervisorStrategy supervisorStrategy() {
8      return new OneForOneStrategy(10, Duration.create("10_
          seconds"), DeciderBuilder
9          .match(RuntimeException.class, ex -> stop())
10         .match(NullPointerException.class, ex -> restart())
11         .match(ArithmeticException.class, ex -> resume())
12         .build()
13     );
14 }
15 @Override
16 public Receive createReceive() {
17     return receiveBuilder()
18         .match(String.class, msg -> {
19             myActor.tell(msg, getSelf());
20         })
21         .match(Integer.class, result -> {
22             System.out.println(result);
23         }).build();
24 }
25 }

```

Listing 2.2 – SupervisorActor Example Code

Already inline-five is defined the method *supervisorStrategy()* being of type *OneForOneStrategy* passing as parameter the maximum number of retries, the interval time between retries and for each type of exception what to do. The method *supervisorStrategy()* can be: (i) *OneForOneStrategy*, wherein the case of failure during processing, only the actor who is performing is affected, causing a retry to be completed. And (ii) *AllForOneStrategy*, where, in case of failure, all actors involved in request processing are affected, redoing the entire execution process.

To execute the actors, it is necessary, within a primary method, to create an *ActorSystem* object and from there create the *SupervisorActor* by calling the *props()* method, as seen in Listing 2.3.

```

1 public static void main(String[] args) {
2     ActorSystem _system = ActorSystem.create("SystemActor");
3     ActorRef supervisorActor = _system.actorOf(SupervisorActor.
        props(), "supervisorActor");
4     supervisorActor.tell("Lorem ipsum dolor sit amet", _system);
5 }

```

Listing 2.3 – Main Method Example Code

The *main()* send the message "Lorem ipsum dolor sit amet" to a supervisor, the message received is matched with *String.class* into *SupervisorActor.createReceive()*. In turn, *supervisorActor* sends the message to *myActor* which response to the *supervisorActor* with the return of the *processMessage()* function passing the received message as a parameter. When *supervisorActor* receives the response, it prints the response.

2.4.2 Play Framework

Play framework² is a high-productivity Java and Scala web application framework that integrates components and APIs for modern web application development. The Play has a Model-View-Controller (MVC) architectural pattern framework that provides predictable and minimal use of a computational resource for highly-scalable web applications;

As a full-stack framework, Play includes all of the components you need to build Web Applications and REST services, such as an integrated HTTP server, form handling, Cross-Site Request Forgery protection, a powerful routing mechanism.

To manage the REST request, Play uses a configuration file where it maps the request URL to an internal system method. Besides, Play has integration with Akka's actors, so any web system receives the message via HTTP and passes the message to the actor, who will process the message and return the message to the client in response.

The most appropriate practice for performing this communication with actors is using Akka's `ask()` method because of return a Future object with the response of the requisition.

2.5 Other Technologies for Smart Cities

There are other technologies to comply with the Smart Cities requirements, for example, Spark, Cassandra and Orion Context Broker responsible for analyzing, store and disclose data respectively. In the next subsections, these technologies will be detailed.

2.5.1 Spark

"Apache Spark³ is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R languages, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming" (APACHE, 2019).

Spark can offer top-level abstraction and APIs as specific three data structures to analyze and process extensive data: Resilient Distributed Dataset (RDD), DataFrame, and Dataset.

- A Resilient Distributed Dataset (RDD) is an immutable distributed collection of data, partitioned between nodes in a cluster that can be operated in parallel with a low-level API that offers transformations and actions. RDD also supports two types of operations: Actions and Transformations;

² <https://www.playframework.com>

³ <https://spark.apache.org>

- DataFrame is a distributed collection of immutable data. Unlike an RDD, data is organized into named columns, such as a table in a relational database. Allows developers to impose a structure on a distributed data collection, allowing for high-level abstraction;
- Dataset is a collection of strongly typed objects, dictated by a case class that you define in Scala or a class in Java.

2.5.2 Cassandra

Apache Cassandra⁴ is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

Cassandra is designed to have read and write throughput both increase linearly as new machines are added, with the aim of no downtime or interruption to applications. Data is automatically replicated to multiple nodes for fault-tolerance. Replication across various data centres is supported. Failed nodes can be replaced with no downtime.

Cassandra has Hadoop⁵/Spark integration, with MapReduce support. Cassandra introduced the Cassandra Query Language (CQL). CQL is a simple interface for accessing Cassandra as an alternative but so similar to the traditional Structured Query Language (SQL).

2.5.3 Orion Context Broker

Orion Context Broker⁶ is a C++ implementation of the Next Generation Service Interfaces (NGSI) REST API binding developed as a part of the FIWARE⁷ platform. The FIWARE aims to support the development of new smart applications and services for the Future Internet in various industries such as Smart Cities, IoT, logistics, renewable energy, sustainable transport. Through a set of open APIs, generic and extensible. APIs are implemented through components called Generic Enablers (GE).

"Orion Context Broker can be used to create context elements and manage them through updates and queries. In addition, you can subscribe to context information so when some condition occurs (e.g. the context elements have changed) you receive a notification" (FIWARE, 2018).

NGSI is an HTTP-based protocol that allows communication between the various FIWARE GEs; currently, NGSI is being used in the second version. Orion is an NGSI server

⁴ <http://cassandra.apache.org>

⁵ <https://hadoop.apache.org>

⁶ <https://fiware-orion.readthedocs.io>

⁷ <https://www.fiware.org/>

implementation to manage context information in a highly decentralized and large-scale manner.

NGSI is defined based on the concept of context entities. Context entities are virtual representations of all kinds of real-world physical objects, such as a table, room, or person. Context elements represent each context entity. A context element has an identifier (EntityID), a type (EntityType), and context attribute set. Each context attribute has a Name, Type and Value and can have a meta-data set. Each meta-data must also have Name, Type and Value.

These context entities in NGSI format are inserted in Orion through REST requests and sending an object in JSON format. In addition to REST requests to register new context entities, there are also requests to update and delete. You can also subscribe to a context element or a particular type of item so that when you have an update, you get notified. An example of a context element represented in NGSI format can be seen in Listing 2.4.

