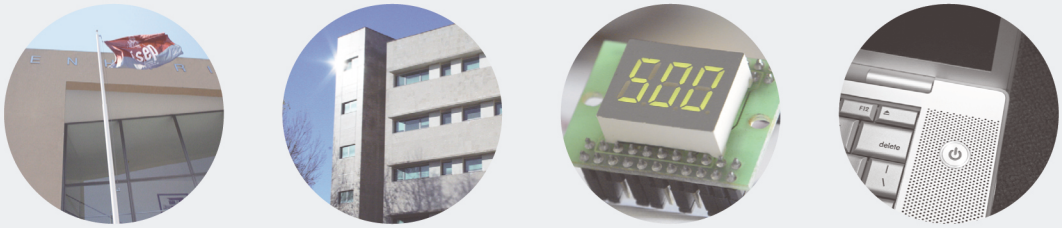
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



CloudAnchor Smart Contracts

EDUARDO HENRIQUE CARPINTEIRO VASCO
novembro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4249-015 Porto

CloudAnchor Smart Contracts

Master in Electrical Engineering and Computers

Eduardo Henrique Carpinteiro Vasco

Supervisor: Prof. Maria Benedita Campos Neves Malheiro

Academic Year: 2021-2022

Abstract

The CloudAnchor platform allows the negotiation of IaaS Cloud resources for Small and Medium Sized Enterprises (SME), either as resource providers or consumers. This project entails the research, design, and implementation of a solution based on *smart contracts*, with the goal of permanently recording and managing the contracts on a blockchain network. The usage of *smart contracts* enables safe contract code execution and raises the level of trust, integrity, and traceability of the platform contracts by keeping the data stored in a decentralised manner. To do so, a method to coordinate and submit transactions to the blockchain network must be implemented. The tests carried out indicate that the solution has been successfully implemented, with contract registration saved in a decentralised and safe manner. As a result, there was an increase in the platform's execution time, caused by the new transactions made to the blockchain.

Keywords: Blockchain, CloudAnchor, smart contracts, decentralisation, traceability;

Resumo

A plataforma CloudAnchor permite a negociação e contratualização de recursos Cloud do tipo IaaS a pequenas e médias empresas, sejam elas fornecedoras ou clientes. Este trabalho inclui o estudo, projeto e implementação de uma solução baseada em *smart contracts*, com o objetivo de administrar e registar de forma permanente os contratos celebrados numa rede blockchain. A utilização de *smart contracts* permite executar o respetivo código de forma segura e aumentar o nível de confiança, integridade e rastreabilidade dos contratos celebrados na plataforma, guardando-os de forma descentralizada. Para tal, é necessário implementar um mecanismo de coordenação e submissão de transações para a rede blockchain. Os testes realizados permitiram concluir que a implementação da solução foi bem sucedida, passando os contratos a ficar guardados de forma descentralizada e segura. Em consequência, verificou-se um aumento do tempo de execução da plataforma provocado pelas novas transações com a blockchain.

Palavras-chave: Blockchain, CloudAnchor, smart contracts, descentralização, rastreabilidade;

Contents

List of Figures

List of Tables

Abbreviations

ABI Application Binary Interface. 51

ACL Agent Communication Language. 15, 16

AI Artificial Intelligence. 7, 8

AOP Agent-Oriented Programming. 8

API Application Programming Interface. v–vii, 39, 40, 48, 51, 52, 57, 59–61, 63, 66, 68, 69, 72–75, 78, 82

ASES Adaptive Systems and Evolutionary Software. 18

AuRa Authority Round. 30

B2A Business-to-Administration. 35, 36, 42

B2B Business-to-Business. 1, 3, 33, 35, 38, 42

B2C Business-to-Consumer. 35, 36, 42

BDI Belief Desire Intention. 11, 12

BFT Byzantine Fault Tolerance. 29, 50, 52

bSLA Brokerage Service Level Agreement. 38, 40, 55–57, 60, 63, 67, 70–74, 76

C2A Consumer-to-Administration. 35, 36, 42

CA Certificate Authority. 58, 63–65

CAS Contractual Agent Societies. 18

CBP Collaborative Business Processes. 37

CFP Call For Proposal. 18, 19

CFT Crash Fault Tolerance. 49, 50, 52

CIA Cooperative Information Agent. 18

CLI Command Line Interface. 66

CNIP Contract Net Interaction Protocol. 17–19, 41

CPU Central Process Unit. 75, 78–80

CRUD Create, Read, Update and Delete. 60

cSLA Coalition Service Level Agreement. 38, 40, 55–57, 63, 67

CVM Chain Virtual Machine. 27

DAG Directed Acyclic Graph. 23, 47

DAI Distributed Artificial Intelligence. v, 7, 8

Dapp Decentralised Application. 30, 32, 33

DLT Distributed Ledger Technology. v, 23, 44, 48

DPoS Delegated Proof of Stake. 30, 45, 46

DPS Distributed Problem Solving. 7

EEA Enterprise Ethereum Alliance. 47, 48, 52

EVM Ethereum Virtual Machine. 27

FICNIP Fixed Iterated Contract Net Interaction Protocol. 17, 19, 41

FIPA Foundation of Intelligent Physical Agents. 15, 17

Geth Go Ethereum. 47, 51, 52

GNU GNU's Not Unix. 47

HLP Hyperledger Project. 44, 48

HTTP Hypertext Transfer Protocol. 48, 59, 60

IaaS Infrastructure as a Service. iii, v, 1, 33, 38, 39

IBFT Istanbul Byzantine Fault Tolerance. 48

IBM International Business Machines. 21

ICNIP Iterated Contract Net Interaction Protocol. 17, 19, 41

IEEE Institute of Electrical and Electronics Engineers. 15

IS Information Systems. 15

IT Information Technology. 37

JADE Java Agent Development Framework. 15

JAR Java Archive. 45

JSON JavaScript Object Notation. 48, 51

KIF Knowledge Interchange Format. 15

KPI Key Performance Indicator. 20–22, 75, 78

KQML Knowledge Query and Manipulation Language. 15

KSE Knowledge Sharing Effort. 15

LGPL Lesser General Public License. 47

MAS Multi-Agent System. 5, 7–10, 33, 37, 42

MSC Message Segquence Chart. 18

MSP Membership Service Provider. 63

OGF Open Grid Forum. 21

P2P Peer-to-Peer. 6, 36, 37

PAIS Process-Aware Information System. 37

PBFT Practical Byzantine Fault Tolerance. 29, 44

PoA Proof of Authority. 29, 30, 44, 47, 48

PoS Proof of Stake. 29, 30

PoW Proof of Work. 29, 30, 43, 44, 47, 48

PRS Procedural Reasoning System. v, 12, 13

QoS Quality of Service. 22

RAM Random Access Memory. 75

RPC Remote Procedure Call. 48, 51

rSLA Resource Service Level Agreement. 38, 40, 55–57, 59, 60, 63, 67, 68, 70–73, 79

RTO Return to Operation. 20

SCM Smart Cloud Marketplace. 37

SDK Software Development Kit. 49, 51–53, 60

SLA Service Level Agreement. 20–22, 32, 33, 38–41, 51, 55, 68, 71, 75, 82

SLO Service Level Objective. 20, 22

SME Small and Medium Sized Enterprises. iii, 1, 38

T&R Trust and Reputation. 42, 55, 76

TTP Trusted Third Party. 21, 22, 31, 32

UID Unique Identifier. 26

VE Virtual Enterprise. 37

VM Virtual Machine. 75, 78, 79

WASM Web Assembly. 45

WS Web Services. 21

WS-Agreement Web Service Agreement. 21

WSLA Web Service Level Agreement. 21

XML eXtensible Markup Language. 21

Acknowledgements

First and foremost, I want to thank Professor Maria Benedita Malheiro and Professor Bruno Veloso for coordinating and guiding me through this project. Professor Maria Benedita Malheiro instilled in me a rational way of thinking throughout the duration of this research, for which I am grateful. Professor Bruno was also very helpful in understanding the platform implementation as a past and present CloudAnchor platform maintainer.

Second, I'd like to express my gratitude to my family, girlfriend, and friends for their unwavering support throughout this journey. This was a long run that kept me occupied for many hours while also causing me to spend less time with them. However, it now appears to have been a demonstration of will, particularly during pandemics that required us to re-adapt most of our working patterns.

Finally, I had a great time performing this project; it completed the gap in my graduation, and now it's time to rethink the next challenge.

Thank you sincerely.

Chapter 1

Introduction

The current chapter introduces the dissertation background, the problem to be solved, the expected outcomes, the execution strategy, and the structure of the dissertation document.

1.1 Context

The investigation, design, and implementation of a blockchain-based solution are detailed in this thesis. Its goal is to keep track of all the negotiated contracts that have been set up on the CloudAnchor platform.

CloudAnchor is a Business-to-Business (B2B) brokerage platform that enables SME to transact IaaS resources in the Cloud. The use of a smart contract solution is expected to improve the platform's data integrity and robustness.

1.2 Motivation

Blockchain is not a new concept. It is now an emerging technology that is revolutionising the way organizations work. It has potential uses in a variety of markets, thus the possibilities are endless. As a result, having the opportunity to learn and explore such technologies is exciting.

1.3 Objectives

The purpose of this project is to improve the CloudAnchor platform with the smart contract technology and assess its impact in the platform's execution. This way, all contracts created on the platform are not only retained on the platform, but are also stored and performed on the blockchain. The following procedures

were taken to attain this goal: (i) study of the blockchain technology; (ii) selection of the blockchain solution; (iii) implementation of the solution; (iv) integration with CloudAnchor; and (v) impact assessment.

1.3.1 Requirements

The following elements must be considered while choosing a blockchain platform and designing a solution:

- The blockchain must be an open-source technology;
- The blockchain must be implemented as a private network;
- It must have smart contracts support;
- It must have a reasonable execution time;
- It must have available libraries for client development.

1.4 Work Plan

The following were established as the work's objectives:

1. Study the theoretical topics and write the state of the art chapter;
2. Study CloudAnchor platform business structure and features;
3. Search and compare different blockchain platforms and clients;
4. Design and implement a solution;
5. Test and validate the developed solution;
6. Write the dissertation.

Below is the chronological plan (Gantt chart) of the accomplished tasks.

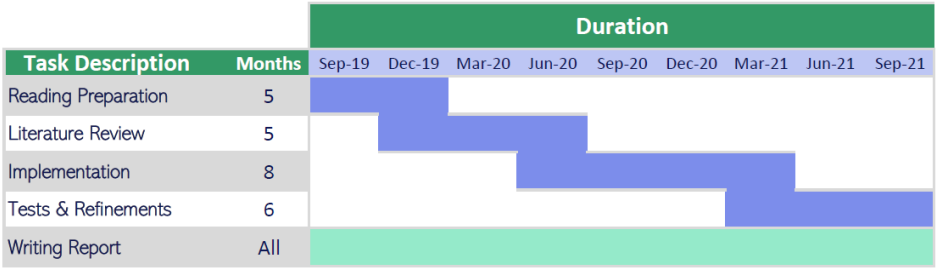


Figure 1.1: Gantt diagram.

1.5 Structure of the Dissertation

This document is divided in eight chapters.

1. Chapter I presents and contextualises the project theme by emphasising motivation, outlining the primary objectives, defining the solution requirements, functional testing, and work schedule, and detailing the dissertation structure.
2. Chapter II is devoted to the state-of-the-art section. It covers the fundamentals of the CloudAnchor platform technologies as well as blockchain technology.
3. Chapter III introduces Business-to-Business (B2B) e-Commerce, presents examples of B2B platforms, and finally describes the CloudAnchor platform as well as its functionalities.
4. Chapter IV introduces and compares a number of blockchain platforms and clients for use in deploying the solution.
5. Chapter V presents the solution proposal.
6. Chapter VI describes the implemented solution.
7. Chapter VII explains the experiments that were carried out and analyzes the outcomes.
8. Finally, chapter VIII draws the conclusions and makes recommendations for future work.

Chapter 2

State of The Art

The literature review topics for the dissertation are covered in this section.

2.1 Distributed Decentralised Systems

Because the blockchain is highly decentralized and distributed, and the CloudAnchor platform is built on a Multi-Agent System (MAS) architecture, understanding the foundations of distributed systems, control delegation, and distributed artificial intelligence is essential. The concepts relevant to these issues are presented in this section.

2.1.1 Distribution and Decentralisation

A distributed system is a group of modules or machines working together as to appear as a single system to the end-user. Working modules, often known as machines, are a group of autonomous entities that interact with one another to solve a problem that cannot be handled individually [Kshemkalyani and Singhal, 2011]. The primary concept of decentralisation is to deliberately disperse power and authority to the organisation's peripheries rather than having one central body in complete control [Bashir, 2018]. The amount of authority delegated to the lowest level will determine the distance between a centralised and a decentralized system. While centralised systems may have aided in the development of the Internet, they have significant drawbacks. In a centralised system, such as the client-server topology, where clients approach the server for resources, the authority is generally managed at a single point [Infante, 2019]. Users of the system are reliant on a single source of service due to this structure. In a decentralized system, on the other hand, data and computation are distributed among several nodes in the network, removing node dependency. The key distinction between

a distributed and a decentralised system is that in distributed systems, a central authority still orchestrates the entire system, whereas in a decentralised system, everything is decentralized (Figure 2.1).

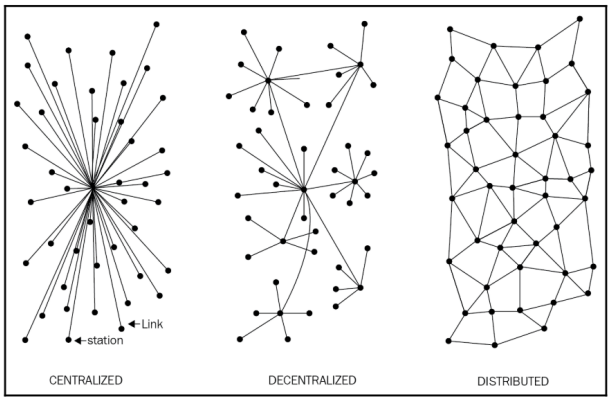


Figure 2.1: Types of Networks [Bashir, 2018].

2.1.2 Network Architecture Models

The term computer network defines the category in which a group of computers is connected [Sandeep, 2013]. To function as a system, a set of computers must each play a unique role in the network. To organise a computers network there are different well-known models which can be used (Figure 2.2), namely the client/server model, the multi-tier model, and the Peer-to-Peer (P2P) model. The P2P paradigm and the traditional client/server approach are frequently compared, and their architectures are frequently contrasted. In client/server architectures, certain computers are completely dedicated to serving others; however, in a P2P paradigm, there are no pre-conceived roles because each peer can act as both a server and a client, and all network transactions are carried out directly between peers. Because all processors play a symmetric part in the computation, there is no hierarchy among them in P2P. Its key advantage is that there is no single point of failure, thus even if one node fails, the system does not come to a standstill. According to Kshemkalyani and Singhal [2011], the P2P network model offers the following advantages:

- Tolerance to Faults/Attacks - When a portion of a network's computers is attacked, the full service is not disrupted. All of the network nodes must collapse at the same time for this to happen.
- Load Balancing - Each host is responsible for a modest portion of the system's load, ensuring that the system does not overflow. If the system receives too many requests at once, the load is balanced by distributing the requests across the available nodes.

- Cost Efficiency - The unified network can supply system services, eliminating the need for huge servers with high bandwidth delivery, which can be costly.

- Security - It is more difficult to bring down a decentralized system with several nodes. The system becomes more secure as the number of nodes increases.

The multi-tier model is a client-server architecture design that separates and organises the user interface, the functional logic, and the computer database.

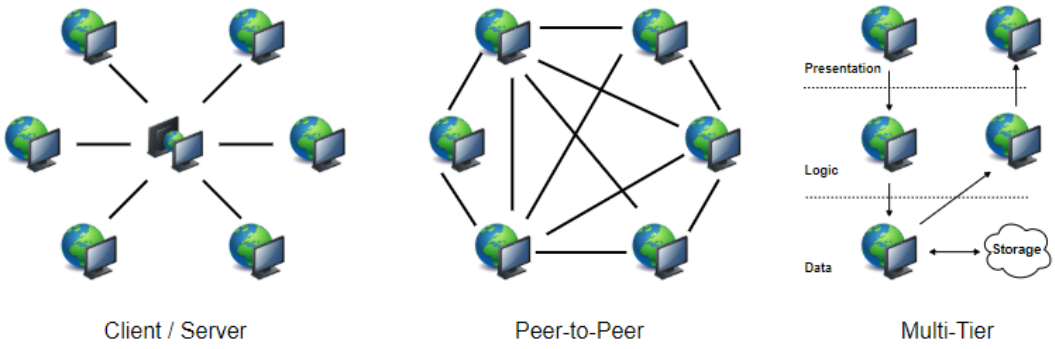


Figure 2.2: Network Models [Bouchrika, 2013]

2.1.3 Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI) is a sub-field of AI that has grown at an exponential rate in terms of research and development [Weiss, 1999]. DAI systems are made up of self-learning processing nodes that are especially interested in investigating difficult issue areas in systems with several independent entities interacting with one another [Barr and Feigenbaum, 1981]. DAI is the predecessor of two areas of research (Figure 2.3): Distributed Problem Solving (DPS) and Multi-Agent System. The primary distinction is in the system's goal. MAS systems are developed to coordinate the intelligent behaviour of a group of autonomous agents, whereas DPS systems are designed for a specific task [Bond and Gasser, 1988].

In [Durfee, 2001], DPS entails combining the expertise, information, and capacities of several problem solvers to generate answers to issues that none of them could solve as well on their own. Agents in MAS systems reason about processes. They are self-sufficient and work together to coordinate their knowledge and activities. The interactions of computational intelligent agents are of primary interest to researchers in the field of MAS [Weiss, 1999].

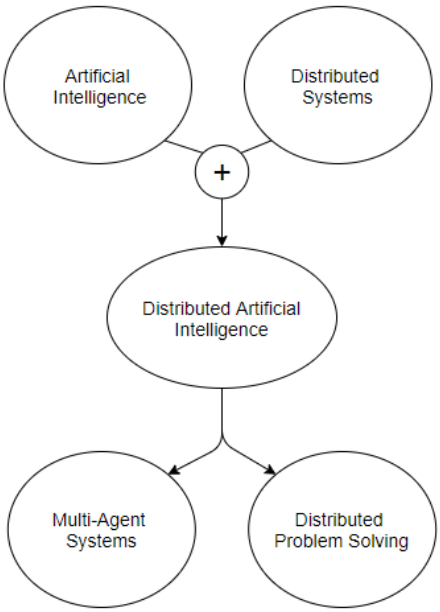


Figure 2.3: DAI Hierarchy Chart [Reis, 2003].

2.2 Multi-Agent Systems

Multi-Agent System automation became well-known in the 1990s for its ability to conceptualise, model, and build distributed solutions for a wide range of applications. For agent systems to perform searching and filtering over massive amounts of data, distributed and open platforms, such as the Internet, are particularly appealing [Decker and Sycara, 1997]. According to Jennings et al. [1998], is a loosely coupled network of problem solvers that interact to solve challenges beyond the individual capability of each problem solver, commonly referred to as agents. Agents are self-contained software components that can take activities on behalf of their owners. The design and coordination of different agents contribute to the difficulty of administering a MAS [Avancini and Amandi, 2000]. The multi-agent paradigm is well suited to solving problems that are intrinsically distributed. The systems' interoperability and connectivity result in solutions that can be organically organised in numerous linked tiers. Some of these capacities are one of the underlying reasons for creating problem-solving systems in organisations [Durfee et al., 1989]. Entities can be represented by software agents, which are in charge of defending their interests and resolving any potential conflicts. According to Sycara [1998], MAS can boost a system performance in multiple conditions, such as: (i) computational efficiency, (ii) reliability, (iii) extensibility, (iv) robustness, (v) maintainability, (vi) responsiveness, (vii) flexibility, and (viii) re-usability. Agent-Oriented Programming (AOP) is a software paradigm derived from AI that applies some of those notions to the well-known

domain of distributed systems. Depending on the designated duty of the agent, the term "agent" may be referred to by several terms. As reported by Nwana [1996], knowbots (knowledge-based robots), softbots (software robots), taskbots (task-based robots), userbots, robots, personal agents, autonomous agents, and personal assistants are all terms used to describe agents. Bellifemine et al. [2007] presents the following MAS technology applications in industry:

- Traffic and Transportation - Because of the scattered structure of traffic and transportation systems, MAS is a useful tool for implementing practical commercial solutions.
- Telecommunications - These are vast, distributed networks of interconnected components that require real-time monitoring and management. As a result, MAS may be used to manage such remote networks as well as to provide enhanced communications services.
- Multi-robotics - For coordination among the many robots, multi-robotic systems can also leverage multi-agent and distributed planning techniques. To allocate duties among robots, the top planning layer employs a market-based technique.
- Health Care - In health care, there are some interesting MAS applications. Medical decision support systems [Hudson and Cohen, 1999] can benefit from MAS applications, which can increase coordination between the various specialists involved in the health-care process [Lanzola and Boley, 2002].

2.2.1 System Modelling

The criteria for modelling a system are always influenced by the problem's nature. Agents must collaborate when interdependent situations develop to guarantee that the inter dependencies are correctly managed [Sycara, 1998]. The goal of organisational analysis and planning is to capture how organizations work and grow, as well as to design a framework for them. These strategies can help with agent modelling and can help with model design efficiency. When modelling a system, according to Dignum and Dignum [2012], the following relationships and aspects must be considered:

1. Represent ideas about agent capability and activity.
2. Accept that the agent's capabilities are limited.
3. Relate roles and agents.
4. Deal with temporal issues such as the time the activities will take.
5. Represent global objectives and their dependency on agents activity.

6. Reason about organisational workflows and task dependencies.
7. Represent organisational dynamics.
8. Deal with normative issues (boundaries for action and violations).

To avoid wasting resources, the profile of each agent (Figure 2.4) must be developed after determining the system's model and the types of agents that will be required. Below are some of those profile properties that, according to Nwana [1996] and Omicini and Poggi [2006], help modelling agents:

- Sociality - Agents cooperate with humans or other agents in order to achieve their goals.
- Autonomy - Agents have the autonomy to decide either to cooperate with other agents for mutual benefit or to compete against them for its own benefit.
- Reactiveness - Agents have the ability to perceive the environment and to respond at the right time to the changes that may occur in the environment.
- Proactiveness - Agents do not simply react to the environment but also have the intention to exhibit goal-directed behaviour by taking initiative.
- Mobility - Agents have the ability to travel to different computer nodes inside a MAS network.
- Rationality - Agents act in order to achieve their goals and never to prevent their goals from being achieved.

2.2.2 Agent Architecture

Architectures are the underlying processes that enable autonomous components to behave effectively in real-world, dynamic, and open contexts [Bellifemine et al., 2007]. A system can be designed in a variety of ways. The architecture of a system can influence its behaviour, which can have an impact on the system's output. According to Chin et al. [2014], when it comes to reasoning and performing knowledge-based actions, an agent architecture is critical. Because each individual agent may reason about non-local implications of local acts, create expectations of others' behaviour, explain and possibly repair disagreements and detrimental interactions, sophisticated individual agent reasoning might boost MAS coherence [Sycara, 1998].

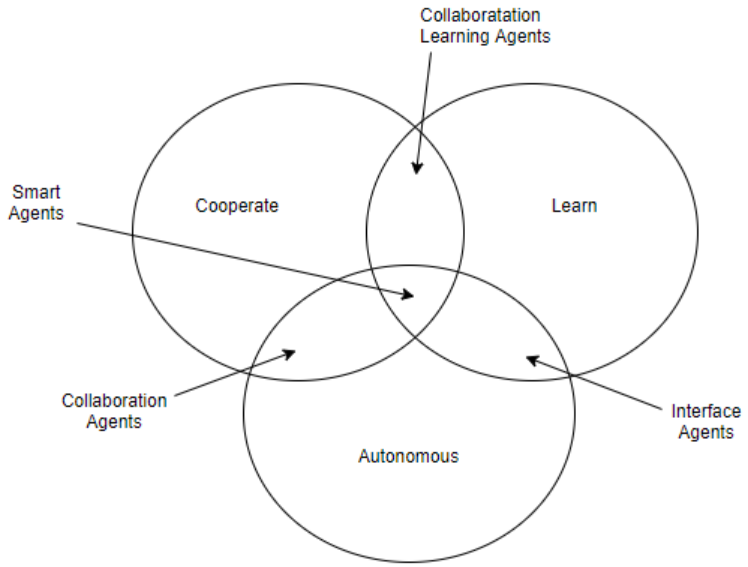


Figure 2.4: Agents Profile [Nwana, 1996].

There are three fundamental groups of agents architectures: classical architectures, cognitive architectures and semantic architectures. Classical architectures include logic-based architectures, reactive architectures, Belief Desire Intention (BDI) architectures and hybrid architectures, also known as layered architectures, which are a combination of logic-based and reactive architectures. Cognitive architectures are based on cognitive sciences, which attempt to study human behaviour in order to create software agents that mimic it. Semantic architectures are based on semantic technologies. Semantic technologies improve decision-making by processing data in a more meaningful way by emulating human reasoning.

2.2.2.1 Logic-based

Logic-based architectures use symbolic representation to express operations and follow a reasoning-type framework. The alteration of the symbolic representation underpins the agent's behaviour. The advantage of this technique is that, because human knowledge is symbolic, humans can understand the logic more easily [Bellifemine et al., 2007]. However, translating the physical world into an accurate symbolic description can be challenging at times, and symbolic representation and manipulation might take a long time to complete, delaying the results.

2.2.2.2 Reactive

Unlike logic-based designs, reactive architectures do not employ complicated symbolic reasoning. They are based on direct stimulus-response mechanisms triggered

by sensor data [Chin et al., 2014]. One of the most recognised reactive architectures is Brook's subsumption architecture [Brooks, 1986]. The agents in this design do not plan; instead, they react to conditions in their surroundings. The advantage is that they will respond more quickly, but the fact that they do not reason or use models becomes a restriction.

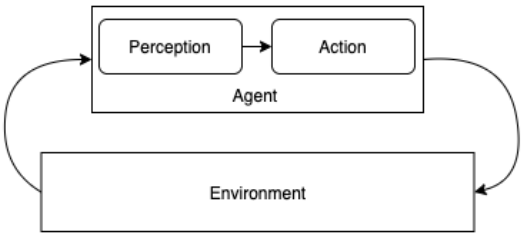


Figure 2.5: Reactive Architecture Perception/Action [Sycara, 2016].

2.2.2.3 Belief, Desire and Intention

The most common agent designs are BDI architectures [Rao et al., 1995]. In general, beliefs are the agent's views of the environment, desires are the agent's motivations or alternatives for carrying out actions, and intentions are the agent's commitments to its wishes and beliefs Chin et al. [2014]. The Procedural Reasoning System (PRS) is one of the most well-known BDI designs that is frequently used as a practical example. This architecture is based on four key data structures beliefs, desires, intentions and plans, plus an interpreter (Figure 2.6). The interpreter's job is to bring all aspects of the agent together, from updating beliefs to making observations about the environment to choosing from a list of active wishes to act as intentions. The plans are a set of actions obtained by the interpreter and suited for the agent to carry out in order to attain its goals. Rather than being generated in real time, they are a pre-compiled plan.