```
1 {  
2   "type": "AirConditioning",  
3   "isPattern": "false",  
4   "id": "ac1",  
5   "attributes": [  
6     {  
7       "name": "Status",  
8       "type": "Boolean",  
9       "value": "true"  
10    },  
11    {  
12      "name": "Temperature",  
13      "type": "Int",  
14      "value": "24"  
15      "metadatas": [  
16        {  
17          "name": "Data",  
18          "type": "DateTime",  
19          "value": "2019-08-13 12:00:00"  
20        }  
21      ]  
22    }  
23  ]  
24 }
```

Listing 2.4 – NGSI example

This JSON code snippet represents a real-life air-conditioning object with two attributes: (i) *Status* that informs if it is turned on and (ii) *Temperature* that store the air-conditioning temperature and date-time that the temperature measurement was taken. To send this context entity to Orion Context Broker simply make a REST request to `http://<orionURL>:1026/v1/contextEntities` with POST verb, throwing in the body of the message this JSON code and setting the message header to *Content-Type: application/json*.

3 Related Works

This chapter presents some works that are strongly related to what is proposed here. For this, a systematic review was performed. Papers ranging of articles, technical reports, monographs, dissertations and theses were evaluated based on the guidelines provided by (KITCHENHAM, 2004) and is divided into five steps:

- Planning - definition of search criteria, and the inclusion and exclusion criteria to collect related works for Blockchain usage in the context of Smart Cities;
- Execution - performing a search for the primary studies (i.e., papers, conference publications, periodicals, thesis) applying the inclusion and exclusion criteria in order to filter the results that were relevant to this review;
- Requirements Taxonomy - definition of the relevant requirements for this study;
- Evaluation Results - extraction of data from the search results to formulate the answers for the research questions;
- Threats to Validity - a discussion of the main threats to the validity of this study.

3.1 Systematic Review

This systematic review was done aims to respond to the RQ1. In the next subsections explain how each one of these steps was performed.

3.1.1 Planning

This step of the systematic review defines: (1) research questions; (2) search strategies; and (3) inclusion and exclusion criteria.

3.1.1.1 Systematic Review Research Questions

The following Systematic Review Research Questions (SRRQ) were defined in order to discover the primary studies that present some information about Blockchain usage in Smart Cities environments. *SRRQ1*: How can Blockchain be used in Smart Cities environments? Moreover, *SRRQ2*: How can Blockchain be used in systems integration?

3.1.1.2 Search Strategies

To find studies related to the research questions, the following keywords are chosen: "Blockchain", "integration", "Smart City" and "System of System".

Similar contexts or synonyms for these keywords were identified. "Integration" may be referred to as "interoperation" or "interoperate" or "interoperability" or other word variations. Also, "System of System" and its abbreviations may be a synonym for "Integration". "Ethereum" may be a synonym for "Blockchain" in specific contexts.

Thus, it was established two search strings, namely:

- ("Interoperability" OR "interoperate" OR "interoperable" OR "interoperation" OR "integrate" OR "integration") AND ("Blockchain" OR "Ethereum") AND ("Smart Cities" OR "Smart City");
- (("Interoperability" OR "interoperate" OR "interoperable" OR "interoperation" OR "integrate" OR "integration") OR ("System of System" OR "SoS" OR "SOS")) AND ("Blockchain" OR "Ethereum").

These strings were used in IEEE Xplorer, ACM Digital Library, and Scholar Google repositories. These search string was adapted for each database in order to perform a directed search.

3.1.2 Inclusion and Exclusion Criteria

A set of inclusion and exclusion criteria to evaluate and select relevant studies were defined. The Inclusion Criteria (IC) was used to include relevant studies in this systematic review, namely:

- IC1: The study proposes, exemplifies or evaluates the use of Blockchain in the context of Smart Cities.
- IC2: The study proposes, exemplifies or evaluates the use of Blockchain in the context of System of Systems.

The Exclusion criteria (EC) was used to exclude studies with no relevance for this review (studies that do not contribute to answering RQ1 or RQ2). The ECs are:

- EC1: The study is not related to Systems Integration or Smart Cities;
- EC2: The study is not in English;
- EC3: The study does not have abstract, or the full text is not available;
- EC4: The study consists of a compilation of studies from conferences or workshops, for example;
- EC5: The study was written before 2015 up to 2018.

The studies selected for this systematic review are defined as a study that satisfies at least one of the inclusion criteria and does not satisfy any of the exclusion criteria.

3.1.3 Execution Results

After completing the searches, papers found were analyzed. The found papers represent the number of papers returned by the automatic searching process. The filtered papers were obtained after reading their titles. Already, the selected papers already represent the filtered papers whose abstracts and keywords present relevant terms and research related to the objective of the systematic review, and they were selected to be thoroughly read.

Table 2 – Systematic Review Summary

	ACM Digital Library	IEEE Explorer	Scholar Google	Total
Papers Found	42	27	82	149
Filtered Papers	20	15	41	76
Selected Papers	7	5	13	25

As summarized in Table 2, from 42 papers found by the ACM Digital Library search engine, 20 were filtered, and seven were selected for the second stage; from 27 found by the IEEE Explorer search engine, 15 were filtered, and five were selected for the second stage; from 82 papers found by the Scholar Google search engine, 41 were filtered, and 13 were selected for the second stage. In the total, it was selected 25 relevant papers for our study, as listed in Table 3. That table shows the identification, author, title and year of each selected paper, and such information will be used to refer to these studies in the following sections.

Table 3 – Selected papers list

ID	Author	Title	Theme	Year
S1	Peterson et al.	A blockchain-based approach to health information exchange networks	Health	2016
S2	Pinna and Ibba	A blockchain-based Decentralized System for proper handling of temporary Employment contracts.	Employment	2017
S3	Ekblaw et al.	A Case Study for Blockchain in Healthcare: “MedRec” prototype for electronic health records and medical research data	Health	2016
S4	Zhang, Xue and Huang	A secure system for pervasive social network-based healthcare	Health	2016
S5	Zhang et al.	Applying software patterns to address interoperability in blockchain-based healthcare apps	Health	2017
S6	Kinai et al.	Asset-Based Lending via a Secure Distributed Platform	Lending	2017
S7	Heijden et al.	Blockchain: Scalability for Resource-Constrained Accountable Vehicle-to-X Communication	Automotive	2017
S8	Sharma, Moon and Park	Block-VN: A distributed blockchain based vehicular network architecture in smart City	Automotive	2017
S9	Chanson et al.	Blockchain as a privacy enabler: an odometer fraud prevention system	Automotive	2017
S10	Brodersen et al.	Blockchain: Securing a New Health Interoperability Experience	Health	2016
S11	Simona Ibba et al.	CitySense: blockchain-oriented smart cities	IoT	2017
S12	Nichol, PMP and CMPI	Co-Creation of Trust for Healthcare: The Cryptocitizen Framework for Interoperability with Blockchain	Health	2016

Table 3 continued from previous page

ID	Author	Title	Theme	Year
S13	Swan	Connected car: quantified self becomes quantified car	Automotive	2015
S14	Korpela et al.	Digital supply chain transformation toward blockchain integration	Supply Chain	2017
S15	Chen et al.	Exploring blockchain technology and its potential applications for education	Education	2018
S16	Ekblaw et al.	MedRec: Medical Data Management on the Blockchain	Health	2016
S17	Ivan	Moving toward a blockchain-based method for the secure storage of patient records	Health	2016
S18	Walker et al.	PlaTIBART: a platform for transactive IoT blockchain applications with repeatable testing	IoT	2017
S19	Rowan et al.	Securing vehicle to vehicle communications using blockchain through visible light and acoustic side-channels	Automotive	2017
S20	Abbasi and Khan	VeidBlock: Verifiable identity using blockchain and ledger in a software defined network	Authentication	2017
S21	Dahri et al.	Interoperable Health Care System Using Blockchain Technology	Health	2019
S22	Brandão et al.	A Smart City's Model Secured by Blockchain	Security	2018
S23	Pieroni et al.	Smarter city: smart energy grid based on blockchain technology	Energy	2018
S24	Rotuna et al.	Smart City Ecosystem Using Blockchain Technology	IoT	2019
S25	Montes et al.	Applying Blockchain to Supply Chain Operations at IBM Implementing Agile Practices in a Smart City Environment	Supply Chain	2018

3.1.4 Requirements Taxonomy

According to the results of the analysis of the paper, some Blockchain requirements in the context of Smart Cities were identified. Table 4 lists the full set of requirements found by our systematic review. In the next few paragraphs, each requirement is defined and discuss its importance to the Blockchain in the context of Smart Cities.

Security is "a composite of the attributes of confidentiality, integrity, and availability" (AVIZIENIS et al., 2004). Blockchain uses cryptography to protect data via many protection mechanisms. For example, it can use authentication of communicating parties/messages, the confidentiality of secret keys, in addition to other security mechanisms.

Privacy is needed since some transaction data should be private and accessible only by authorized participants. It can be ensured by using a public-private key to masquerade sensible data or using a private Blockchain (ØLNES; UBACHT; JANSSEN, 2017).

Data Sharing is "the transmission of bulk data to other institutions [...] forces institutions to yield operational control of their data" (PETERSON et al., 2016). As Blockchain is a decentralized database, therefore, it should share, replicate and synchronize all data among network members. This fact could be a challenge since each transaction has to be logged in multiple places at the same time.

Decentralized requirement "is necessary in order to ensure the involvement of many actors in the same network and guarantee full transparency and reliability between people who do not know each other" (PINNA; IBBA, 2017). The Blockchain does not

have a single authority that can approve the transactions or set specific rules to have transactions accepted. All the participants in the network have to reach a consensus to accept transactions.

Resilience is served by the **fault tolerance** mechanism present in Blockchain's technology. A System is resilient when they are protected against the faults that might affect it (AVIZIENIS et al., 2004). Smart Cities solutions need to be resilient enough to sustain data availability, even at a high level of requests by their applications/clients. Since Blockchain has a decentralized database, if one Blockchain's node becomes inaccessible, it is possible to access the transactions data from another node.

Reliability is the continuity of executing a service correctly (AVIZIENIS et al., 2004). Due to Blockchain's decentralized nature, a failure of a single peer¹ should not affect the operation of the rest of the network. "This avoids the single point of failure and ensures the high reliability of applications that are based on Blockchain technology" (CHEN et al., 2018).

Immutability is the fact that data cannot be edited after stored. In the Blockchain context, after concluding a transaction, transaction data cannot be altered/updated. Blockchain uses cryptography, hashing, and decentralized structure to protect the data, making it impossible to change the data after inserted in the transaction.

Data Integrity / Consistency is the trustworthiness and accuracy of a data stored due to the absence of inadequate system change (AVIZIENIS et al., 2004). Blockchain has to ensure that the data is correct and that only authorized people are allowed to add new data. Furthermore, the data stored in Blockchain should be validated by the rest of the Blockchain network.

Traceability / Historical Data is the ability to perform a "traceback verification of all the identity instances available in the block" (ABBASI; KHAN, 2017). Blockchain should be capable of accessing all historical data stored at the transactions, making it possible to track data and verify any information.

Scalability, in the Blockchain context, for (KAKAVAND; SEVRES; CHILTON, 2017) is the increase of:

- Number of nodes: system performance changes as the number of nodes changes;
- Number of transactions: system performance changes as the number of transaction submissions per second changes;
- Number of users: system performance changes as the number of active users submitting transactions changes;
- Geographic dispersion: system performance changes as the geographic dispersion of nodes changes.

¹ In this paper, peer and node are considered synonymous

In a Smart City scenario, Blockchain network should be scalable since the increase of those aspects could occur very quickly.

Interoperability is defined as "the ability for different information technology systems and software apps to communicate, exchange data, and use the information that has been exchanged" (ZHANG et al., 2017). Blockchain should share data and services with IT systems, allowing interoperability between such computational solutions.

Transparency in the Blockchain context, is the fact that "all Blockchain network exchanges are cleared in the Blockchain, which means a total, verifiable and unchanging record of any action exists" (SHARMA; MOON; PARK, 2017). Blockchain should ensure that all transactions are viewed and could be scanned by all nodes in the network, presenting the transparency desired by Smart City applications, even with some private information.

Smart Contracts are executed as a program or a script to ensure the correct execution of a transaction. "All the nodes in a Blockchain network have to execute the programs/scripts independently. Deterministic execution of programs/scripts on the same set of inputs and conditions always produces the same output on all the nodes, which helps avoid inconsistencies" (SINGHAL; DHAMEJA; PANDA, 2018). In the Bitcoin, scripts are simple and not Turing-complete and allow only a few instructions. Already Ethereum and Hyperledger allow complex executions, Ethereum's smart contracts are executed Ethereum Virtual Machine, and Hyperledger's Chaincodes are running of compiled machine codes inside Docker images from Java and Go languages.

3.1.5 Evaluation Results

This section presents the evaluation results of the systematic review. Table 4 shows how each one of those 25 selected papers (as show in Section 3.1.3) are related to each requirement presented in Section 3.1.4.

In this direction, grades to specify how each paper covers the requirements: grade 0 (zero) if the study does not mention the requirement; grade 1 (one) if the study only mentions the requirement, without any explanation; grade 2 (two) if the study explains the importance of the requirement; and grade 3 (three) if the study describes how the authors implement the requirement.

As a way to give an ordered classification of the list of selected articles, the grades for each requirement was added to reach a final grade for each paper. It is essential to mention that while some requirements are more critical than others, to simplify the analysis, it was considered that all requirements have the same weight.

Table 4 – Requirements Taxonomy

Requirements	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25
Security	2	1	2	3	1	2	1	3	1	3	1	2	2	2	3	2	2	1	3	3	1	3	1	2	2
Privacy	3	0	1	3	1	0	1	2	2	3	1	3	2	1	2	1	1	2	0	2	1	2	1	0	1
Data Sharing	3	1	2	2	2	0	0	2	1	2	1	1	1	1	1	2	3	0	1	0	0	0	1	1	1
Decentralized Access	0	2	1	0	1	0	0	1	0	1	1	2	1	1	2	2	2	1	1	0	1	2	1	0	1
Resilient	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
Immutability	1	2	1	0	2	0	0	0	1	2	0	2	0	0	2	1	2	1	0	0	1	0	1	1	2
Data Integrity or Consistency	1	1	1	1	1	0	1	0	1	1	0	1	0	0	1	1	0	1	1	0	1	0	0	1	1
Traceability or Historical Data	1	2	1	0	1	0	0	1	1	0	1	2	0	0	2	2	1	0	0	1	1	2	1	0	1
Scalability	0	0	0	0	1	0	2	1	1	2	1	1	1	0	0	0	1	0	0	0	1	0	0	1	0
Interoperability	3	0	3	0	3	0	0	0	0	3	0	3	0	2	0	2	2	1	1	0	1	0	1	2	0
Transparency	0	2	1	0	2	0	1	2	0	2	2	2	0	0	1	1	0	0	0	0	1	0	1	0	0
Reliability	0	1	0	0	0	0	1	0	1	1	1	0	0	1	2	0	0	0	1	1	1	0	0	1	1
Smart Contracts	2	2	3	0	3	0	0	1	0	2	2	2	2	2	2	2	2	2	0	1	1	2	2	1	0
Total	16	14	16	9	19	2	7	13	9	22	11	21	9	10	18	16	16	10	8	8	11	12	10	11	10

Papers S1, S3, S4, S5, S10, S12, S16, S17 and 21 adopts Blockchain to store data about health care. Article S1 is concerned with data sharing securely. It allows the use of such data by third parties to provide data interoperability. On the other hand, papers S3 and S16 present a software prototype for electronic records and medical research data safely.

To achieve this, they use smart contracts to orchestrate content-access into the distributed storage system. Thus, they also are concerned about interoperability and data sharing. Paper S4 and S22 are focused on security requirements. It uses security proofs and formal verification to ensure safe Blockchain specification to share health data in a pervasive social network.

The papers S5 and S17 describe the motivations for using Blockchain technology to solve healthcare interoperability issues, focusing on integration complexity, data storage requirements, security concerns, and tracking health data from large patient populations. Besides, S5 study presents a case study to highlight implementation challenges while taking care of essential requirements.

The integration benefits of current health IT investments with a permission Blockchain distributed ledger technology was explored in S10. To reach this, S10 presents three use cases where "Blockchain technology could significantly help in solving intractable issues that have had a prolonged negative impact on the healthcare industry" (BRODERSEN et al., 2016). Paper S12 presents a concept for the co-creation of trust for healthcare applications, taking into account three requirements: interoperability, security, and payment system.

In this case, Blockchain is used to integrate and facilitate the reliable purchase of drugs since "this will provide greater transparency to reduce and eliminate fraud in prescriptions, billing, and security breaches that trigger patient identity fraud. (NICHOL; PMP; CMPI, 2016)".

A temporary employment contract system is discussed in S2. This study uses Blockchain to model the states of an employment relationship safely and complying with legislation. It aims to protect the rights of workers and companies at the same time. Traceability can be used to detect illegal actions and verify the contracts, once the "immutability of the data saved on Blockchain makes the payments that are consistent with what is stated in the contract terms" (PINNA; IBBA, 2017).

Asset-based lending via a secure distributed platform is presented in S6. This study uses Blockchain to verify, authorize and control each transaction that occurs between a seller and buyer in a supply chain.

Blockchain is used as a solution for automotive problems in studies S7, S8, S9, S13, and S19. Paper S7 uses Blockchain to improve the trust requirements on users of a vehicular communication system, since each vehicle accumulates information about its state and, through received messages, about other vehicles at the same area. S19 is also

concerned about vehicular networking and communication.

In the proposed solution, "the scheme verifies the location and identity of the vehicle being communicated with and incorporates that vehicle identifier in the cryptographic setup exchanges "(ROWAN et al., 2017) and uses Blockchain technology to provide interoperability among untrusted vehicles and manufacturers. Paper S8 proposed the Block-VN model, a distributed Blockchain architecture based on the vehicle network. It allows creating a network of vehicles to discover and share their resources.

Article S9 uses Blockchain to store car odometer data since odometer fraud is a massive problem in many countries. With the Blockchain, the odometer data is stored securely and efficiently checked. Lastly, paper S13 proposes the use of Blockchain in the vehicle ownership record, providing authenticated on-demand real-time access. It could be a better alternative for current keyless entry mechanisms.

Solutions involving Internet of Things (IoT) is presented in S11, S18 and S24. Article S11 proposed a Blockchain-based system, called CitySense, to manage the environmental context, making transparent data from IoT devices. With this system, the people can improve monitoring environmental quality in the city to promote greater awareness of city health. The S18 presents Transactive IoT Blockchain Applications with Repeatable Testing (PlaTIBART) platform. Such a platform allows developing resilient, fault-tolerant IoT-Blockchain applications and middleware.

In the context of paper S14 and S25, Blockchain is used in the context of a digital supply chain to create a business to business integration. This study shows the collection of the essential requirements to perform this integration and discusses how these requirements can be met when using Blockchain.

Article S15 explores the potential of Blockchain for education applications. "It (Blockchain) can store a complete, trustworthy set of record of educational activities, including the processes and results informal as well as informal learning environments. It can also record teachers' teaching behaviours and performance, thus providing a reference for teaching evaluation" (CHEN et al., 2018).

Blockchain is used to verify identity in a Software-Defined Network in paper S20. Such article proposes VeidBlock: a security framework based on the core concepts of Blockchain and distributed ledger. Using anonymous authentication and maintaining security and privacy with a security protocol, such an approach provides authentication and identity verification.

In the last line of Table 4, there is the total sum of the scores assigned to each requirement for each paper. Although all papers present a solution for Smart Cities using Blockchain as a technological solution. Studies S10 and S12 (with higher grades in the total sum) were concerned with presenting the justification for the need of most of the requirements while other articles were only concerned with presenting the Blockchain and show how the Blockchain was used in the proposed solution by the paper.

3.1.6 Threats to Validity

A significant threat to the validity of this systematic review can be considered the completeness of this study, i.e. whether all relevant related articles were included. This problem may have occurred because relevant studies may not have been found by search engines, for example, by the technical limitations of search engines.

3.2 Others Blockchain Solutions

Even some works presented in Section 3.1 already use Blockchain in the context of Smart Cities (for example money, health (NICHOL; PMP; CMPI, 2016), automotive (SHARMA; MOON; PARK, 2017), education (CHEN et al., 2018)), but none of these applications used a platform that could be generic enough to be used generally for other types of domains.

On the other hand, there are other solutions for general purpose that are developed for use by the market, out of the Smart Cities context, which focuses on different approaches, namely:

- **EOSIO**² is a Blockchain platform for businesses with industry-leading transaction speed and flexible utility. "The smart contract platform claims to eliminate transaction fees and also conduct millions of transactions per second." (GRIGG, 2017)