2.2.2.4 Layered

Agent behaviour can be reactive or deliberate in layered (hybrid) structures. Subsystems built as tiers of a hierarchy are used to handle both types of agent behaviour, allowing for this flexibility. There are two types of control flows in layered architectures: horizontal [Ferguson, 1992] and vertical stacking [Müller et al., 1994].

- The layers in horizontal layering are the links between sensory input and action output. For each of the n different types of behaviour, there are n layers. The key advantage is the design's simplicity, as it's quite simple to handle the information with only one accessible option for each situation. The enormous number of conceivable interactions between horizontal layers,

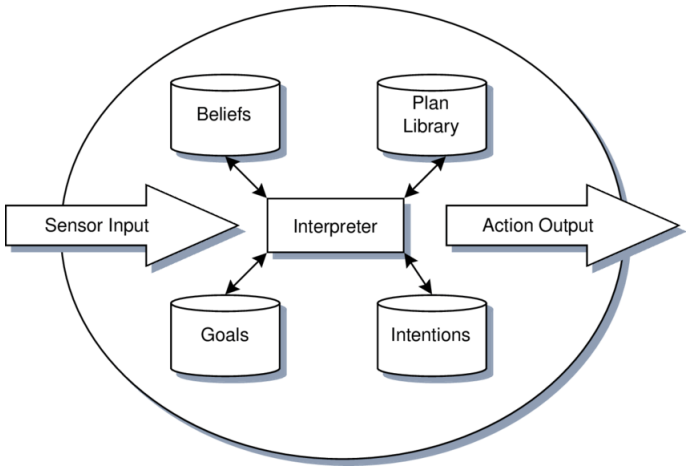


Figure 2.6: PRS architecture model [Bordini et al., 2011].

on the other hand, is a drawback because the number of levels can quickly rise.

- When compared to horizontal layering, vertical layering is similar to an automated implementation because the number of levels is much lower. Layers can be divided by tasks depending on the implementation, resulting in a system that models every received input. One-pass and two-pass control structures can be found in the vertical layered architecture. Data goes up the sequence of layers in one pass to the last layer, which generates an action. The data passes up and then back down the series of layers in a two-pass algorithm, and only then does it yield an action. The fundamental deficiency of this architecture is that it is not fault tolerant, meaning that if one layer fails, the entire system crashes [Chin et al., 2014].

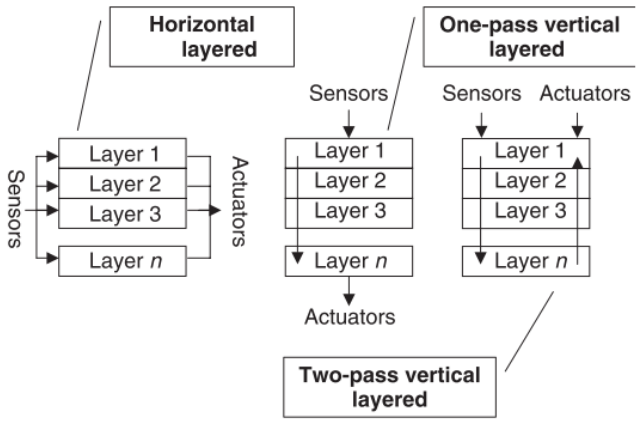


Figure 2.7: Layered Architectures [Bellifemine et al., 2007].

2.2.3 Communication and Coordination

Because agents must know when to act and how to behave, communication and coordination are essential components of multi-agent systems. Agents with competing aims must be able to coordinate their actions and collaborate if they are not to compete for the same resources [Dignum and Dignum, 2012]. It's possible that communication isn't limited to agent-to-agent. They might also be able to interface with other system components, such as system resources. Cost et al. [2001] define coordination as the process by which agents reason about their own actions and the actions they expect others to take in order to ensure that all agents in a community act in a coordinated manner toward a goal or set of goals. The main issue with coordination in agent-based communities is the desire to achieve goals [Reis, 2003].

2.2.3.1 Types

Agents can communicate information with other agents in two ways, according to Huhns and Stephens [1999]: directly and with the help of facilitators. Agents who communicate directly organise their own communication without the assistance of another agent. They are connected to other agents and can communicate with them directly (Figure 2.8). Agents who facilitate communication between two agents who are not directly connected serve as facilitators in facilitated communication. To communicate, the agents must first transmit their message to the facilitator agent, who will then forward it to the final agent (Figure 2.9).

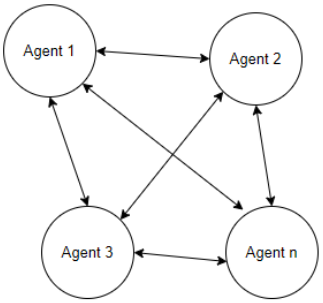


Figure 2.8: Direct Communication.

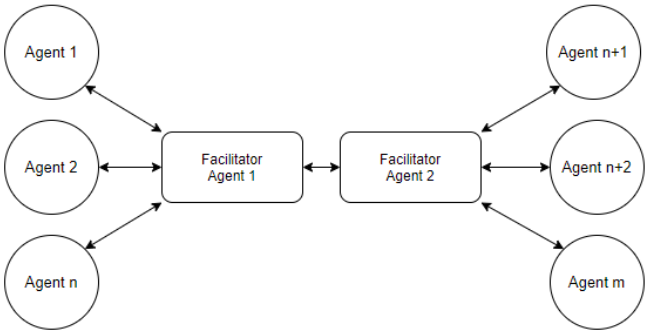


Figure 2.9: Assisted Communication [Reis, 2003].

2.2.3.2 Communication Module

By exchanging information, an agent can communicate and interact with other agents in the environment. A communication module is required for an agent to communicate.

2.2.3.3 Languages

In the beginning of the 90's, the Knowledge Sharing Effort (KSE) was founded with the objective to develop protocols to represent and exchange information between autonomous Information Systems (IS). KSE released two main protocols: the Knowledge Query and Manipulation Language (KQML), and the Knowledge Interchange Format (KIF). These were the first languages launched for agent systems. KQML is an external language [Finin et al., 1994] for agent communication focused on the format of the agents messages. The characterisation of the information is more crucial to KQML than the message's substance. KIF language addresses the representation of the knowledge of a domain. It was primarily developed to represent the content of KQML messages [Finin et al., 1993].

Currently, FIPA ACL is one of the most used and studied agent communication language. It incorporates many aspects from KQML. ACL stands for Agent Communication Language (ACL) and it was proposed by Foundation of Intelligent Physical Agents (FIPA) [FIPA, 2002c], an IEEE Computer Society standards organisation that promotes agent-based technology and the interoperability of its standards with other technologies. According to Omicini and Poggi [2006], ACL relies on speech act theory and is based on a separation between the communicative acts and the content language. It enables agents to communicate in a variety of content languages and to organise conversations using pre-defined interaction protocols.

Agents developed with the Java Agent Development Framework [JADE, 2020] use ACL to exchange messages with other agents. Below are some of the fields required by this format [Caire and Cabanillas, 2002]:

- Sender - The sender of the message;
- Receiver - The list of receivers;
- Intention - The communicative intention indicating what the sender intends to achieve by sending the message;
- Content - The content of the message;
- Language - The content language - the syntax used to express the content which must be known by the sender and the receiver to make an effective communication.
- Ontology - The vocabulary of the symbols used in the content and their meaning.
- Control fields used to manage several conversations: *conversation-id*, *reply-with*, *in-reply-to*, *reply-by*.

Listing 2.1 presents an example of a message being send to inform an agent, with the nickname Peter, about the weather using ACL.

Listing 2.1: ACL Message Example [Caire and Cabanillas, 2002].