- **NEO**³ "is the use of Blockchain technology and digital identity to digitize assets, the use of smart contracts for digital assets to be self-managed, to achieve "smart economy" with a distributed network" (NEO... , 2018).
- **Hyperledger Fabric**⁴ is the first ledger developed by The Linux Foundation in partnership with IBM and Digital Assent. It is a framework for application development where the Smart Contracts are named as ChainCode. The communication between nodes in the Hyperledger Fabric occurs through channels, making transactions private to channel members only. The main feature of this solution is that it is modular and configurable according to the application developed.
- **Quorum**⁵ is an open-source Blockchain platform, based on Ethereum, that combines the innovation of the public Ethereum community with enhancements to support enterprise needs. Brings some extensions over public Ethereum focused on enterprise needs, namely: transaction and contract privacy, multiple voting-based consensus mechanisms, network/peer permissions management, and higher performance.

² www.eos.io

³ www.neo.org

⁴ www.hyperledger.org/projects/fabric

⁵ www.goquorum.com

- **Parity Substrate**⁶ "is a Blockchain development framework with many cool features like upgradable Blockchains, modular architecture, customizable block execution logic, and hot-swappable consensus."
- **Corda Platform**⁷ Corda is an open-source Blockchain platform that enables businesses to transact directly and in strict privacy using smart contracts, reducing transaction and record-keeping costs and streamlining business operations. In a world of permission-less Blockchain platforms, in which all data is shared with all parties, Corda's strict privacy model allows businesses to transact securely and seamlessly. R3 delivers two interoperable and fully compatible distributions of the platform – Corda, a free download based on the code available on GitHub; and Corda Enterprise, a commercial version which offers features and services fine-tuned for modern-day businesses.
- **BigchainDBn**⁸ is an open-source, decentralized database with the purpose of adding Blockchain data structure features to existing NoSQL distributed database architectures. Features such as scalability, read and write speed, and ability to extract data through query languages such as SQL are elements that contrast with the nature of traditional Blockchains.

As can be seen, despite their relevance to the works presented in this chapter, these works do not encompass a solution that can be used in a general domain in Smart Cities context. Thus, there is a gap between those two approaches. A Blockchain solution for general propose focused on scalability, flexibility and reliability in Smart cities context; this is precisely the Orthus' purpose.

⁶ www.parity.io/substrate/

⁷ www.r3.com/corda-platform/

⁸ www.bigchaindb.com

4 Orthus Platform

This chapter aims to present Orthus: a Blockchain platform for Smart Cities. The choice of the name Orthus comes from the symbology of the two-headed dog that holds the cattle of giant Geryon of Greek mythology¹, once Orthus platform also aims to holds and protect data in a distributed and reliable way. It also presents the current state of implementation of the various Orthus architecture modules, as well as the current implementation issues of each of its components.

Figure 3 – Orthus Logo

Source: Prepared by the author.

4.1 Architecture Overview

Orthus platform was designed based on the requirements elicited as described in Chapter 3. It is a permissioned (private) Blockchain platform that supports the requirements to share information between Management Information System (MIS) present in Smart Cities and promote Interoperability between them. The reason for using Blockchain to provide system integration in the context of Smart Cities is because it involves systems from multiple sources (different private companies and municipal or federal agencies) where there is no reliability between those systems. With Blockchain data from these systems would be shared securely, immutable with decentralized management.

To perform this integration, each MIS must implement the Orthus component to be part of the network. Orthus components are instantiated once for each system, and this association (System + Orthus components) is called a node. Communication between a system and the Orthus Gateway component occurs through REST requests. The Orthus network is comprised of distributed nodes and a network of brokers that enables data exchange, in the form of contextual elements, by sharing transactions and blocks that were created by one of the nodes and validated by all.

An example of using Orthus to integrate the systems: Hospital Information System, Medlab and Pharmacy Control as described in the scenario presented in the Introduction

¹ <https://www.greeklegendsandmyths.com/orthus.html>

can be seen in Figure 4. In it, you can see the integration of these systems in two different cities where each city would have its Orthus network giving rise to an Orthus organization. You can also see the integration of the organization of these two or more cities, by using a new broker to integrate them. The justification for integrating these organizations would be to be able to access patient data in another city, for example.

Figure 4 – Medical scenario integration through Orthus

Source: Prepared by the author.

To achieve this integration between the systems, the platform architecture must contain the following components, namely: i) an API gateway, to receive the REST requests of creating new transactions from systems; ii) a Controller Service, to orchestrate the solicitations done in Orthus component; iii) a Transaction Service, this component is responsible for validating and creating new transactions; iv) Block Service, the component responsible for creating a new block with the transactions added before; v) Blockchain Service, this component is responsible close the block with the transactions and inserts the block into the database (adding this block into the Blockchain); vii) a Database, the component responsible for storing the Blockchain; vi) Communication Service, the component responsible for communication with other Orthus peers, it utilizes a context broker bus to facilitate the conversation and viii) Context Broker, this component is responsible for integrating all Orthus peers being a bus of communication for them.

Figure 5 shows the integration of the internal components of Orthus node as previously described. It also shows the integration with a possible system to be integrated into the Orthus platform. In this case, it exemplifies with the integration of Pharmacy Control System with Orthus platform. Where a user requests the Pharmacy Control System and it sends a REST request to Orthus to access the Blockchain data or request a new transaction.

Figure 5 – Orthus components integration

Source: Prepared by the author.

Remember that for each system to be integrated into the platform, an instance of the Orthus node is required. In this way, it is possible that a system can share information securely between the systems that are part of the platform. Since, in each of the Orthus nodes, the data is verified and validated.

4.2 Requirements Fulfillment

In this section describes how each requirement that was elicited, defined and discussed by the systematic review in Section 3 is fulfilled by the proposed platform.

Security is supported by the Blockchain's structure, which uses cryptography to protects the data via several different mechanisms. It uses authentication of communicating parties/messages and uses the *Security Library* component to ensure **privacy** by using the concept of the public and private key.

Transparency is served by the free access of any transaction stored by the Blockchain in any peer in the network. Even though some information can have a certain level of privacy using a public and private key. **Data sharing** is supported by the *Context Broker* and *Communication Service* components that are responsible for replicating the database of a peer to all network peers, promoting **decentralization** of the Blockchain.

The Blockchain technology heavily relies on a distributed agreement (protocols) across on an excellent fault tolerance mechanism. Fault tolerance is a means for achieving some levels of reliability, availability, data integrity, resilience. **Resilience** is supported by the fact that the *Communication Service* component sends and receives all transactions made by the network as a way to replicate all data in all peers. If a peer becomes inaccessible, the user can access another peer in the same way.

Moreover, when that peer goes back up and running, it only needs to synchronize

the database with the rest of the network to get back to regular operation. In this way, it also ensures **reliability**. **Immutability** is supported by the Blockchain's structure, which uses old block information to generate the new hash, thus making it unfeasible to change old data. This factor, along with the fact that every transaction is validated before being added to a block, ensures **data integrity/consistency**.

Traceability/Historical Data is supported by the *Blockchain Service* component. Through this component, it is possible to access all transactions performed and stored in the Blockchain. **Scalability** is supported by the *Controller Service* component that checks the number of transactions to be performed. If this amount is greater than the one supported, the Blockchain should ask other peers to ensure that all transactions are processed.

Interoperability is supported by data sharing through the peer to peer network, in which data can be accessed in any peer since data is replicated across all peers in the network. Thus, the proposed platform can fulfil almost all the requirements found in the systematic review. However, yet is needed to implement the Smart Contracts. In Orthus architecture, it will be used to request some task for the systems, making possible share services between the Orthus platform.

4.3 Components Communication Process

To improve understanding of the messages exchanged by the Orthus components, this section presents some diagrams in the Business Process Model and Notation (BPMN) format of the main processes performed by the services into the platform. BPMN provides a graphical notation for specifying business processes where it is composed of *pool* that represents the participants in a process (WHITE, 2004).

These *pools* are used to organise and categorise activities within a pool according to function or role. A *pool* contains the *flow objects*, *connecting objects* and *artefacts*. *Flow objects* are of three types: *events*, *activities*, *gateways*. An *event* is represented with a circle and denotes a start or end event. An *activity* is a *flow object* and is represented with a rounded-corner rectangle and describes the kind of work which must be done.

Flow objects are connected to each other using *connecting objects*, most often by sequence flow or message flow. A *sequence flow* is represented with a solid line and arrowhead, and shows in which order the activities are performed. A *message flow* is represented as a dashed line with an open arrowhead and is used to show the flow of messages between two separate process participants.

4.3.1 New Transaction Creation

The messages exchanged to create a new transaction can be seen in figure 6. The creation of a new transaction begins when a user sends POST requests to a node with a JSON object. The attributes contained in this JSON varying by application and context, but following the NGSI standard, an example can be seen in the Listing 6.1.

This request arrives in the Gateway which it then forwards to Controller Service. Arriving at Controller Service, the message is transformed to JsonMessage and sent to Transaction Service, as can be seen in Figure 6.

Figure 6 – BPMN of creation of new transaction process

Source: Prepared by the author.

Within Transaction Service, this message is transformed into a transaction type object to be validated. After validation, this transaction is placed inside a ValidatedTransactionMessage object. This message is returned to the Controller Service, which in turn forwards to (i) Communication Service for insertion into Context Broker and (ii) to the user in response to the request made by him. In this way, a new transaction is created and disseminated on the network for later insertion into a block and being mined.

4.3.2 Add a Transaction

The messages exchanged to add a transaction can be seen in Figure 7. The process of adding a transaction that was created by another node, as can be seen in the Figure 7 begins when the Context Broker send POST requests to all nodes with the transaction in NGSI format since all Orthus nodes are subscribed to receive notifications when new transactions are entered.

As with the creation of a new transaction process, the Gateway receives the request and forwards the JSON object as AddTransactionMessage class to Controller Service. In turn, Controller Service forwards it to the Transaction Service that stores the transaction

Figure 7 – BPMN of add a transaction process

Source: Prepared by the author.

in a temporary vector, since this transaction has already been verified by the node that created it. This transaction stays in the temporary vector until the process of creating a new block and its mining begins.

4.3.3 Create and mine a new Block

The process of creating and mine a new block can be seen in Figure 8. When it comes time to create a new block, the Controller Service sends a message to Transaction Service requesting transactions stored by the actor Transaction Service. These transactions are sent back to Controller Service within the CreateBlockMessage class, which in turn forwards this message to Block Service.

The Block Service creates a new Block with all transactions stored in the temporary vector, that was previously received. Once created, the block is validated, and Block Service sends back this block to Controller Service as ValidatedBlockMessage. Upon receiving validatedBlockMessage, Controller Service forwards the unclosed (unmined) block to be disseminated by Communication Service to all other nodes by the Context Broker. At the same time, Controller Service also sends data from this block to Blockchain Service to be mined.

If this node can generate the hash to close the block before the other nodes, then the block is inserted into the database and forwarded to the Controller Service which will (i) notify the user of the success of the transaction and (ii) notify the others nodes of the block by that node through Communication Service to be verified by the other nodes, saving processing time this way.

Figure 8 – BPMN of create and mine a new block process

Source: Prepared by the author.

4.3.4 Add and mine a Block

The messages exchanged to mine a block can be seen in figure 9. The process of add and mine a block created by other node begins when the Context Broker makes a POST request to a node URL with /addBlock path with a JSON object representing a Block of in NGSI format.

The request is forwarded from Gateway to the Controller Service which in turn forwards it as AddBlockMessage with block data to be validated by Block Service. If the Block Service validates the block, it is sent back to Controller Service as InsertBlockMessage that is forward to Blockchain Service to be mined and inserted into the database. After that, the Controller Service notifies the user of block closure and the end of transaction processing and notifies others through Communication Service.

Figure 9 – BPMN of add and mine a block process

Source: Prepared by the author.

5 An Orthus Implementation

This chapter presents a possible implementation for the Orthus platform according to the architecture presented in Chapter 4. For this implementation, a set of technologies was chosen, but they can be replaced by similar ones that meet the architectural description. The justification for choosing each of the technologies and details of its implementation will be presented throughout this chapter.

5.1 Technologies chosen

A set of open-source technologies powering scalable real-time and data-driven applications was chosen to be used in the implementation, as described in Chapter 2.4. Play Framework for Java is used to create a REST API component to enable communication between the node and the system; this way, they can be on separate servers. The choice to use Play Framework to create the REST API is easy integration with Akka technology, but it could also use the Spring Framework.

Already the Akka for Java is used in the implementation of the actors of the Orthus component, these actors communicate by exchanging messages following the Reactive Design Patterns (KUHN; HANAFEE; ALLEN, 2017). Messages are exchanged locally, allowing maintain the relative simplicity of the event-driven programming model. As it is one of the leading and most used implementation of the Reactive Design Patterns, it is the reason to use Akka in the Orthus implementation.

Spark is used into the process to validate the transactions and also the blocks where it is necessary to verify Blockchain historical data. The reason for using Spark is that some studies, for example, Samadi, Zbakh and Tadonki (2016), show better data processing efficiency than Hadoop since it loads the data sets into memory. Nevertheless, other data processing mechanisms could be used, replacing Spark.

Cassandra is used to storing block generated data (transaction hash, previous block hash, timestamp) by the Blockchain within each node. The data itself will be stored within each of the systems based on the validated blocks. The reason for choosing the Cassandra database is because it integrates with Spark, where you can easily retrieve the data and transform it into RDD used by Spark. However, other databases could be used to store transaction and block data.

The development of the Orthus platform is part of the Smart Metropolis¹ project that has a partnership with FIWARE². In this project, computational solutions are developed using FIWARE components known as Generic Enablers (GE). For this reason,

¹ <http://smartmetropolis.imd.ufrn.br/>

² <https://www.fiware.org/>

Orion (a FIWARE GE) was chosen to be the context broker to communicate across the network, in distributed systems. Orion Context Broker is used as a bus to share the transactions and the blocks all over the nodes. Orion Context broker will also be used (not yet implemented) to communicate across organisations. For this, one organisation's Orion would subscribe to the integrating Orion to receive data from the other organisation, enabling data exchange between organisations and making their data available in more than one location.

This implementation was developed following the model-view-controller layer pattern offered by the Play Framework. In the next subsections, each layer of the Orthus component will be detailed. The folder structure is presented in Figure 10. The *Model* layer is formed by *DAO* and *transactions* folder. The *Control* layer is formed by *actors* folder. Into the *View* layer have some html files and a configuration file as will be described in the Subsection 5.1.3.

Figure 10 – Folder Structure

Source: Prepared by the author.

Into the *util* folder is defined some classes of objects used to support the implementation of other layers, such as messages that are exchanged during the execution of the actors. Next, each layer will be detailed.

5.1.1 Model Layer

Into the Model Layer is located the implementation of the classes that represent the objects will be stored in the Blockchain and the transactions that are validated and stored the actors of Controller Layer. The class *AbstractCapitalGood* describes all real-life objects used in Blockchain transactions (see Figure 11). New objects, in this layer, must be extended from the abstract class according to the application domain.

Figure 11 – Model Layer

Source: Prepared by the author.

The abstract class *AbstractTransaction* is responsible for defining an abstract base structure for transactions to create and store records that connect a capital good with people or status, as well as to keep the constraint of those transactions. A new *ValidatedTransaction* should extend *AbstractTransaction* to implement a transaction with a method to validate this transaction which will depend on the application domain.

The *Block* class contains all data structure and methods to support Blockchain functionalities, such as methods to create a new block with the transactions effected, methods to mine blocks to generate the hash and methods to validate the Blockchain.

The persistence of those block is a responsibility of the *BlockchainDAO* (see Figure 11). This class stores all blocks transaction data into the Cassandra database with the help of the *CommunicationActor* of Controller Layer. This Actor deals with the communication with other peers in the peer-to-peer network, supporting and facilitating the communication process using the *Orion Context Broker*, to assist the block mining process and to disclose and replicate the Blockchain data on other nodes.

5.1.1.1 Extension Points

As stated earlier, to use the Orthus platform is necessary to create classes within Model Layer according to the application domain and context. Thus, it is necessary to

implement classes from the `AbstractCapitalGood` class and the `AbstractTransaction` class. If the domain supports transactions without verification and validation, transactions can also be implemented from `SmartContract` class.

It is also necessary to define the JSON object format to represent the transactions that will be sent via POST to be registered in the Orthus platform. An example within a case study can be seen in Listing 6.1.

5.1.2 Controller Layer

The Controller layer is composed of Java Actor classes implemented using the Akka toolkit. Figure 12 shows the integration of the actors of the Controller Layer.

Figure 12 – Controller layer of reference platform for Blockchain usage in the Smart Cities

Source: Prepared by the author.

The *FacadeActor* the intermediary between the application's REST requests and the *ControllerActor*. This actor is responsible for control the number of transactions received to be created and validated in this node. When a new request arrives at a node, it is forwarded to the *ControllerActor* that delegates, according to the type of request, to the actors responsible for the transactions *TransactionActor*, blocks *BlockActor* and Blockchain *BlockchainActor*. These last three actors, (*TransactionActor*, *BlockActor*, *BlockchainActor*) were declared using Akka RoundRobinPoll with a resizer to dynamically create new instances of these actors according to the number of requests.

CommunitationActor component is responsible for communicating from the node to Orion. When initialised, this component performs subscriptions for the node to be informed when a new transaction or block is created.

5.1.3 View Layer

Once the Orthus component is accessed through an API, which returns the data requested by request, the View Layer has only the homepage with information about the API. In creating this API using the Play Framework, a route configuration file is created where it maps the URL path with the Facade class method to respond to the request made.

The Listing 5.1 shows the URL path of Orthus platform and the Facade method corresponding for each of them. Inline 2, it define a path to POST */newTransactionJson* to invoke the method *newTransactionJson* of *FacedeActor* aims to create a new transaction. The process of the creation of a new transaction will be presented in 4.3.1. Inline 4, it defines a path to POST */addTransaction* to invoke the method *addTransactionJson* of *FacedeActor* aims to add a transaction that was created by another node into the Orthus network. This process will be presented in 4.3.2.