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME);
msg.setLanguage("English");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Today it's raining");
send(msg);
```

2.2.3.4 Ontologies

Agents cooperating in a multi-agent setting need a shared ontology and a shared set of communication conventions [Wooldridge and Jennings, 1995]. This way they can communicate using a common vocabulary that is well-defined amongst themselves and prevent different terminologies for the same occurrence. According to [Huhns and Singh, 1997], an ontology is nothing more than the representation of the knowledge of a given domain, available to all other components of a information system [Weiss, 1999], defends that an ontology not only specifies the classes or types taxonomy, but also represents the existent interrelationships between them. In a multi-agent context, an ontology is the formal representation of concepts, characteristics, and interrelationships in a given domain, granting a common understanding between people and agents [Reis, 2003]. Figure 2.10 presents an example of an ontology of a generic concept, in a multi-agent system platform.

2.3 Automated Negotiation

Automated negotiation is a type of interaction in which numerous autonomous agents communicate in order to obtain agreements through a series of offers [Faratin et al., 2002]. This section is dedicated to automated negotiation and its importance to the world of distributed systems.

2.3.1 Negotiation

Negotiation precedes the signing of a contract [Kecskemeti, 2016]. According to Jennings et al. [2001], negotiation underpins attempts to cooperate and coordinate, both between artificial and human agents, and is required both when the agents are self interested and when they are cooperative (competitive agents vs. cooperative agents). Negotiation is the process where a group of agents come to a mutual agreement upon the terms of the service, e.g., rental of the service, quality level of service to be provided, responsibilities taken by each part (contractor/seller) in the agreement, penalties in case of the requirements are not

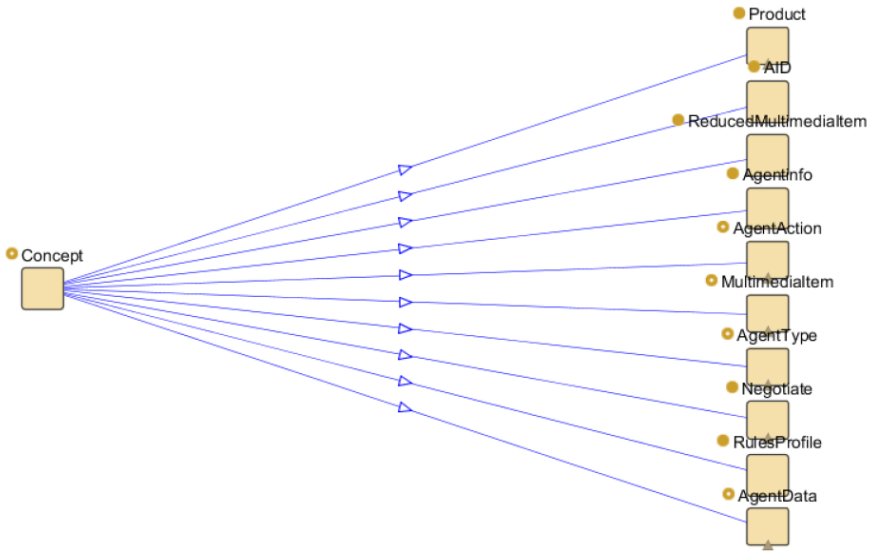


Figure 2.10: Ontology Example [Veloso, 2012].

meet, and so on. Usually the negotiation is followed by set of procedures and rules, defined by negotiation protocols.

2.3.2 Protocols

Negotiation protocols are used to regulate transactions between entities using negotiation [Alberti et al., 2004]. There are multiple negotiation protocols for automated negotiation which rely on different mechanisms, such as game-based, auction-based or iteration-based mechanisms, to govern one-to-one, one-to-many and many-to-many interactions.

In historical terms, protocols such as ARPANET and TCP/IP, which became the foundation of the Internet, are some of the first distributed control protocols that revolutionised the computational communication [Farber, 1970]. In 1980, Reid G. Smith [Smith, 1980] presented the Contract Net Interaction Protocol (CNIP), a high-level protocol for communication among the nodes in a distributed problem solver. According to the author, this negotiation protocol would facilitate the distributed control of cooperative task execution with efficient internode communication.

In this context, a set of standard one-to-many protocols was proposed by FIPA: the FIPA CNIP, the FIPA Iterated Contract Net Interaction Protocol (ICNIP), and the Fixed ICNIP (FICNIP) [Adnan et al., 2016].

2.3.2.1 Contract Net Interaction Protocol

CNIP is a task-sharing protocol used to allocate tasks in a network of autonomous asynchronous agents, wherein each agent can communicate with every other agent by sending messages. The protocol specifies the interaction between agents for fully automated competitive negotiation through the use of contracts [Alibhai, 2003]. Since its release, it suffered several improvements. In 1993, the TRACONET (TRAnsportation COoperation NET) system by [Sandholm, 1993] was released as an extension with bidding and awarding decision processes based on marginal cost calculations. In 1997, Verharen et al. [1997] described a Cooperative Information Agent (CIA) model, introducing an architecture based on a language-action perspective capable of specifying the obligations and authorisations of a contract and separating the agents tasks. Later, a concept named Contractual Agent Societies (CAS) was released by an MIT work group focused on adaptive systems and evolutionary software ASES, and it consisted on self-configurable information systems through a set of negotiated social contracts [Xu and Weigand, 2001].

CNIP was the first protocol to use a negotiation process involving a mutual selection by both managers and contractors Sandholm [1993]. Managers and contractors are roles that each agent can have at different times or for different tasks. As a manager, an agent send requests for bids on each specific subtask to all the other agents, selects the most appropriate bid and allocates the task to that agent. This agent is now a contractor, and in case of not being capable of solving the task on its own, a contractor may also ask for help the other agents by decomposing the task into subtasks, and allocating subtasks to the other agents, as subcontractors. The negotiation is a one-to-many since it relates one manager with multiple contractors. The protocol is specified by [FIPA, 2002a]. Figure 2.11) displays a message sequence chart (MSC) of the agents interaction in the selection process. The process starts with by the manager sending a call-for-proposal (CFP) message to m possible agents. Then, it is up to each agent to send back a proposal or to reject the invitation. After receiving the answers from the agents, the manager picks the one with the best offer, by sending an accept-proposal to that agent, and rejecting all others. Then, the contract is established and the negotiation process is over. The selected agent may or may not send an inform message to complete the negotiation.

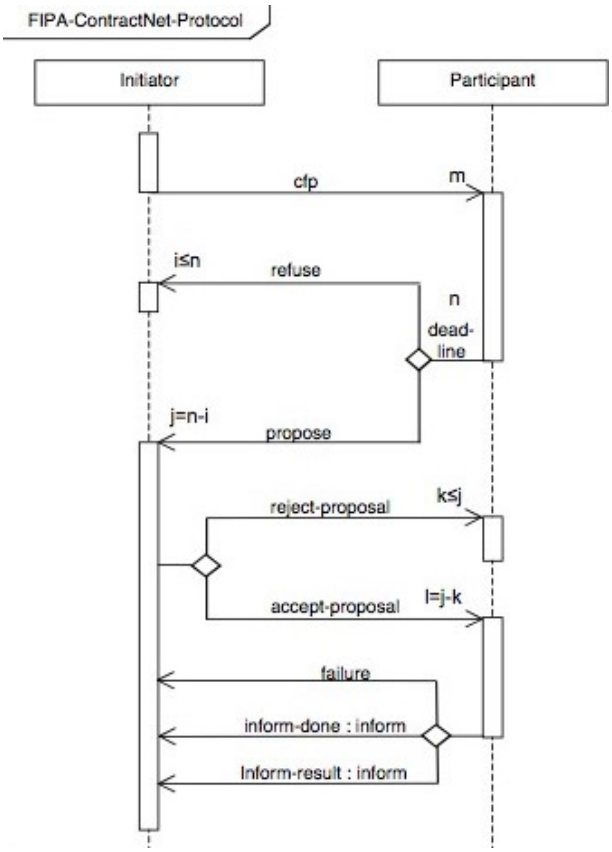


Figure 2.11: Contract Net Interaction Protocol [FIPA, 2002a].

2.3.2.2 Iterated CNIP

FIPA ICNIP was proposed in 2000 by FIPA [2000], as an extension of the basic CNIP. It is also a one-to-many negotiation protocol and its primary feature is to allow multi-round iterative bidding. Figure 2.12 shows that the negotiation may not end after the reception of the first set of proposals. The protocol flow initiates in the same manner as the CNIP. A CFP request is sent to m agents, and a group of n agents answers the request and only k are proposals, the rest refuses the CFP request. The manager selects the best proposals and rejects the others. Then the manager has two options: end the negotiation by accepting a proposal or send a new CFP message requesting a new iteration. This cycle repeats as many times as the manager wishes until it finally accepts a proposal.

2.3.2.3 Fixed Iterated CNIP

The Fixed ICNIP, or FICNIP, is a variant of the ICNIP protocol that has a fixed number of iterations. While the ICNIP protocol ends when the moderator receives a satisfactory proposal, the FICNIP protocol stops after a number of

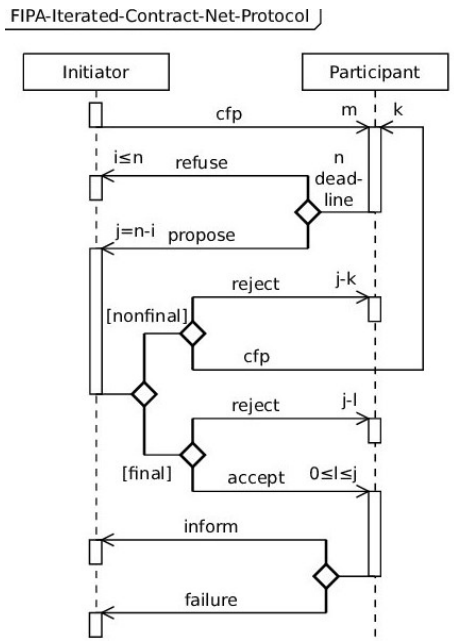


Figure 2.12: Iterated Contract Net Interaction Protocol [FIPA, 2002b].

fixed iterations regardless of the values of the buyers' offers received so far [Omatu et al., 2013].

2.3.3 Service Level Agreements

In the Internet world, the product/service quality assurance is measured by the product or the service performance. The Service Level Agreement (SLA) represents the legal document signed and agreed upon by providers and costumers, which specifies the agreed service parameters. Basically, it describes a set of non-functional requirements so that the service model can be verified against them. A requirement is defined through an objective and a set of indicators that prove the objective is being achieved [Gâteau, 2014]. The objective is called Service Level Objective (SLO) and the indicators are called Key Performance Indicator (KPI). An example of a non-functional requirement would be the maximum Return to Operation (RTO) [1] time admitted, e.g., RTO under ten minutes, and a set of KPI for monitoring. The agreement usually contains penalties in case of the requirements are not met.

2.3.3.1 Specifications

The process of making a contract should be clear for all interested parties. Standards and specifications are very important to make sure that both parties are

[1] RTO - Is the time spent to bring up a service in case of a failure.

negotiating using the same terms.

For representing an SLA there are two major specifications, both based on web services (WS) [Hung et al., 2004], namely Web Service Agreement (WS-Agreement), specified by the Open Grid Forum (OGF), and Web Service Level Agreement (WSLA), specified by the International Business Machines (IBM). In WS-Agreement (Figure 2.13), an SLA is defined in XML and a protocol, named One-Shot, used to declare the terms between providers and consumers. In WSLA (Figure 2.14), an SLA aggregates two different sections: the identification of the partners who celebrate the contract (provider, consumer, and all the Trusted Third Party (TTP)) and the services and their conditions [Ludwig et al., 2003].

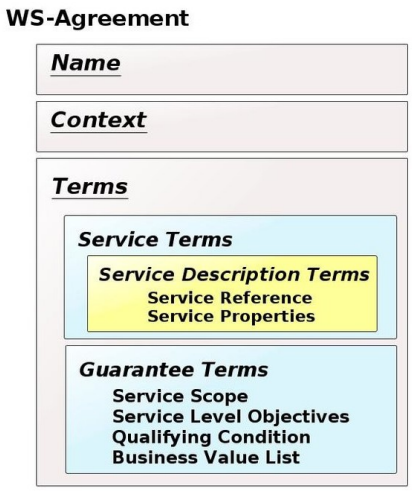


Figure 2.13: WS-Agreement Structure [Uriarte, 2015]

Figure 2.14: WSLA Agreement Structure [Uriarte, 2015].

2.3.3.2 Management

The establishment of a contract starts with the negotiation of the terms between the parties. However, the contract's management process does not end after the agreement is concluded. Contracts must be continuously monitored and may suffer readjustments if the performance does not match the expectations. As a result of this process, an SLA conditions are carefully monitored and have better chances to deal with unexpected conditions that might occur.

Observation

This stage starts once the agreement is signed. While the service is up and running, factors under external control such as the network infrastructure, may affect unpredictably the behaviour of the service. To ensure SLA metrics are consistently met, monitoring is part of the process [Sahai et al., 2002], by evaluating KPI periodically. Due to the complexity of contracts, parties tend to

delegate Quality of Service (QoS) monitoring to intermediate systems, like software agents to act on their best interest. Feedback from the monitoring system can significantly augment the precision of the analysis [de Boer et al., 2018].

Reaction

The fulfilment of an SLO describes a state of service when all of the key performance indicators are within specified thresholds. The service providers have the permission to measure and affect the KPI to be able control and alter the system if needed [Patel et al., 2009]. When the system monitoring reports violations of the agreed SLA, the goal of the service provider is to dynamically adapt the resource configuration so that violations remain under a penalty threshold, minimising the impact in the running system [de Boer et al., 2018]. This enables to automatically keep the values of the KPI under control according to the values specified by the SLO and, thus, avoid the violation of the SLA. Analysis of historical KPI data can also be used to specify new or modify existing policies [Gâteau, 2014]. This results from the evolution of automated SLA protection.

2.4 Blockchain

"Is blockchain one of the greatest technological revolutions in history or it is just hype?" [Gates, 2017].

People entrust credit card firms and banks with the obligation of withdrawing the proper amount from their bank account and depositing it in the account of the recipient in everyday life when transferring money or paying for products and services. Blockchain's main feature is that it allows untrustworthy people to securely communicate and transact without the necessity of a Trusted Third Party (TTP), making it a breakthrough technology from a personal standpoint.

The concept of "chain of blocks" was introduced in a revolutionary paper entitled *Bitcoin: A Peer-to-Peer Electronic Cash System* written in 2008 under the pseudonym Satoshi Nakamoto [Gates, 2017]. Over the years, the term evolved into the word blockchain which represents an ordered chain of blocks, where each block is identified by its cryptographic hash [Alharby and Van Moorsel, 2017]. Since then, blockchain has been applied to cryptocurrencies and smart contracts with great success.

The Bitcoin blockchain's functionality is simple: anytime two network members interact, they notify all other network members (nodes), who then record the transaction in a block with a finite capacity [Ammous, 2016]. Smart contracts first debuted in a business logic setting in 2014, with the goal of facilitating contract agreement and negotiation by ensuring trustworthiness and data integrity.

2.4.1 Definition, Advantages and Limitations

Blockchain is a peer-to-peer, distributed ledger that is cryptographically secure, append-only, immutable (very difficult to change), and updatable only through peer consensus or agreement. According to Carlozo [2017], it can be defined as a list of continuous records preserved in blocks. Distributed Ledger Technology (DLT) and blockchain are not interchangeable. Even though blockchain is a DLT, the opposite is not true. Apart from blockchain, DLT technologies also comprise Directed Acyclic Graph (DAG) and Tempo technologies (Figure 2.15), which basically differ in terms of data structure, distribution of data storage, and the consensus mechanisms being applied [BitOrb, 2019].

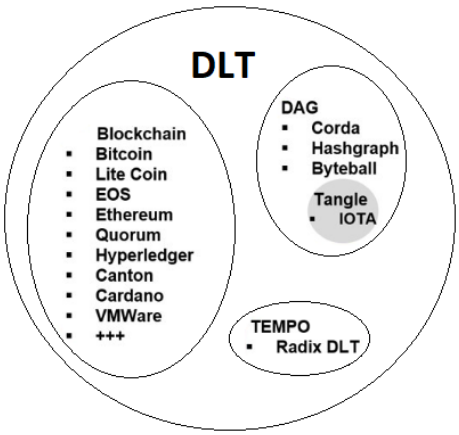


Figure 2.15: DLT Technologies [Creer, 2020].