```
1# Map static resources from the /public folder to the /assets
   URL path
2POST /newTransactionJson controllers.FacedeActor.
   newTransactionJson
3
4POST /addTransaction controllers.FacedeActor.
   addTransactionJson
5
6POST /addBlock controllers.FacedeActor.addBlockJson
```

Listing 5.1 – routes file

In the last line of the Listing 5.1, is defined a path to POST */addBlock* to invoke the method *addBlockJson* of *FacedeActor* aims to add a Block into the Blockchain, this block was created by another node into the Orthus network. This process will be presented in 4.3.2.

5.2 Implementation details

This section presents the code of the main points of the Orthus platform. The source code of the case study implementation is accessible via the link: <https://projetos.imd.ufrn.br/stefano/orthus>

5.2.1 Actors Instantiation

The code snippet is shown in Listing 5.2 is responsible for the instantiation of the actors of Control layer. In it, the `ControllerActor` class is a singleton responsible for controlling all requests that are made on an Orthus node. In lines 3 to 5, the actors are declared and will be instantiated when invoking the `getProps()` static method of the `ControllerActor` class. Inside the `getProps()` method, as seen inline 8 to 11, is invoked the method `createChildren()` and is returned the `ControllerActor Props`.

Inside the `createChildren()` method of the `ControllerActor` is instantiated the actors with a resizer that was instantiated as can be seen on line 16 and `RoundRobinPool`. This allows the application to control the number of instances of each actor type based on the number of requests. This resizer ensures that the response time for each request is met as quickly as possible.

```
1 @Singleton
2 public class ControllerActor extends AbstractActor {
3     static ActorRef blockchainActor;
4     static ActorRef transactionActor;
5     static ActorRef blockActor;
6     static ActorRef communicationActor;
7
8     public static Props getProps(ActorSystem system, String
9         systemName) {
10         createChildren(system, systemName);
11         return Props.create(ControllerActor.class);
12     }
13     public static void createChildren(ActorSystem system, String
14         systemName) {
15         blockchainActor = system.actorOf(BlockchainActor.getProps(
16             system));
17         int lowerBound = 2;
18         int upperBound = 500;
19         DefaultResizer resizer = new DefaultResizer(lowerBound,
20             upperBound);
21         transactionActor = system
22             .actorOf(
23                 TransactionActor.getProps(systemName).withRouter(new
24                     RoundRobinPool(50).withResizer(resizer)));
25         blockActor = system
26             .actorOf(BlockActor.
27                 getProps(systemName).withRouter(new RoundRobinPool(50).
28                     withResizer(resizer)));
29         communicationActor = system
30             .actorOf(
31                 CommunicationActor.getProps().withRouter(new
32                     RoundRobinPool(50).withResizer(resizer)));
33     }
34 }
```

Listing 5.2 – ControllerActor Code

5.2.2 Orion Subscription

The code snippet responsible for subscribing to Orion Context Broker is shown in Listing 5.3 where it has the *subscribeOrionContext()* method of actor CommunicatinActor. This method, in line 1, takes as its parameter the Orion URL, Orthus node URL, a description for the signature, the context element type that will be subscribed, and the context element attribute that when changed notify all subscribed nodes in him. Inline 3, a string is defined with these parameters and, in line 21 and 22, an Orion request is made by passing the previously created String as a parameter and request-response are returned.