Although the advantages of blockchain-based systems have led to their usage in a wide range of applications, their decentralized nature also creates significant obstacles.

Advantages:

- Peer-to-peer – There is no central network control because all participants are directly connected to one another.

- Distributed ledger – All peers in the network contribute to the ledger, with each holding a copy of the entire ledger. This eliminates single-point-of-failure scenarios and prevents a single controller from tampering with the shared central database [Subramanian, 2017].

- Cryptographically-secure – Cryptography is used in transactions to provide the ledger with security services against tampering and misuse. These

services include non-repudiation[2], data integrity and data origin authentication.

- Append-only – It means that once data is added to the blockchain, changing it is very hard. It can be changed if there are valid reasons, but it is regarded as practically immutable.

- Updatable via consensus – Because there is no single authority in charge of the ledger, the consensus empowers decentralisation. Any modification to the blockchain is vetted against strict criteria set by the blockchain protocol, and it is only added to the blockchain after all participating peers have reached a consensus.

Limitations:

- Scalability – The electricity expenses of running a blockchain may make it impractical to manage the volume of transactions handled by credit card firms like Visa or MasterCard at the current rate of energy usage. This aspect may have an impact on blockchain network scalability.

- Regulation – Because of the scale and cost of replacing existing systems, governments and banks are hesitant to change. As a result, blockchain may face a lengthy regulatory and integration procedure.

- Relatively immature technology – Despite the fact that it is often regarded as one of the most significant developments since the Internet, it is still a relatively new technology. Although there is a lot of potential, most of the applications are still hypothetical.

- Lack of privacy – Because there is no privacy on the ledger in public networks, all transactions are public. Users can, however, secure critical information from other nodes via private networks.

- Risk of attack – Although blockchain is recognised as one of the most secure technologies, it is not dismissed of suffering attacks. In a network where the decisions are made after reaching a consensus among the majority of the peers, 51 % attacks are among the most discussed vulnerabilities [Academy, 2017]. If a single entity manages to control more than 50 % of the network hashing power, it can disrupt the network by intentionally excluding or modifying the ordering of transactions.

[2]Non-repudiation - Legal concept that assures that someone cannot deny the validity of something. This provides proof of origin and integrity of data.

2.4.2 Structure

A blockchain, as previously defined, is made up of blocks of transactions that are packaged together and logically organized. Because there are several transactions going on at the same time, when a block reaches its transaction limit, a new block is chained to the network after being confirmed and validated, and its header references to the previous block hash. All nodes in the network receive the chained chunks of data [BitOrb, 2019]. The size of the blocks is determined by the blockchain's kind and design. Figure 2.16 presents the blockchain generic structure.

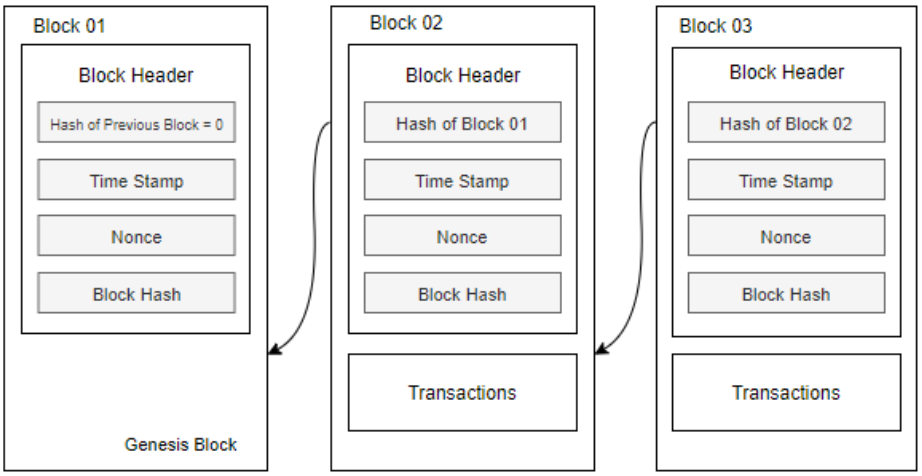


Figure 2.16: Blockchain structure [Lastovetska, 2018]

2.4.2.1 Network

Blockchain can be seen as a distributed peer-to-peer network layer running on top of the Internet (Figure 2.17).

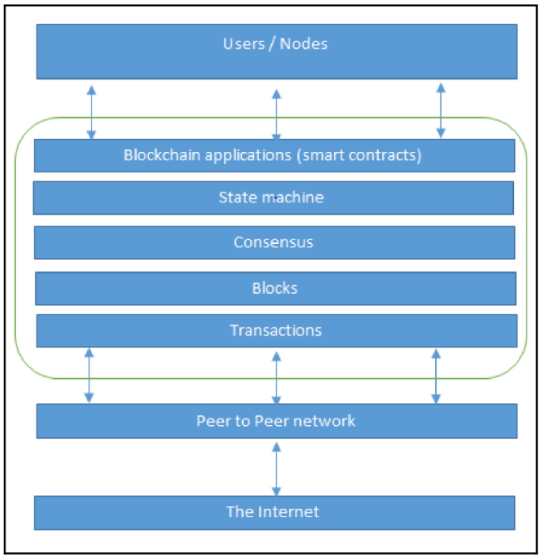


Figure 2.17: A network view of blockchain [Bashir, 2018].

Specifically, a blockchain network includes the following components:

- Transaction – Is a blockchain's fundamental unit. It denotes a value transfer from one address to another.
- Address – Is an Unique Identifier (UID) used in a blockchain transaction to denote senders and recipients. The address is usually a public key issued for each transaction to avoid associating numerous transactions to the same owner.
- Block – Multiple transactions are included, as well as other information including the previous block hash (hash pointer), timestamp, and nonce. The genesis block is the first block of the chain, and it is hard-coded when the blockchain is created.
- Nonce – Is a generated number that is only used once throughout each transaction for the purpose of validation.
- Scripting or Programming Language – Are used to facilitate the execution of various operations within a transaction occurrence. Programming languages can be *Turing complete*[3]. An example of a Turing complete programming language is Ethereum's Solidity [Solidity, 2014], which is used to define, create and run smart contracts.

[3]Turing complete - The name was given after Alan Turing who developed the Turing machine that can run any algorithm.

- Virtual Machine – Is a computational environment where smart contracts execute, such as Ethereum Virtual Machine (EVM), EOS-VM, and Chain Virtual Machine (CVM) [Cachin and Vukolić, 2017].
- State Machine – Is a mathematical model of the blockchain network's current state. As a result of the transaction execution, validation, and finalisation processes, it adopts multiple transition states.
- Node – Is a network component that can mine blocks, validate transactions, verify payments, and sign transactions, depending on its purpose.
- Smart Contract – Is a blockchain-based application that encapsulates the business logic to be performed when specific criteria are met.

2.4.2.2 Types

There are two primary paradigms when it comes to blockchain participants:

- Permissionless – The network is open to anyone who wants to be a part of it. A user does not need permission to contribute processing power to the network, which is commonly done by creating a node in exchange for a monetary incentive [Peters and Panayi, 2016].
- Permissioned – Permission is required to create network nodes. A central authority normally pre-selects contributors.

From the perspective of the users, blockchain networks can be:

- Public - Network access is public, i.e., anyone can read and submit transactions to the blockchain. These networks are more secure because they have more nodes and contributors, but become slower and apply fees to transactions.
- Private - Network access is restricted to an organisation or group of organisations. These networks are less secure, since they are more centralised than public chains [Wüst and Gervais, 2018], but tend to be faster and offer free transactions to the participating organisations.

Table 2.1 compares these different blockchain networks. The architecture blockchain to choose can be as public and permissionless as the trust put in the validators [Varghese et al., 2018]. Besides, the architecture will impact the system scalability. A public and permissionless architecture, due to the amount of validators, will consume more computational resources and time since all validators have to be synchronised to validate the chain updates [Lastovetska, 2019]. However, it will be more secure because of the distributed control since it is more manageable to tamper a single node than a thousand nodes simultaneously.

Table 2.1: Blockchain Types Cross-Comparison [News, 2019].

	Permissionless	**Permissioned**
Public	Anyone can join/read/write/commit Hosted on public servers Low scalability	Anyone can join/read Only authorised nodes write/commit Medium scalability
Private	Only authorised nodes join/read/write Hosted on private servers High scalability	Only authorised nodes join/read Only the network operator can write/commit Very high scalability

2.4.2.3 Tiers

There are three blockchain levels (Figure 2.18) known as blockchain tiers, representing blockchain technology evolution.

- Tier 1 (Blockchain 1.0) is primarily dedicated to cryptocurrencies and it was first introduced with the appearance of the Bitcoin.
- Tier 2 (Blockchain 2.0) is used by financial services and smart contracts. Ethereum, Hyperledger, and other newer blockchain platforms are considered part of Blockchain 2.0.
- Tier 3 (Blockchain 3.0) is to implement applications beyond the financial services industry such as for government, health, media, arts, and justice. Blockchains with the ability to code smart contracts are considered part of this technology tier, such as Ethereum, and Hyperledger.

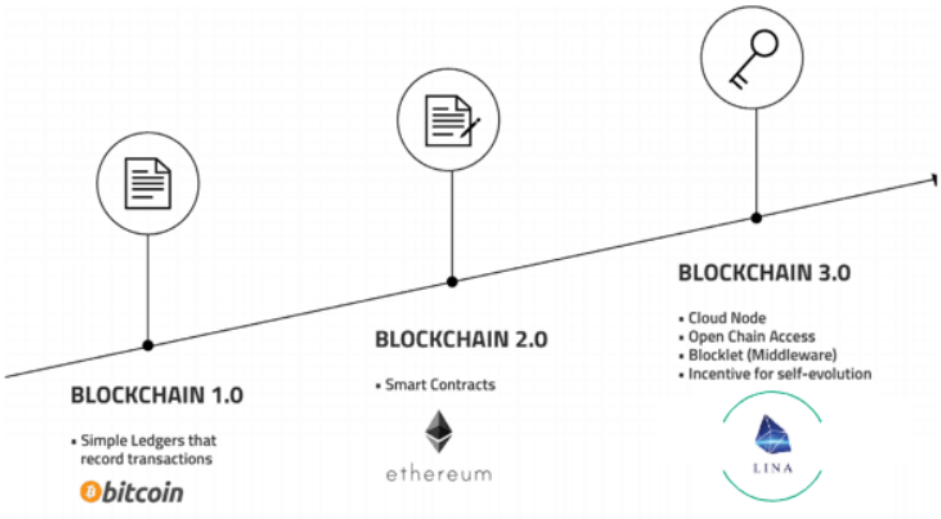


Figure 2.18: Blockchain generations [Lina.Network, 2019].

2.4.3 Consensus Protocols

As previously stated, the blockchain is updated via a consensus process that maintains a common, clear ordering of transactions while also ensuring the network's integrity and consistency among distributed nodes [Baliga, 2017]. The ability to reach a "distributed consensus" is essential since it constitutes an agreement between many mutually-distrusting parties who lack identities [Poelstra, 2015]. The mechanism utilised to reach consensus among untrusted nodes, which is inspired in the Byzantine generals problem, is known as Byzantine Fault Tolerance (BFT) consensus protocol [Zheng et al., 2017].

The Byzantine generals problem was proposed by Lamport et al. [1982] to illustrate how to handle computer malfunctioning components that give conflicting information to different parts of a system. A number of generals has surrounded an enemy city, each in charge of a part of the Byzantine army. They communicate by oral messenger and must agree on a common battle plan. However, there may be a few traitors among them who will try to boycott the plan. The problem is to find an algorithm that ensures loyal generals will reach an agreement. Traitors will do whatever they wish. If more than two thirds of the generals agree to attack, the army attacks the city, otherwise it retreats.

Fault tolerance in distributed systems refers to two types of faults: benign faults caused by hardware or software crashes that cause node failure and prevent it from participating in the consensus protocol, and Byzantine faults caused by software bugs or as a result of the node being compromised that cause the node to behave erratically. The consensus protocol has to be able to operate correctly and reach consensus in the presence of Byzantine nodes as long as the number of Byzantine nodes within a distributed system are limited. Practical Byzantine Fault Tolerance (PBFT), Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA) are instances of blockchain consensus protocols based on BFT.

- PBFT - Uses the concept of replicated state machine and voting by replicas for state changes. Additionally, it provides several optimisations, such as signing and encrypting messages exchanged between replicas and clients [Baliga, 2017].

- PoW - Each node of the network calculates a hash value of the block header. The block header contains a nonce that is frequently changed to calculate different hash values. The calculated value must be equal to or smaller than a certain given value. When one node reaches the target value, it broadcasts the block to other nodes and all other nodes must mutually confirm the correctness of the hash value [Zheng et al., 2017]. If the block is validated, other miners append this new block to their own blockchains. This consensus strategy is used by Bitcoin [Miller, 2014] where the nodes

that calculate the hash values are called miners and the procedure is called mining. This mechanism has low efficiency and high energy cost due to the amount of calculations that the miners have to do.

- Proof of Stake (PoS) - PoS is an energy-saving alternative to PoW. It completely replaces the mining operation with an alternative approach where nodes make virtual currency stakes to become validators. If anything wrong happens, i.e., if the node has malicious intentions and validates a fraudulent transaction, it looses its holdings, and can no longer be a validator [Saleh, 2020]. Nonetheless, if the node demonstrates to be honest, it is rewarded. The selection of validators follows a random process which takes into consideration node characteristics such as its stake (wealth). This protocol implies that wealthy nodes are less likely to attack the network [Zheng et al., 2017]. However, it could be dangerous because the richest node is bound to dominate the network, and, consequently, perform an attack.

- Delegated Proof of Stake (DPoS) - Created by Daniel Larimer, DPoS is an upgrade of PoS mechanism that sees democracy as fundamental to offset the negative effects of centralisation [Schuh and Larimer, 2017]. According to the author, in DPoS, a set of witnesses, formerly known as delegates, sign the blocks and are voted on by those using the network. Basically, the process of selecting a group of validators is voted rather than performed by random algorithms. The validation of a transaction requires a validator plus a set of 20 witnesses [Xu et al., 2018], forming a group of 21 block producer nodes. The producers are then shuffled to ensure that a balance between all producers is maintained.

- Proof of Authority (PoA) - PoA consensus is a straightforward and efficient form of PoS, which relies on a set of known validators and a governance-based penalty system. This consensus protocol restricts the creation of a block to a fixed set of n authority nodes [Ekparinya et al., 2019]. For the initial ceremony, 12 initial keys will be created and distributed to individual validators by the master of ceremony [Pavel et al., 2018]. To become a validator, the node has to prove his identity using a "proof of identity"[4]. PoA was originally proposed as part of the Ethereum ecosystem for private networks [De Angelis et al., 2018], and implemented in the Authority Round (AuRa) and Clique clients. Comparatively to DPoS, PoA demands proof-of-identity rather than the holdings of the validator. According to Metadium [2018], this is an advantage of PoA over DPoS because those with larger holdings may not be the ones placing highest stakes. PoA, by putting identity at stake, guarantees the integrity of validators.

[4]Proof of identity - In the case of the PoA consensus protocol, a proof of identity is a Decentralised Application (Dapp) connecting the user identity to his/her wallet.

2.4.4 Smart Contracts

"A smart contract is an electronic transaction protocol that executes the terms of a contract" Szabo [1997]. This idea of smart contracts was first implemented by Bitcoin in 2009.

Deriving from the development of blockchain technology, smart contracts are computer programs running on blockchain nodes to facilitate, execute and enforce the terms of an agreement between untrusted parties without the involvement of any TTP [Hu et al., 2018]. A smart contract usually contains some business logic and a limited amount of data [Bashir, 2018]. Smart contracts run autonomously on the behalf of the involved parties in the blockchain network, i.e., whenever specific criteria are met, business logic executes. Although smart contracts may execute on different types of decentralised platforms, blockchain has become their de facto standard decentralised execution platform due to its intrinsic security benefits.

According to Alharby and Van Moorsel [2017] there are two types of smart contracts: deterministic and non-deterministic. A deterministic smart contract is a one that does not require information from an external party, such as oracles or data feeds, during its execution. A non-deterministic smart contract, on the other hand, does require information from an external party, such as oracles or data feeds. Also, smart contracts can be public or private according to the blockchain platforms in which they operate. Public smart contracts run on a public blockchain, which means that anyone can use them. Organisations that hold a private blockchain use private smart contracts. Compared to the inefficient and expensive validation processes of public blockchains, private blockchains are more suitable for stimulating business collaborations [Hu et al., 2018].

2.4.4.1 Behaviour

There are some principles and protocols to model the behaviour of a smart contract, such as:

- They should work on the principle that code is law; meaning that there is no need for an arbitrator or a TTP to control or influence the execution of the smart contract;
- They are enforceable, which means that all contractual terms are executed as defined and expected, even in the presence of adversaries;
- They must be secure and unstoppable, which means that these computer programs must be fault-tolerant and executable in a fair amount of time.
- They should be able to execute and maintain a healthy internal state, even if external factors are unfavourable.

2.4.4.2 SLA Integration

SLA management is one of the most promising blockchain use cases [Hyperledger, 2019a] because sensitive SLA can benefit from the security benefits that blockchain provides. In the event of a SLA violation, for example, the customer might file a claim with the service provider for compensation. However, because both sides can be dishonest, the process may not be straightforward. Blockchains and smart contracts can assist to address these difficulties by offering an immutable data storage system that can detect service violations, confirm them, and ensure that the customer is compensated [Zhou et al., 2018]. Thus, it provides an alternative trustful system to TTP, which is costly and increases bureaucracy, to ensure the correct payment between parties [Scheid et al., 2019].

2.4.5 Decentralised Applications

Decentralised applications, also known as Dapp, are a new model for building successful and massively scalable aplications. They are considered one of the latest advancements in decentralisation technology [Johnston et al., 2014]. Its goal is to make commercial transactions, government processes, supply chains, and all other systems that require mutual trust between consumer and supplier or user and provider more transparent [Infante, 2019].

A Dapp should be built on top of open, decentralised, peer-to-peer infrastructure services. According to Hu et al. [2018] and Johnston et al. [2014], for an application to be considered a Dapp, it must comply with the following criteria:

- Be open-source and operate autonomously, with no entity controlling the majority of its tokens[5].
- Store cryptographically in a public decentralised blockchain its data and records of operation.
- Generate tokens according to a standard algorithm or set of criteria.
- Adapt its protocol in response to proposed improvements and market feedback, with changes being decided by majority consensus of its users.

There are three types of Dapp [Dapp, 2019]:

- Type I - Have their own the blockchain, i.e., are independent instances.
- Type II - Are protocols that rely on Type I Dapp for functioning.

[5]Token - The purpose of a token is to allow access to a computer application, e.g., an individual must own a number of bitcoins in order to be able to perform any transaction on the Bitcoin network [Johnston et al., 2014].

- Type III - Are protocols that depend on Type II Dapp instances.

According to this criteria, it may be expected that there will be less Type I Dapp, more Type II Dapp and a considerable number of Type III Dapp instances.

2.5 Summary

The key principles involved in designing and implementing a SLA smart contract solution for the CloudAnchor B2B brokerage platform, which is a MAS dedicated to the automated negotiation of IaaS resources between providers and clients, were covered in this chapter. To that purpose, the most important elements of distributed systems, multi-agent systems, automated agent negotiation, and, lastly, blockchain have been summarised.

The next chapter discusses B2B platforms in general and CloudAnchor in particular.

Chapter 3

B2B Platforms and CloudAnchor

This chapter introduces the Business-to-Business (B2B) e-Commerce model, reviews some B2B platforms, and describes the CloudAnchor platform, which is used in this project.

3.1 Electronic Commerce Transactions

Electronic commerce, or e-Commerce, is a type of commerce where commercial transactions occur via the Internet. There are different e-Commerce types, depending on the product or service, the business sector, the technology being utilised, the amount, and so on. The most common e-Commerce categories are: Business-to-Business (B2B), Business-to-Consumer (B2C), Business-to-Administration (B2A), and Consumer-to-Administration (C2A). This section describes four different categories of e-Commerce, associating each category to a specific market scope.

3.1.1 Business-to-Business

According to Lucking-Reiley and Spulber [2001], B2B includes all electronic transactions between companies, including trades, purchases of services, resources, technology, manufactured parts and components, and capital equipment. It is a fast growing market with higher consumer activity. In 2017, e-Commerce was the preferred method for approximately 2,3 trillion USD in B2B sales [Mire, 2018]. B2B commerce can be divided in three main categories, namely e-Marketplace, e-Procurement, and e-Distribution.

- e-Marketplace is an environment where companies, can buy or sell goods, and connect commercially. These can either be hosting companies within a

certain type field, assuming a vertical structure, or can cover a wider group of types of business, assuming an horizontal structure.

- e-Procurement is a type of electronic platforms where companies can design a supply plan. Companies, which depend on a set of specific goods or services, may choose this kind of provisioning to guarantee the required goods during a specific period of time.
- e-Distribution is a virtual location where companies can integrate their business with their suppliers and distributors, forming a supply chain in order to sell goods.

3.1.2 Business-to-Consumer

B2C represents the transactions operated between company and consumers. These transactions could happen dynamically and often, or can be sporadic and punctual, depending on the type of commerce that is being practised. This type of e-Commerce has increased its share due to the natural evolution of the Web, by having more people shopping online.

3.1.3 Business-to-Administration

B2A includes all the electronic transactions performed between companies and public administration. These transactions may occur in a wide variety of areas, such as fiscal affairs, social security, or employment.

3.1.4 Consumer-to-Administration

C2A represents the transactions performed between consumers and public administration. This kind of transactions may occur in areas such as social security, health care, education, and taxes. It also results from society modernisation and the natural evolution of the Internet [Cunha et al., 2017].

3.2 B2B Negotiation Platforms

There are a large number of e-Commerce applications using agent-based technology to automate business activities, decreasing the time spent and also the transaction costs. In this section are presented and compared a few agent-based e-Commerce platforms in terms of structure, architecture, and features. Table 3.1 summarises the gathered information.

- Tamani and Evripidou [2006] present an agent-based community-centred P2P decentralised organisation of service providers, clients and intermediaries, that effectively and securely collaborate with each other. The purpose

of this application is the efficient discovery of trustworthy services matching client requests to the greatest extend. This MAS arranges agents in groups such as service provider agents, community agents, broker agents, reputation manager agents, and so on. They measure partners trust with the help of a trust and reputation model.

- Chichin et al. [2014] present Smart Cloud Marketplace (SCM), which is an agent-based cloud platform for trading cloud services. Software agents are used to represent cloud service consumers and providers in the marketplace, and make smart judgements on their behalf. The platform allows the agents to use various trading policies to negotiate more efficiently in different situations.

- In Cretan [2016] is proposed an agent-based intelligent platform to model and support parallel and concurrent negotiations among organizations acting in the same industrial market. The platform enhances the concept of a Virtual Enterprise (VE) to form a temporary coalition between companies in order to share skills and resources, to respond better to business opportunities. The coalition is established through contracts after a period of negotiation that, depending on the number of participants, can be bilateral (one-to-one), one-to-many, or many-to-many.

- Cocconi et al. [2018] propose a cloud-based platform for executing Collaborative Business Processes (CBP). For a business to execute CBP they must have a Process-Aware Information System (PAIS), which is high-costly and demands a complex IT infrastructure to deploy. The platform uses a MAS system to model the PAIS, using a software agent for each business PAIS, called process agent. The communication architecture adopted between the platform and businesses is P2P, which means that all interactions are carried out in a decentralised way [Cocconi et al., 2017].

- Brousmichc et al. [2018] propose an agent-based simulation framework to experiment blockchain-backed energy marketplaces. Basically, it combines MAS technology with blockchain to offer a secure decentralised solution. Each agent represents a household located in Lille, France, modelling their energy production and consumption profiles. Based on these data, each agent proposes to buy or sell energy volumes on the platform market. The platform defines two types of smart contracts: “The Wallet” contracts to represent the households accounts and the “Marketplace” contract for representing the marketplace where the offers are placed.

3.3 CloudAnchor Platform

CloudAnchor is a brokerage platform, written in Java, conceived to help Small and Medium Sized Enterprises (SME) trading Infrastructure as a Service (IaaS) resources, either as providers or consumers. The project was financed by the Portuguese Innovation National Agency (ANI). It started essentially in a scholar context, receiving contributions by four final degree students, all coordinated by Prof. Maria Benedita Malheiro, an enthusiastic for distributed systems and artificial intelligence.

The development of a predecessor platform started in 2010 with a layered multi-agent brokerage and negotiation platform for the "personalisation" of the publicity transmitted by media distributors during intervals by Sousa [2012]. In the end of 2012, with the contributions of Veloso [2012], not only the platform filled programme intervals with ads compatible with the profile, context and expressed interests of each viewer, but acted as B2B electronic marketplace for advertising agencies (content producer companies) and multimedia content providers (content distribution companies). In 2015, Veloso et al. [2015a] reapplied the concept to the recommendation, negotiation and transaction of personalised media content. This version negotiated media items on behalf of the media content distributors and sources, to provide viewers with a personalised electronic programme guide (EPG). The platform was enriched with SLA support, namely, brokerage SLA (bSLA) between individual businesses and the platform regarding the provision of brokerage services; and item SLA (iSLA) between producer and distributor businesses about the provision of media items Veloso et al. [2015b].