```

1 public CompletionStage<HttpResponse> subscribeOrionContext(
    String urlOrion, String urlresponse, String description,
    String type, String attribute) {
2
3     String jsonString =
4     "{"
5     + "\"description\": \"" + description + "\", "
6     + "\"subject\": "
7     + "{"
8     + "\"entities\": [{"
9     + "\"idPattern\": \".*\", "
10    + "\"type\": \"" + type + "\"
11    + "}], "
12    + "\"condition\": {\"attrs\": [\"" + attribute + "\"]}}, "
13    + "\"notification\": {"
14        + "\"attrsFormat\": \"legacy\", "
15        + "\"http\": {\"url\": \"" + urlresponse + "\"}, "
16    + "\"attrs\": [\"*\"]}, "
17    + "\"throttling\": 5"
18    + "}";
19
20    try {
21        HttpRequest request = HttpRequest.POST(urlOrion).withEntity
            (HttpEntities.create(ContentTypes.APPLICATION_JSON,
                jsonString));
22        CompletionStage<HttpResponse> response = http.singleRequest
            (request, materializer);
23        return response;
24    }
25    /* Catches have been omitted */
26}

```

Listing 5.3 – Orion Subscription Method Code

With the parameters received, a String in JSON format is assembled, and an HTTP request is made to the POST verb for Orion with this String. Upon receiving this request, the Orion subscribes and returns a receipt confirmation. Orion makes a request to the registered URL of all Orthus nodes when a context element with a specific type and attribute is created or changed. In the example, Orthus nodes with the data that has been changed or created.

5.2.3 Block Hash Method Calculation

The Listing 5.4 is a code snippet responsible for calculating the Block Hash used to ensure the immutability of data in the block. As can be seen from line 3 to 5, where the previous block hash information, timestamp, and transaction data are used to calculate the hash of the block and the `applySha256()` method is executed with these parameters.