In 2014, the concept was applied to the automated negotiation and transaction of single and federated IaaS resources between SME providers and consumers as part of the CloudAnchor project. The first version of the CloudAnchor broker negotiated and established Brokerage Service Level Agreement (bSLA) between the platform and each provider or consumer, Coalition Service Level Agreement (cSLA) between the members of a coalition of providers, and Resource Service Level Agreement (rSLA) between a consumer and a provider. Federated resources were detained and negotiated by virtual providers on behalf of the corresponding coalitions of providers Veloso et al. [2016]. This broker was supported by a common API taxonomy for the integrated management of open source and proprietary IaaS resource platforms developed by Meireles and Malheiro [2014]. The second version of CloudAnchor brokerage platform was released in 2016. Cunha et al. [2017] added a contract renegotiation mechanism designed by Veloso [2017] based on past business behaviour. Basically, the contracts with fulfilling partners are rewarded with discounts, whereas those with faulty partners are penalised with larger fees. This contract renegotiation mechanism increases the satisfaction of compliant companies and incites the non-compliants to change their be-

haviour or stop bidding for contracts. In 2020, Veloso et al. [2020] report a third CloudAnchor version which implements trust and reputation based brokerage and negotiation. The contributions timeline is illustrated in Figure 3.1.

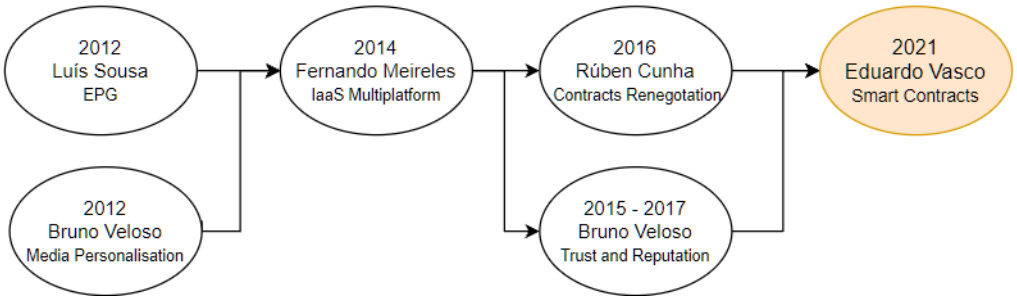


Figure 3.1: CloudAnchor timeline.

3.3.1 Architecture

The CloudAnchor broker presented in Figure 3.2 interacts with the interoperable API DeltaCloud [Foundation and RedHat, 2009] for the IaaS resource management. DeltaCloud offers a unique Application Programming Interface (API) which fully integrates the different types of clouds, abstracting any existing differences. This way, CloudAnchor ensures interoperability with the different IaaS providers, and, at the same time, avoids vendor lock-in problems.

CloudAnchor is also sub-divided in layers, with distinct responsibilities within the application. The main layers of CloudAnchor (Figure 3.3) are: the interface layer, the contract layer, the business layer, and the market layer. Additionally, there are five types of dedicated agents: (i) interface agents to interact with consumer and provider businesses; (ii) contract agents to manage SLA instances; (iii) business agents; (iv) market agents; and (v) layer agents responsible for the management of each platform layer [Veloso, 2017]. Each layer comprises a layer and multiple business agents.

3.3.1.1 Interface Layer

Operating as a bridge between the real world enterprise and platform, the interface layer handles all bi-directional communication. Each business has its own interface agent which offers a group of actions for the business to access all the relevant platform information, including the results, represented graphically. It behaves as the API of the platform. To avoid any malicious control over business agents, it uses a token mechanism which adds the business identifier to each request.

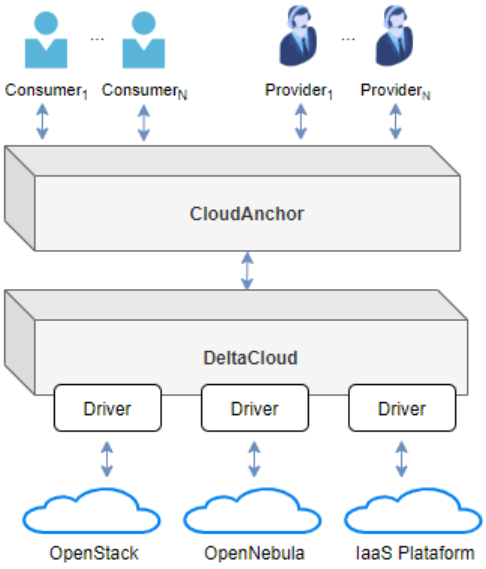


Figure 3.2: DeltaCloud integration API.

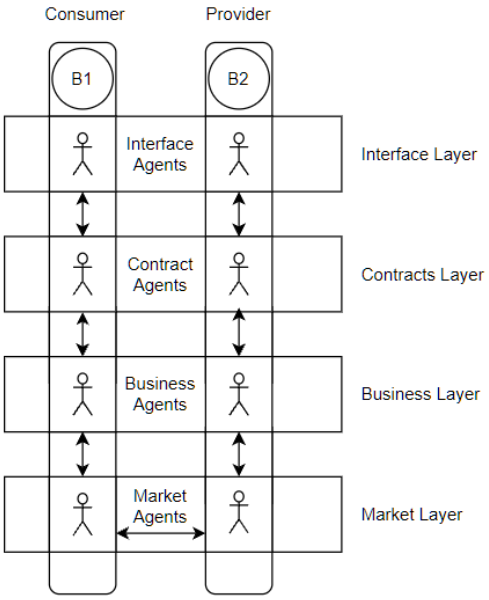


Figure 3.3: CloudAnchor application layers.

3.3.1.2 Contract Layer

For a company to join the platform, it needs to negotiate up to three types of SLA: brokerage SLA (bSLA), resource SLA (rSLA), and coalition SLA (cSLA).

- bSLA contracts specify the fee a business pays to the platform every time it successfully trades a resource.
- rSLA contracts specify the terms of the service to be supplied by the provider to the consumer.
- cSLA contracts are celebrated when a collection of providers join resources to fulfil a large resource request, that they could not support individually. These contracts detail the coalition supply terms and the providers that participate in the coalition.

When a brokerage contract is terminating, the business agent starts a contract renewal negotiation with the platform. Figure 3.4 describes the life-cycle of an SLA. Every contract instantiates a common template to identify the negotiating entities. If the negotiation succeed, the contract is filled with negotiation terms, otherwise the negotiation ends and the template is discarded. In case of a brokerage SLA, the termination of the contract may lead to the termination of the partnership between business and platform, or can be followed by a renegotiation offer to continue to use the platform.

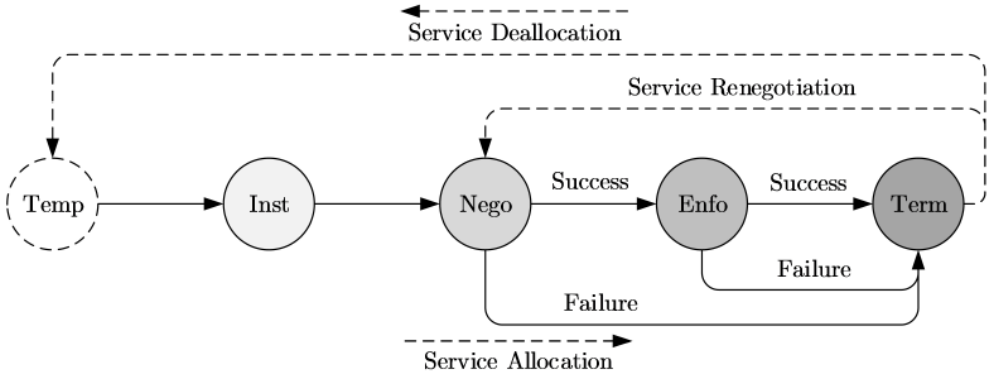


Figure 3.4: Contract life-cycle [Veloso, 2017].

3.3.1.3 Business Layer

The business layer, or the enterprise layer, is the layer where real world businesses are modelled by software agents. Each agent within this layer is responsible for representing a business, keeping its strategic knowledge, acting autonomously, and reporting the results whenever requested to.

3.3.1.4 Market Layer

The market layer, which is also the bottom layer, is populated by market agents responsible for negotiating resources on behalf of the businesses they are representing. To negotiate the resources, agents use the protocols CNIP, ICNIP, and FICNIP.

3.3.2 Trust and Reputation based Negotiation

Trust and Reputation (T&R) modelling allows the platform to select the best partners for providing or consuming services. The platform creates and maintains a log registry on all entities (platform, consumer, provider, and virtual provider[1]), analysing their individual SLA fulfilment track record. This allows the platform to continuously adjust individual trust and reputation levels based on past interactions [Veloso, 2017].

Every time an SLA is established between providers and consumers, there is the possibility of SLA failure, resulting in bad notoriety for the platform. So, to anticipate SLA failures scenarios, the providers and consumers which fail less, build a better reputation and get rewarded with better contract conditions, and the providers and consumers which tend to violate contracts more frequently, get

[1]Virtual providers are temporary coalitions of providers established on the fly to provide federated resources, i.e., resources which were not offered by any single provider [Veloso, 2017].

less opportunities and pay higher fees. This way, the platform benefits the trustworthy partners and prevents the others from affecting the overall performance.

3.4 Summary

This chapter introduced electronic commerce transactions - B2B, B2C, B2A, and C2A - and described selected e-Commerce agent-based platforms - Tamani and Evripidou [2006], Chichin et al. [2014], Cretan [2016], Cocconi et al. [2018], and Brousmichc et al. [2018] - together with the features of the proposed CloudAnchor version. Table 3.1 compares these platforms in terms of architecture and communication paradigm, SLA, smart contract support and negotiation strategy. The Brousmichc et al. [2018] platform is the one that most closely resembles the CloudAnchor platform. However, the CloudAnchor platform is the only one having a Trust and Reputation module, despite the fact that both systems implement the MAS and automated negotiation concepts. The comparison highlights the relevance of the current proposal – enrich the CloudAnchor broker with smart contracts.

Table 3.1: Platforms Comparison.

	MAS	P2P	SLA	Smart Contracts	T&R
Tamani and Evripidou [2006]	Yes	Yes	Yes	No	No
Chichin et al. [2014]	Yes	Yes	Yes	No	No
Cretan [2016]	Yes	Yes	Yes	No	No
Cocconi et al. [2018]	Yes	Yes	Yes	No	No
Brousmichc et al. [2018]	Yes	Yes	Yes	Yes	No
Current Proposal	Yes	Yes	Yes	Yes	Yes

The next chapter analyses existing blockchain development platforms to choose the most appropriate for this project.

Chapter 4

Blockchain Development Tools

This chapter examines the benefits and drawbacks of various blockchain technologies in order to determine which is the best fit for CloudAnchor.

4.1 Blockchain Platforms

Although there are several blockchain platforms that allow smart contracts, each one has its unique set of capabilities and advantages. As a result, it's critical to evaluate and compare them before deciding which one best meets the standards outlined in Section 1.3.1.

The analysed blockchain platforms are Ethereum [2020], Hyperledger [2019b], Corda [2019], and EOS [2020].

4.1.1 Ethereum

Founded in 2014 by Vitalk Buterin, Ethereum is a public open-source blockchain platform featuring smart contracts. Ethereum smart contracts are written using a built-in Turing-complete programming language, named Solidity [2014]. Solidity is a high-level contract-oriented language with similarities to JavaScript and C languages. Additionally, Ethereum includes an internal coinage called Ether (ETH), sometimes known as gas, that users use to pay for everyday transactions and services.

Ethereum public blockchain uses a PoW-based system, which has a work rate of about 15 t/s (transactions per second) and takes 20 s to generate a block. The program, on the other hand, can be customized for use in a private network, which increases its efficiency. Table 4.1 presents the information gathered about Ethereum public blockchain.

Table 4.1: Ethereum Public Blockchain Specification.

Official Website	ethereum.org
Year Released	2014
Open-Source	Yes
Smart Contracts	Yes
Blockchain	Public, Permissionless
Smart Contract Language	Solidity
Consensus Mechanism	PoW-based
Transactions per Second (t/s)	15
Block Mining Time (s)	20
Crypto	Ether (ETH)

4.1.2 Hyperledger

Hyperledger is an enterprise-grade open-source project started in 2015 by the Linux Foundation. The Hyperledger Project (HLP) works as an umbrella for six different Distributed Ledger Technology (DLT), namely Besu [2020], Burrow [2020], Fabric [2020a], Indy [2020], Iroha [2020], and Sawtooth [2020] projects. The Hyperledger sub-projects set themselves apart from other blockchain systems by having a modular and extendable architecture that enables for the creation of highly scalable solutions [Hyperledger, 2017]. The Fabric DLT, being the first project under the Hyperledger, provides smart contracts support. One of the biggest advantages of using an Hyperledger DLT is the transaction rate. Fabric developers declare to have reach nearly 20 000 t/s using Fabric DLT. Since then, the technology has seen this rate increase. The Hyperledger projects, unlike Ethereum, do not require mining because, as private blockchain solutions, the nodes can reach agreements using other consensus algorithms. However, the Besu project is an Ethereum client that can implement a public and private solution, using PoW or PoA mechanisms. The consensus methods in Fabric, on the other hand, are not rigid: the user can plug in the mechanism that best suits his needs, or simply not use one at all, because Hyperledger uses the PBFT protocol by default [Arogyalokesh, 2018]. Table 4.2 presents the information gathered about the Hyperledger platform.

Table 4.2: Hyperledger Fabric Specification.

Official Website	hyperledger.org
Year Released	2015
Open-Source	Yes
Smart Contracts	Yes
Blockchain	Private, Permissioned
Smart Contract Language	Various
Consensus Mechanism	Pluggable Mechanism
Transactions per Second (t/s)	20 000
Block Mining Time (s)	None
Crypto	None

4.1.3 Corda

Founded in 2013, by R3 Consortium, Corda is a private open-source blockchain with smart contracts support, designed for regulated financial institutions [R3 Consortium, 2013]. Corda allows to build and deploy distributed apps that transact in strict privacy under the name of CorDapps [CorDapp, 2020]. Corda, unlike other blockchains, does not rely on blocks to keep track of transactions. Instead, it employs a notary service distributed over clusters that validates transactions and prohibits "double-spends" by participants [Corda, 2020]. According to the R3 website, the Corda blockchain can handle about 600 t/s and, as a private blockchain, it is highly-scalable. Also, its consensus is established at the level of individual deals [Brown et al., 2016], i.e., only individuals reach consensus, not the global system. The consensus mechanism is "pluggable", allowing notary clusters to choose a consensus algorithm based on their privacy, scalability, or compatibility requirements. CorDapps are a set of JAR files, containing class definitions, written in Java and/or Kotlin. Table 4.3 presents the main specifications of the Corda private blockchain platform.

Table 4.3: Corda Blockchain Specification.

Official Website	corda.net
Year Released	2013
Open-Source	Yes
Smart Contracts	Yes
Blockchain	Private, Permissioned
Smart Contract Language	Java, Kotlin
Consensus Mechanism	Pluggable Mechanism
Transactions per Second (t/s)	600
Block Mining Time (s)	None
Crypto	None

4.1.4 EOS

Founded in 2018 by Block One [2019], EOS is a high-performance alternative to Ethereum with smart contract support. EOS can implement a public or private blockchain and is completely open-source. When it comes to making EOS ready for scalable and speedy decentralized applications, the developers have great hopes. In fact, according to [Xu et al., 2018], EOS has two main ambitious goals in its road map: the complete removal of transaction fees in its public implementation and the ability to process millions of transactions per second. This technology is still on its early days when compared to Ethereum, but looks very promising. EOS smart contracts can be programmed in various languages since it has a Web Assembly (WASM) compiler. So, languages such as C, C++, Java or Rust can be compiled into WASM files and then be executed by its interpreter. The technology implements the DPoS mechanism, which, in fact, was