```
1//Calculate new hash based on blocks contents
2public String calculateHash() {
3    String calculatedhash = StringUtil.applySha256(
4        previousHash + Long.toString(timestamp) +
5        Integer.toString(nonce) + merkleRoot);
6    return calculatedhash;
7}
```

Listing 5.4 – Calculate Hash Method Code

6 Platform Evaluation

The purpose of this chapter is to compare the Orthus platform with other Blockchain solution about the response time for creating a transaction through Management Information Systems in the context of Smart Cities. The average response time and throughput are used as indicators in the analyses. The Orthus platform assessing will be done in comparing with the Hyperledger Fabric platform using the same case study. The choice to use Hyperledger Fabric platform for comparison is that it is widely used in the market and Androulaki et al. (2018) promises one of the highest transfer rates of the platforms found, more than 3500 transactions per second.

For this reason, this chapter contrasts the observed empirical results with the implementation of a case study, as described in Section 6.1.2.1, using Orthus against Hyperledger Fabric implementation. The remainder of the chapter is structured as follows. Section 6.1 describes the experimental settings and justifies the decisions made to ensure the study validity. The results gathered from applying average time response metrics are discussed in Section 6.2.1, while the results in terms of throughput are addressed in Section 6.2.2.

6.1 Study Setting

This section describes the configuration of our study, including the study phases, the case study implementation on both platforms, the assessment goals, and the study hypothesis.

6.1.1 Study Hypothesis

The hypothesis tested in this empirical assessment is stated as follows: "the use of the Orthus platform to provide a secure, scalable and reliable data sharing has enough transaction creation time to meet the needs of Management Information Systems in the context of Smart Cities". In order to test this hypothesis, a performance test was performed against the implemented case study in Orthus and Hyperledger Fabric platforms. This test verifies the average response time (in millisecond) and throughput (transactions per second) for a large number of simultaneous requests, which is quite common in Smart City applications.

6.1.2 Target Systems

This study used the implementation of the case study (described as described below) involving the land transfer operation between people to assess the average response time and the throughput on both platforms. The details of the implementation of this case study on the platforms can be seen in the First and Second phases in Section 6.1.3.

6.1.2.1 Case Study

This case study was designed to improve the land registry system into Natal Smart City Initiative¹. Natal is a city of Northeastern Brazil that has joined the IEEE Smart City initiative as an affiliated city. This initiative aims to transform Natal into a Smart City through the development of systems and applications to bolster the use of ICT as a means of contributing to improving the life quality of its citizens.

The use of Blockchain in land registry systems has been developed and tested in some countries according to the news item "Sweden tests Blockchain technology for land registry"² written by Gertrude Chavez-Dreyfuss. Moreover, this same author believes that: "The Blockchain technology works by creating permanent, public "ledgers" of all transactions that could potentially replace complicated systems such as clearing and settlement with one simple database."

Nikhilesh De in *Russia's Government to Test Blockchain Land Registry System*³ affirms that the Russian government is testing a blockchain-based land registry system since early 2018, and he also quoted a Russian resolution⁴: "The use of Blockchain will be aimed at increasing the availability of information on the property registry, guarantees of protection of property rights, as well as the level of citizens' trust in the sphere of turnover of real estate."

Currently, in Natal, there is a bureaucratic process to get the building permission involving three institutions: Land Registry Office, Municipal Secretary of Environment and Urbanism (Semurb) and Municipal Secretary of Taxation (Semut).

Figure 13 presents a Business Process Model and Notation (BPMN) of this process to show all the steps described below. First of all, the landowner has gone to the Land Registry Office to get land registration to prove ownership of the land. Then, he/she should go to the Semurb with the land registration and the architectonic project to begin an environmental impact assessment. If the land and the project conform to the environmental and architectural rules, then construction permission is granted. At the end of the construction process, it is necessary to go back to Land Registry Office, Semurb

¹ <https://smartcities.ieee.org/municipal-partners>

² Available in www.reuters.com/article/us-sweden-blockchain/sweden-tests-blockchain-technology-for-land-registry-idUSKCN0Z22KV

³ Available in www.coindesk.com/russias-government-test-blockchain-land-registry-system

⁴ <http://economy.gov.ru/minec/press/news/2017181003>

Figure 13 – Business Process Model and Notation (BPMN) of Case Study

Source: Prepared by the author.

and Semut to inform the conclusion of the building, registry the build in the land and start to pay the council taxes.

Sometimes there is no inspection in buildings sites because the inspection bodies are not notified of the new buildings, and there is no integration of the city hall system with the inspection body system. The problem is that these institutions are isolated and do not share data information, making fraud possible in this process. Blockchain can be used to solve this problem as a distributed database across a network of computers to store land information with security, reliability, and privacy in addition to enabling interoperability between these institutions.

The most critical operation involving this context is the land transfer operation between people as it is necessary to ensure that the seller does have the land. Moreover, it is precisely this transaction chosen to be implemented by the blockchain platforms to be assessed.

6.1.3 Study Phases

The study was divided into five major phases: (1) case study implementation using Orthus, (2) case study implementation using Hyperledger Fabric, (3) version alignment, (4) test environment definition, and (5) performance test case definition for the implementations developed in phase 1 and 2. Next, it will be described how each phase of the study was carried out.

- **First phase: Orthus implementation.**

This phase comprised the implementation of the case study using the Orthus platform. In it, each system has an internal Blockchain node which communicates with each other via Orion Context Broker. New transactions can be done by any system of the case study (Registry Office's System, Semut's System or Semurb's System) where each system is considered an Orthus node.

Figure 14 – Case Study Integration Diagram

Source: Prepared by the author.

Those transactions are validated and shared by the other peers as can be seen in Figure 14. From time to time, valid transactions are grouped in blocks; the first node that can close the block releases it to the other nodes in the network. When a node receives a block, it validates and inserts it in the Cassandra database. This process is known in the literature as consensus algorithms.

It is necessary to implement the Land class based on the AbstractCapitalGood class with the required terrain information (e.g. area, address, status) and implement the TransferLandTransaction class based on the AbstractTransaction class with the information required (buyer identifier, seller identifier, land identifier, value) as shown


```

10     "name": "VendorPublicKey",
11     "type": "string",
12     "value": "UAA4GNADCBiQKBgQDH+wPrKYyy..."
13 },
14 {
15     "name": "BuyerPublicKey",
16     "type": "string",
17     "value": "MIGfMA0GCSqGSIB3DQEBAQUAA4..."
18 },
19 {
20     "name": "LandPublicKey",
21     "type": "string",
22     "value": "ADCBiQKBgQDH+wPrKYG1KV1zQU..."
23 },
24 {
25     "name": "Value",
26     "type": "float",
27     "value": "40000"
28 }
29 ]
30 }

```

Listing 6.1 – TransferLand Transaction JSON

• Second phase: Hyperledger Fabric Implementation

The Hyperledger Fabric platform was used to recreate the same case study in the second phase. Within the developers' manual⁵ there is an example that involves the registration of automobiles, named Fabcar, implemented using Java language. This example has been modified to meet the requirements of the case study that consists of the record of land.

The code for this implementation can be accessed through the link: <<https://projotos.imd.ufrn.br/stefano/hyperledger-fabric-land-registry>>. The Listing 6.2 shows the Land class where land attributes are defined. In the implementation of Hyperledger Fabric in Java, markers are used to indicate that a class is a data type, using `@DataType()` (see line 1 of Listing 6.2), to be stored in the Blockchain and its attributes, using `@Property()` (see line 4 of Listing 6.2)

```

1 @DataType()
2 public final class Land {
3
4     @Property()
5     private final String status;
6
7     @Property()
8     private final float area;
9
10    @Property()
11    private final String address;
12

```

⁵ <https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html#example-ledger-fabcar>


```

13  @Property()
14  private final String owner;
15
16  /* get methods have been omitted */
17
18  public Land(@JsonProperty("status") final String
           status, @JsonProperty("area") final float area,
           @JsonProperty("address") final String
           address, @JsonProperty("owner") final String owner
           ) {
19      this.status = status;
20      this.area = area;
21      this.address = address;
22      this.owner = owner;
23  }
24}

```

Listing 6.2 – Land class

In Listing 6.3, it is possible to observe the definition of the *changeLandOwner* transaction. This transaction is defined in the *FabLand* class in which its methods are marked with *@Transaction()*. This transaction takes as a parameter the context of the ledger, the land key and the identifier of the new owner (see line 2 of Listing 6.3). With this information, it is possible to check the existence of the land (see line 4 of Listing 6.3) and if confirmed, changes the owner of the land (see line 15 of Listing 6.3)

```

1 @Transaction()
2 public Land changeLandOwner(final Context ctx, final
   String key, final String newOwner) {
3     ChaincodeStub stub = ctx.getStub();
4
5     String landState = stub.getStringState(key);
6
7     if (landState.isEmpty()) {
8         String errorMessage = String.format("Land_%s_
           does_not_exist", key);
9         System.out.println(errorMessage);
10        throw new ChaincodeException(errorMessage,
           FabLandErrors.LAND_NOT_FOUND.toString());
11    }
12
13    Land land = genson.deserialize(landState, Land.
        class);
14
15    Land newLand = new Land(land.getStatus(), land.
        getArea(), land.getAddress(), newOwner);
16    String newLandState = genson.serialize(newLand);
17    stub.putStringState(key, newLandState);
18
19    return newLand;
20 }

```

Listing 6.3 – Transaction changeLandOwner of FabLand class

Tests must be carried out under the same conditions on both platforms to maintain fairness between tests. For that, both solutions should make it possible to perform overload tests using the JMeter tool.

However, the Hyperledger Fabric platform currently does not automatically provide access to the creation of new transactions using the REST API. So, it was necessary to create a REST API web service for the case study to communicate with the Hyperledger Fabric platform since Orthus already provides this need automatically. This web service was done based on a tutorial on the website Medium⁶ that uses JavaScript to create web service.

- **Third phase: Version alignment**

This phase was implemented in order to make sure that both implementations were assured that implemented functionalities were the same. Several test cases were exhaustively used for all the releases of the Orthus and Hyperledger Fabric versions to ease the alignment process.

These alignment and validation steps were performed by the researcher. Their execution was also aimed at maximizing the equitability and fairness of the comparison between the alternative program versions. Inevitably, some minor refactorings in the versions of the target applications had to be performed when misalignments were observed during testing.

- **Fourth phase: Test environment definition**

The computational environment chosen to run the case study on both platforms was Google Cloud⁷ instance n1-standard-2 (2 vCPUs, 7,5 GB of memory). One of this instance for each platform was used to run the case study test. Both cases studies were implemented with tree peers each and made requests via the REST API using JMeter from an external computer to the cloud to capture the information previously described.

One instance of that cloud was used to deploy and simulate the different systems in a distributed way, see Figure 16, where Docker⁸ containers were used to isolate each node of the Orthus solution, as well as the Orion Context Broker.

Another instance of that same cloud configuration was used to deploy the Hyperledger Fabric case study implementation. Furthermore, it was also used Docker containers, see Figure 16, to isolate each Hyperledger Fabric node and the REST API web service used in the communication.

⁶ <https://medium.com/@kctheservant/an-implementation-of-api-server-for-hyperledger-fabric-network-8764c79f1a87>

⁷ <https://cloud.google.com/>

⁸ <https://www.docker.com/>

Figure 16 – Test environment

Source: Prepared by the author.

Apache JMeter⁹ was used to running outside the cloud the perform load testing for measuring and analyzing the performance of the solution for this case study. With it, some requisitions were made to simulate a high number of simultaneous requests.

- **Fifth phase: Performance test case definition.**

After the implementation of the same case study and the definition of the computational environment. Four test cases were defined, where the number of simultaneous users was changed. First test case with five users, then 50, 500, 5000, 10000 and ending with 15000 simultaneous users, each requesting in sequence ten land transfer transactions between two people. Thus, the number of executions for each test case is equal to the number of users multiplied by ten. In each test case, the shortest and longest time, average request time, standard deviation, error rate and throughput will be captured.

6.2 Study Results

After running the case study test for each of the platforms, as described above, shortest and most prolonged time, average request time, standard deviation, error rate and throughput for each case test was summarised in Table 5.

⁹ <https://jmeter.apache.org/>

Table 5 – Comparison Orthus vs Hyperledger Fabric

Users	Samples 10 x #Users	Response time (Millisecond)			Deviation	Error	Throughput
		Average	Min	Max			
Orthus							
5	50	12	2	36	9.7	0.0%	402.8/sec
50	500	45	2	210	36.43	0.0%	494.2/sec
500	5000	630	2	1970	303.72	0.0%	486.7/sec
5000	50000	1258	3	2586	813.24	0.0%	437.3/sec
10000	100000	1680	5	3153	1298.24	0.0%	377.8/sec
15000	150000	2974	18	4542	2239.11	2.5%	262.4/sec
Hyperledger Fabric							
5	50	15	5	87	23.0	0.0%	473.6/sec
50	500	67	7	286	47.32	0.0%	468.8/sec
500	5000	834	9	2262	412.17	0.0%	447.1/sec
5000	50000	1586	15	3487	1013.39	0.0%	403.4/sec
10000	100000	2134	21	4257	1876.71	0.0%	328.3/sec
15000	150000	x	x	x	x	x	x

According to Table 5, the throughput of Orthus the number remains between approximately 380 to 495 transactions per second. The throughput of five users test case of Orthus is lower than fifty simultaneous users test case because of a characteristic of the Orthus platform can dynamically increase the number of instances of its actors. As for the results of the tests using the Hyperledger Fabric platform, the throughput kept a decrease of approximately 470 in the test case with five simultaneous users to approximately 330 transactions per second in the last test case with 10000 concurrent users.

During the execution of the test cases with 15000 users, while the Orthus implementation managed to fulfil all the requests, but with a two and a half per cent error in the answers. The implementation using the Hyperledger Fabric did not support the number of requests. This case study implementation crashed the server in the cloud, making it impossible to complete the test.

Therefore, the maximum possible number of simultaneous transactions for the case study using Hyperledger Fabric is approximately 13500, since the last JMeter log was for user number 13426.

Next, the average response time and throughput alone will be analyzed.

6.2.1 Average Response Time

Figure 17 visually shows the difference in the average response time for each of the platforms where Orthus has a better average time in all test case. This result indicates that Orthus performed better in terms of the number of transactions performed per second

than the same case study test using Hyperledger Fabric in the same test case.

Figure 17 – Effect of number of users variation in the analysis of transaction execution by platforms upon average response time (milliseconds)

Source: Prepared by the author.

There is a very sharp growth in the average response time on both platforms between 5 and 500 simultaneous users. However, the growth becomes less pronounced after 500 users until the test of 10,000 users in both approaches.

6.2.2 Throughput

To facilitate the comparison, Figure 18 visually shows the difference in the throughput (transaction per second) for each of the platforms in the different test cases. It should be noted that Orthus' throughput for the test case with five users is lower than the test case with fifty users because the number of Akka actors is dynamically controlled according to the need.

Figure 18 – Effect of number of users variation in the analysis of transaction execution by platforms upon analysis throughput (transaction per second)

Source: Prepared by the author.

It is also possible to note that between 50 and 500 simultaneous users, as shown in this graph of Figure 18, the throughput on the Orthus platform shows a small decrease while on the Fabric platform, this decrease is much higher compared to Orthus. Moreover, after 500 users, the degree of decline is practically the same on both platforms, but Orthus' throughput remains higher than Hyperledger Fabric.

6.3 Threats to Validity

The conclusions obtained in this study are restricted to the involved metrics and the target software systems. Hence, the obtained results regarding the advantages and drawbacks of Orthus should not be directly generalized to other contexts. However, this study allowed us to make a useful evaluation of whether the utilization of Orthus could provide a way to share information with security, reliability, and privacy in addition to enabling interoperability between some systems.

Another criticism of this study may be that the chosen test environment uses Docker to simulate isolation between the platform nodes. However, in this way, latency in communication between the nodes was disregarded since the platforms use different methods of communication.

7 Conclusion

This chapter presents the conclusion of this master dissertation and is divided into contributions, limitations and future works. Section 7.1 presents the points that lead this work to contribute to state of the art. Section 7.2 describes the main limitations of this work. In Section 7.3 presents threats to validity. Section 7.4 presents the limitations of Orthus platform and describes the possible solutions for these limitations as future works.

7.1 Contribution

Blockchain technology has been widely used in the context of cryptocurrency as it enables bank transactions to be carried out without bank control. However, it can also be used in other contexts to creates a trusted entity with the potential to replace transaction certifying and centralizing entities such as governments, notaries.

With the advance of technology, most municipal public services already use some Management and Information System to store their data and, in some cases, provide some service to the population, what gave rise to the term Smart Cities. However, integration between different data sources is often necessary to provide new services to the population where it can only be achieved by integrating these systems. The main problem with integrating data and services is the necessary trust between them.

The use of Blockchain technology has much to contribute to integrating systems in the Smart City context. Once it enables the creation of solutions in distributed networks being able to share data and services in a transparent, secure and reliable way without centralizing entities, enabling the integration of city services to improve the quality of life of the population, in this context, this work presented how Blockchain can be used in Smart Cities environments, through the definition and implementation of the Orthus platform. Orthus enables interoperability by securely exchanging data between many Information Systems.

The main contributions of this work are fourfold: (i) a systematic review of the use of Blockchain in the context of Smart Cities; (ii) a Blockchain platform for Smart Cities named Orthus; (iii) a land registry case study using Orthus; and (iv) a comparative study between Orthus and Hyperledger Fabric. Each one of these contributions is described in the following.

A systematic review of the use of Blockchain in the context of Smart Cities. Chapter 3.1 presented the first main contribution of this work, a systematic review that elicited a set of requirements needed to uses Blockchain in Smart Cities context and discover the leading platforms used in this context.

A Blockchain platform for Smart Cities named Orthus. As introduced

in Chapter 4, Orthus platform is described. Orthus is proposed for share and storing transactions of Management Information Systems in a distributed and secure way to support the amount of data generated from city devices in Smart Cities context where these transactions can meet any data type with velocity and veracity. In addition to presenting the platform architecture decoupled from specific technologies, it presents an example of an implementation using technologies focused on the Big Data context. It is emphasizing that these technologies can be replaced by others that meet the architectural description of the platform.

A land registry case study using Orthus. The Orthus platform was used to develop a case study involving land registration in a city to prove its applicability. The transfer of land between people of this city involving three government agencies was simulated.

A comparative test between Orthus and Hyperledger Fabric. At last, the fourth contribution of this work, as described in Chapter 6, The same case study was performed in the same computing environment reasonably to be able to compare the Orthus platform with Hyperledger Fabric. Average response times and transactions per second were compared.

7.2 Limitations

Although the good results of Orthus in comparison with Hyperledger, Orthus still has some limitations. In the following list, some of these limitations will be highlighted:

- The tests performed were not exhaustive and cannot assess the performance and load characteristics of the tool. It may be that the platform has limitations that were not identified in the evaluated case study.
- The difficulty to implement case studies using Orthus and Hyperledger Fabric platforms was not compared.

After looking at the platform's contributions and limitations, the next sections of this chapter will present the threats encountered and what needs to be worked out in the future to improve Orthus.

7.3 Threats to Validity

As the main threat to the validity of the process, one can cite the fact that the simulation of the systems of the city hall and the notary office was part of the case studies used to validate the Orthus platform.

Besides, the entire case study test environment was on the same cloud. In a real application, each system would be under the responsibility of different organs and consequently, in different computational environments.

7.4 Future Works

The current limitations on Orthus implementation can be resolved with the following future work. Namely:

Smart Contract Addition. In the current implementation of Orthus, there is no use of Smart Contracts. Smart Contracts execution must be performed by the systems that make up Orthus nodes, thus enabling interoperability between systems as described in the platform architecture.

Orion network. Orthus currently uses only one instance of Orion Context Broker. As described in the architecture, it is necessary to use an Orions network to ensure communication between organizations and to avoid having only one way of communication between nodes.

Node Management Engine. Because Orthus was designed to be a private Blockchain platform, a management mechanism is required to control the addition of new nodes to the platform. Thus, as future work, it is necessary to develop Node Management Engine.

Byzantine Fault Tolerance as consensus algorithms. In addition to the creation of the Node Management Engine, a voting mechanism needs to be implemented to be able to use Byzantine Fault Tolerance as a consensus mechanism and to obtain a better transaction rate per second.

Bibliography

ABBASI, A. G.; KHAN, Z. Veidblock: Verifiable identity using blockchain and ledger in a software defined network. In: ACM. *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. [S.l.], 2017. p. 173–179. Cited 2 times on pages 48 and 49.

AIXA, D. R. Analysis and study of data security in the internet of things paradigm from a blockchain technology approach. Universitat Oberta de Catalunya, 2018. Cited 2 times on pages 23 and 30.

ALVES, G.; CAVALCANTE, E.; BATISTA, T. On the use of new blockchain-based technologies for securely distributing data. In: SBC. *SBSeg 2018*. [S.l.], 2018. p. 81–88. Cited 4 times on pages 25, 31, 33, and 34.

ANDROULAKI, E. et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: *Proceedings of the Thirteenth EuroSys Conference*. [S.l.: s.n.], 2018. p. 1–15. Cited on page 73.

APACHE. *Spark Documentation*. 2019. Available at: <<https://spark.apache.org/docs/latest/>>. Accessed on: Jul. 2019. Cited on page 41.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. Cited 2 times on pages 48 and 49.

BACK, A. et al. Hashcash-a denial of service counter-measure. 2002. Cited on page 32.

BLAIR, G. et al. Holons: Towards a systematic approach to composing systems of systems. In: ACM. *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware*. [S.l.], 2015. p. 5. Cited on page 35.

BRANDÃO, A. et al. A smart city's model secured by blockchain. In: SPRINGER. *International Conference on Software Process Improvement*. [S.l.], 2018. p. 249–260. Cited on page 48.

BRITO, J.; CASTILLO, A. *Bitcoin: A primer for policymakers*. [S.l.]: Mercatus Center at George Mason University, 2013. Cited 2 times on pages 23 and 24.

BRODERSEN, C. et al. *Blockchain: Securing a New Health Interoperability Experience*. [S.l.]: Accenture LLP, 2016. Cited 2 times on pages 47 and 52.

BROWNSWORD, L. et al. *System-of-systems navigator: An approach for managing system-of-systems interoperability*. [S.l.], 2006. Cited on page 24.

CHANSON, M. et al. Blockchain as a privacy enabler: an odometer fraud prevention system. In: ACM. *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*. [S.l.], 2017. p. 13–16. Cited on page 47.

CHAUM, D. Blind signatures for untraceable payments. In: SPRINGER. *Advances in cryptology*. [S.l.], 1983. p. 199–203. Cited on page 23.

CHEN, G. et al. Exploring blockchain technology and its potential applications for education. *Smart Learning Environments*, SpringerOpen, v. 5, n. 1, p. 1, 2018. Cited 4 times on pages 48, 49, 53, and 54.

COETZEE, L.; EKSTEEN, J. Internet of things—promise for the future? an introduction. 2011. Cited on page 36.

DAHRI, K. et al. Interoperable health care system using blockchain technology. *Sindh University Research Journal-SURJ (Science Series)*, v. 51, n. 3, p. 437–440, 2019. Cited on page 48.

EKBLAW, A. et al. A case study for blockchain in healthcare: “medrec” prototype for electronic health records and medical research data. In: *Proceedings of IEEE Open & Big Data Conference*. [S.l.: s.n.], 2016. v. 13, p. 13. Cited 2 times on pages 47 and 48.

FIWARE. *FIWARE-ORION Documentation*. 2018. Available at: <<https://fiware-orion.readthedocs.io/en/master/>>. Accessed on: Jul. 2019. Cited on page 42.

GAMA, K.; ALVARO, A.; PEIXOTO, E. Em direção a um modelo de maturidade tecnológica para cidades inteligentes. *Simpósio Brasileiro de Sistemas de Informação, VIII*, 2012. Cited on page 35.

GRIGG, I. Eos-an introduction. (*Whitepaper*) iang.org/papers/EOS_An_Introduction.pdf, 2017. Cited on page 54.

HEIJDEN, R. W. van der et al. Blockchain: Scalability for resource-constrained accountable vehicle-to-x communication. *arXiv preprint arXiv:1710.08891*, 2017. Cited on page 47.

IVAN, D. Moving toward a blockchain-based method for the secure storage of patient records. In: *ONC/NIST Use of Blockchain for Healthcare and Research Workshop*. Gaithersburg, Maryland, United States: ONC/NIST. [S.l.: s.n.], 2016. Cited on page 48.

KAKAVAND, H.; SEVRES, N. K. D.; CHILTON, B. The blockchain revolution: An analysis of regulation and technology related to distributed ledger technologies. 2017. Cited on page 49.

KANTER, R. M.; LITOW, S. S. Informed and interconnected: A manifesto for smarter cities. 2009. Cited 2 times on pages 23 and 34.

KINAI, A. et al. Asset-based lending via a secure distributed platform. In: *ACM. Proceedings of the Ninth International Conference on Information and Communication Technologies and Development*. [S.l.], 2017. p. 29. Cited on page 47.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Cited on page 45.

KORPELA, K. et al. Digital supply chain transformation toward blockchain integration. In: *proceedings of the 50th Hawaii international conference on system sciences*. [S.l.: s.n.], 2017. Cited on page 48.

KUHN, R.; HANAFEE, B.; ALLEN, J. *Reactive design patterns*. [S.l.]: Manning Publications Co., 2017. Cited 2 times on pages 37 and 65.

- LI, N. *The Difference between Blockchain and a Distributed Database*. 2018. Available at: <<https://medium.com/ultrain-chain/the-difference-between-blockchain-and-a-distributed-database-556f8361e6b3>>. Accessed on: 12 dez. 2019. Cited on page 30.
- MATTILA, J. *The blockchain phenomenon—the disruptive potential of distributed consensus architectures*. [S.l.], 2016. Cited on page 32.
- MONTES, J. M. et al. Applying blockchain to supply chain operations at ibm implementing agile practices in a smart city environment. *Research in Computing Science*, v. 147, p. 65–75, 2018. Cited on page 48.
- NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. 2008. Cited 2 times on pages 30 and 32.
- NARAYANAN, A. et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. [S.l.]: Princeton University Press, 2016. Cited on page 23.
- NEO White Paper. Available at <https://docs.neo.org/en-us/whitepaper.html>, 2018. Cited on page 54.
- NICHOL, P. B.; PMP, C.; CMPI, C. Co-creation of trust for healthcare: The cryptocitizen framework for interoperability with blockchain. 2016. Cited 3 times on pages 47, 52, and 54.
- ØLNES, S.; UBACHT, J.; JANSSEN, M. *Blockchain in government: Benefits and implications of distributed ledger technology for information sharing*. [S.l.]: Elsevier, 2017. Cited on page 48.
- ÖZSU, M. T.; VALDURIEZ, P. *Principles of distributed database systems*. [S.l.]: Springer Science & Business Media, 2011. Cited on page 29.
- PATGIRI, R.; AHMED, A. Big data: The v's of the game changer paradigm. In: IEEE. *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. [S.l.], 2016. p. 17–24. Cited on page 36.
- PETERSON, K. et al. A blockchain-based approach to health information exchange networks. In: *Proc. NIST Workshop Blockchain Healthcare*. [S.l.: s.n.], 2016. v. 1, p. 1–10. Cited 2 times on pages 47 and 48.
- PIERONI, A. et al. Smarter city: smart energy grid based on blockchain technology. *Int. J. Adv. Sci. Eng. Inf. Technol*, v. 8, n. 1, p. 298–306, 2018. Cited on page 48.
- PINNA, A.; IBBA, S. A blockchain-based decentralized system for proper handling of temporary employment contracts. *arXiv preprint arXiv:1711.09758*, 2017. Cited 3 times on pages 47, 48, and 52.
- ROTUNA, C. et al. Smart city ecosystem using blockchain technology. *Informatica Economica*, Academy of Economic Studies-Bucharest, Romania, v. 23, n. 4, p. 41–50, 2019. Cited on page 48.

ROWAN, S. et al. Securing vehicle to vehicle communications using blockchain through visible light and acoustic side-channels. *arXiv preprint arXiv:1704.02553*, 2017. Cited 2 times on pages 48 and 53.

SAMADI, Y.; ZBAKH, M.; TADONKI, C. Comparative study between hadoop and spark based on hibenx benchmarks. In: IEEE. *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*. [S.l.], 2016. p. 267–275. Cited on page 65.

SHARMA, P. K.; MOON, S. Y.; PARK, J. H. Block-vn: A distributed blockchain based vehicular network architecture in smart city. *Journal of Information Processing Systems*, v. 13, n. 1, p. 84, 2017. Cited 3 times on pages 47, 50, and 54.

SINGHAL, B.; DHAMEJA, G.; PANDA, P. S. *Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions*. [S.l.]: Springer, 2018. Cited 3 times on pages 33, 34, and 50.

SUN, J.; YAN, J.; ZHANG, K. Z. Blockchain-based sharing services: What blockchain technology can contribute to smart cities. *Financial Innovation*, Springer, v. 2, n. 1, p. 26, 2016. Cited on page 24.

SWAN, M. Connected car: quantified self becomes quantified car. *Journal of Sensor and Actuator Networks*, Multidisciplinary Digital Publishing Institute, v. 4, n. 1, p. 2–29, 2015. Cited 3 times on pages 25, 31, and 48.

VANTIEGHEM, C. *Monetary practices In ancient Egypt*. 2012. Available at: <http://www.nbbmuseum.be/en/2012/05/nederlands-geldgebruik-in-het-oude-egypte.htm>. Accessed on: 19 April 2018. Cited on page 23.

WALKER, M. A. et al. Platibart: a platform for transactive iot blockchain applications with repeatable testing. In: ACM. *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things*. [S.l.], 2017. p. 17–22. Cited on page 48.

WHITE, S. A. Introduction to bpmn. *Ibm Cooperation*, v. 2, n. 0, p. 0, 2004. Cited on page 60.

ZHANG, J.; XUE, N.; HUANG, X. A secure system for pervasive social network-based healthcare. *IEEE Access*, IEEE, v. 4, p. 9239–9250, 2016. Cited on page 47.

ZHANG, P. et al. Applying software patterns to address interoperability in blockchain-based healthcare apps. *arXiv preprint arXiv:1706.03700*, 2017. Cited 2 times on pages 47 and 50.

APPENDIX A – List of Publications

A.1 Publications resulted from this work

Stefano Loss, Nélío Cacho, Frederico Lopes, João Marcos do Valle. Orthus: A Blockchain Platform for Smart Cities. Best Paper Award in: 5th IEEE International Smart Cities Conference (IEEE ISC2 2019), 2019, Casablanca/Morocco. Proceedings of IEEE International Smart Cities Conference ISC2 2019, 2019.