created by one of its founders. As previously noted in Section 2.4.3, DPoS only requires a small group of block producers to validate transaction, ensuring a high transaction throughput. Although there is not a precise transaction rate value, developers claimed that they reached nearly 10 000 t/s during tests [Medium, 2020]. Due to DPoS, a new block takes about 500 ms to be produced, which is comparatively less when compared to Ethereum public blockchain. Table 4.4 presents the main specifications about the EOS blockchain platform.

Table 4.4: EOS Blockchain Specification.

Official Website	eos.io
Year Released	2018
Open-Source	Yes
Smart Contracts	Yes
Blockchain	Public/Private, Permissioned
Smart Contract Language	C++, Rust, Java
Consensus Mechanism	DPoS
Transactions per Second (t/s)	> 2000
Block Mining Time (s)	0.5
Crypto	EOS

4.2 Implementation Clients

To install and operate the blockchain network, a blockchain client must be chosen. A client refers to a single blockchain technology implementation and configuration. The same blockchain, for example, can be designed for low throughput and low hardware resource consumption, or for serving a high-demand network. It's critical to examine and compare the many blockchain clients before deciding which one to use to deploy our network. One of the prerequisites is that the network be private, therefore here are some of the clients who use the blockchain platforms we've already discussed:

- Ethereum - the literature highlights four Ethereum clients that can be used for deploying a private network solution: Geth [2013], Nethermind [2020], OpenEthereum [2020], and Hyperledger Besu [2020];
- Hyperledger - the main active projects that currently work with Hyperledger technology are: Fabric, Indy, Iroha, and Sawtooth [Hyperledger, 2020b];
- Corda - has various community CorDapps and projects to allow the user to perform a variety of network deployments. The specifications are similar to the ones presented in Table 4.3.
- EOS - Cleos [2020], and Eoslime.js [2020].

These clients support smart contracts and can be used to implement our solution. However, due to the available documentation, and transaction, the following clients will be considered: Ethereum Geth, Hyperledger Besu, and Hyperledger Fabric. Both Geth and Besu clients provide Ethereum private implementations, while Fabric has its own blockchain implementation.

4.2.1 Geth

Go Ethereum (Geth) is one of the three original implementations of the Ethereum protocol. It is written in Go language, fully open-source, and licensed under GNU LGPL v3 [Geth, 2013]. Geth supports both the standard DAG-based PoW consensus and a PoA consensus implementation called Clique [Zhang, 2018]. Clique PoA is expected to boost the performance of the Ethereum network, improving the values presented in Table 4.1. As a result of its private nature, the validation procedure should be faster than that of the Ethereum public blockchain. Using Geth, smart contracts can be written in Solidity, and compiled using a Solidity compiler program. Also Geth is compatible with web3j, and Nethereum libraries, that can be used for controlling the blockchain from an external application. Figure 4.1 presents a common Ethereum application node architecture in which Geth serves as a client.

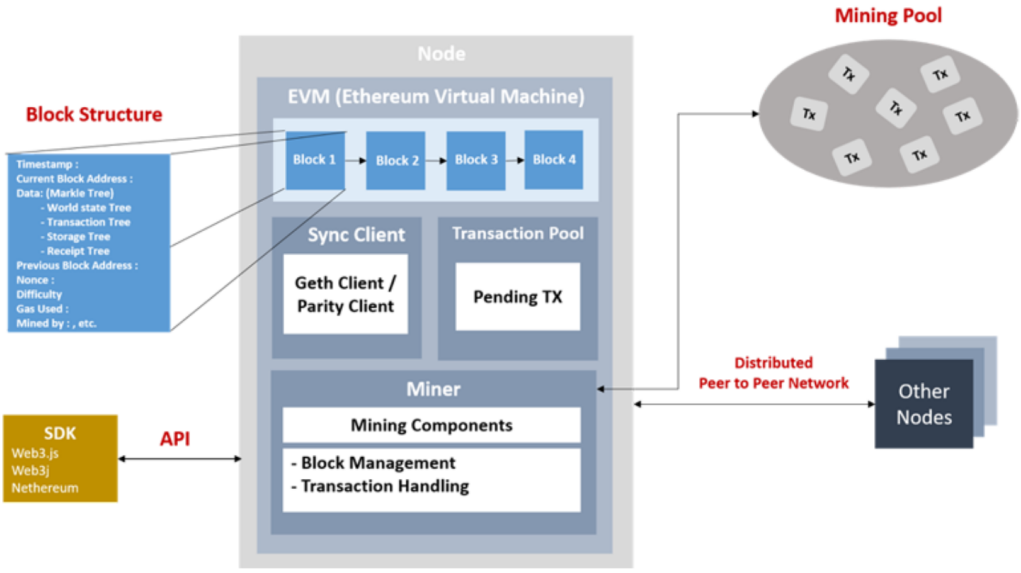


Figure 4.1: General Ethereum Node Architecture [Flentas, 2019].

4.2.2 Besu

Written in Java, Hyperledger Besu is one of the few clients that supports Enterprise Ethereum, as defined by the Enterprise Ethereum Alliance (EEA) specifi-

cation. This open-source project is unique among the Hyperledger DLT because it is specified outside of the HLP governance. It is built on a modular design that enables pluggable implementations of key components such as consensus, cryptography and signing wallets [Zhang, 2018]. From the referenced Ethereum clients presented above, only Besu is Apache 2 licensed under HLP stewardship. According to Dawson [2019], EEA specification was established to create common interfaces amongst the various open and closed source projects within Ethereum, to ensure users do not have vendor lock-in, and to create standard interfaces for teams building applications. It offers three consensus algorithms to choose from: PoW, IBFT, and Clique PoA. The Besu API can be accessed using JavaScript Object Notation (JSON) with Remote Procedure Call (RPC), RPC Pub/Sub over WebSockets, or GraphQL over HTTP. Web3j and ethereumj are some of the available client libraries that provides easy access to the Besu API. The user can additionally secure his identity apart from the Ethereum network and the application using the client. By using EthSigner [2020], an open-source Ethereum transaction signer written in Java and released under the Apache 2.0 license, it adds an extra layer of security to the application. Figure 4.2 presents the Besu core architecture.

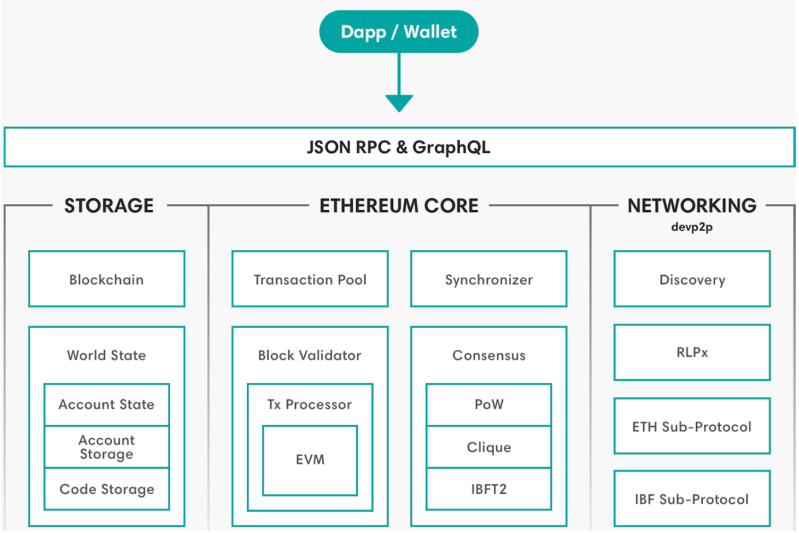


Figure 4.2: Hyperledger Besu Architecture [Dawson, 2019].

4.2.3 Fabric

Hyperledger Fabric is a high modular and versatile DLT designed for enterprise use. According to Fabric [2020b], the Fabric architecture accommodates a diversity of enterprise use cases through plug and play components, such as consensus, privacy and membership services. Also, one of the many compelling Fabric features is the enabling of a network of networks. Fabric also features smart

contracts which are known as chaincode. A single chaincode can contain multiple smart contracts (Figure 4.3). In Fabric, chaincode contracts can be written using different programming languages such as NodeJs, Golang, and Java. To access the chaincode from an external application, Fabric developers built the Fabric Gateway Software Development Kit (SDK), which supports various programming languages. Because the endorsing peers in Fabric each have their own instance of the ledger and chaincode, the consensus processes act at a peer level. They must all synchronise the received data with other peers. In Fabric, consensus can be divided in three phases:

1. Phase I starts when an application generates a transaction proposal and sends it to a set of endorsement peers in the network. These peers are responsible for executing the chaincode and generating a transaction proposal response using the transaction proposal. Once the application has received a sufficient number of signed proposal responses, the first phase of the transaction flow is complete [Hyperledger, 2020c].

2. Phase II of the transaction workflow is the packaging phase. The orderer, a special peer, receives transactions including endorsed transaction proposal responses from a variety of applications and orders their transcription into blocks [Hyperledger, 2020c].

3. In phase III, the orderer sends the generated transactions blocks to the peers, where they can be validated and committed to the ledger. Peers are connected to the orderer via the channel. For every transaction, each peer will verify that the transaction has been endorsed by the required organisations, i.e., according to the endorsement policy of the chaincode which generated the transaction [Hyperledger, 2020c].

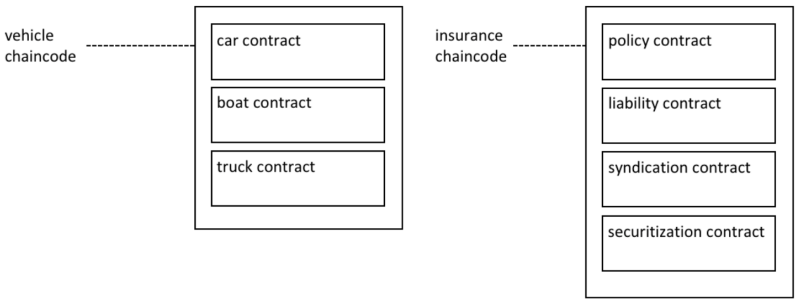


Figure 4.3: Hyperledger Chaincode [Hyperledger, 2020a].

The “pluggable consensus” refers to the type of orderer selected to conduct the transaction flow process. There are three main orderers available, namely Raft, Apache Kafka, and Solo. Raft is a Crash Fault Tolerance (CFT) ordering

service based on an implementation of the Raft protocol [Ongaro and Ousterhout, 2014]. CFT is not the same as a BFT because the fault is counted as a system crash or network down. CFT algorithms only require $2n + 1$ nodes to manage n faults, while BFT need at least $3n + 1$ to manage n faults, which makes CFT algorithms cost efficient solutions. In technical words, the Raft consensus method assures that the system can handle a failure while still processing the client request. Raft follows a "leader and follower" model [Hyperledger, 2020d] which claims that there are three states that a node can be in: *follower*, *candidate*, and *leader*. All the nodes start as a *follower*. The *leader* is the node that receives the client request and logs the request to the *follower nodes*. When the majority of the *follower* nodes sends back a confirmation message to the *leader*; the leader can commit the client's request, and generate a response to send to the client. To be promoted to a *leader*, the nodes must participate in a voting process, where nodes vote in possible *candidates*, which are *follower* self-promoted nodes. The *candidates* with more votes become *leader* nodes (Figure 4.4). In case of draw, one node times out, so the other node wins. Kafka is also a CFT implementation of the ordering service deprecated in Hyperledger v2.0. It is an adaption of Kafka distribution streaming platform, and is similar to the Raft ordering service. Solo is another implementation of the ordering service that, according to Hyperledger [2020d], has been deprecated and may be removed entirely in a future release. Fabric gains performance and scalability by separating the endorsement of chaincode execution from ordering, which eliminates bottlenecks that might occur when execution and ordering are handled by the same nodes.[Hyperledger, 2020d]. Additionally, he amount of endorsements has an impact on how many transactions per second (t/s) are processed. According to Ferris [2019], the higher the number of endorsement peers the higher the transaction rate, because the load gets divided through the peers, resulting in faster responses.

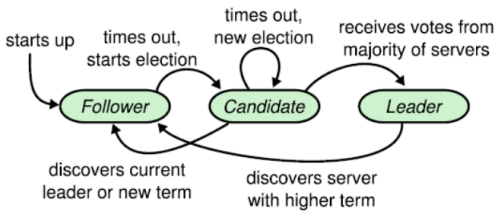


Figure 4.4: Raft Node Election Process [Singh, 2019].

4.3 Interface Libraries

Client interface libraries give access to the network smart contracts from an external application, whereas blockchain clients allow for the building and operation of the blockchain network. The purpose of this project is to link a blockchain net-

work with the CloudAnchor platform so that the defined SLA can be managed. As previously stated, the CloudAnchor platform is a Java-based application, hence Java interface libraries should be used. Table 4.5 gathers the list of interface libraries that work with the analysed clients. The final comparison focused on the two Java libraries: web3j for the Ethereum clients, and the Java Fabric SDK for the Fabric client.

Table 4.5: API List [Flentas, 2019].

API	Platform	Language
web3.js	Ethereum	GO
web3j	Ethereum	Java
Nethereum	Ethereum	C#, .NET
ethereum-ruby	Ethereum	Ruby
fabric-gateway-java	Hyperledger Fabric	Java
fabric-gateway-node	Hyperledger Fabric	NodeJs

4.3.1 Web3J

Web3j is a Java and Android toolkit for working with smart contracts and interacting with Ethereum client nodes that is lightweight, highly flexible, reactive, and type safe [web3j, 2020]. It performs the auto-generation of Java smart contract wrappers from Solidity ABI files, providing a Java API to create, deploy, transact with and call smart contracts from native Java code. Web3j uses JSON-RPC, provided by Ethereum, to allow external applications to communicate with Ethereum networks (Figure 4.5). Since the library is compatible with Besu and Geth clients for working with Ethereum blockchain networks, it could be a good fit for this project.

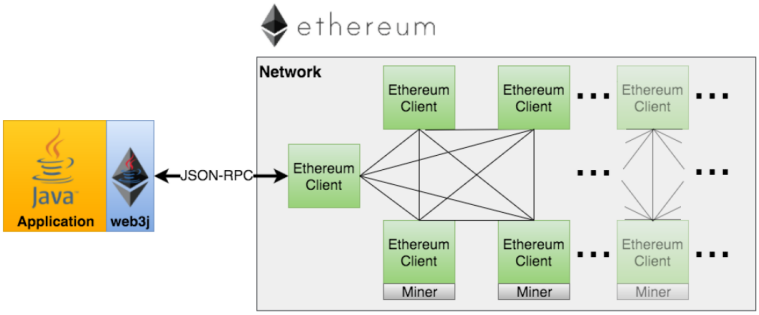


Figure 4.5: Web3J API [web3j, 2020].

4.3.2 Fabric SDK

Fabric Gateway SDK provides a set of interfaces that enable a Java application to interact with a Fabric blockchain network through the RPC protocol (Figure

4.6). The SDK also provides the means to execute user chaincode, query blocks and transactions, and keep track of channel events [IBM, 2019]. This library works with the Fabric client to create and manage a blockchain network, as well as interface with the chaincode that has been deployed.

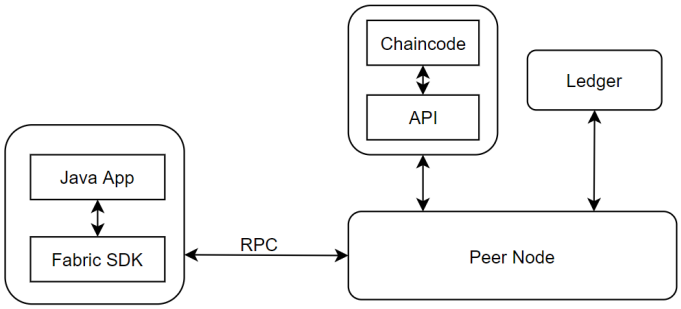


Figure 4.6: Fabric API.

4.4 Comparison

Table 4.6 presents the Geth, Besu, and Fabric clients and compares each supported consensus algorithms, the available development languages, and the compatible interface libraries. All of the selected clients accomplish the criteria expressed in Section 1.3.1. While Geth and Besu clients are viable solutions for deploying a private Ethereum network, Fabric is a modular platform with CFT based consensus mechanisms. When comparing Geth with Besu, the last has a few key points that puts it in advantage. Besu, for example, implements the EEA specification, is licensed under the Apache 2.0 license, and supports the EthSigner technique to improve user identity validity. When comparing Besu with Fabric, both allow for the plugging of consensus mechanisms and give solutions with modularity and scalability. However, for Fabric, the CFT algorithm requires fewer nodes to manage the same number of faults as the BFT algorithm employed by Besu or Geth. Although no statistics on transaction rates for these three customers could be uncovered, when deploying a private solution, both of them may approach hundreds of transactions per second. However, the Fabric client has apparently the highest throughput. Also, according to Ferris [2019], with just a few endorsing nodes, a network can process hundreds of transactions per second. Both smart contract technologies, solidity and chaincode, utilise hashmaps and a key-value format in their data structures.

Table 4.6: Clients Comparison.

Client	**Geth**	**Besu**	**Fabric**
Platform	Ethereum	Ethereum	Hyperledger
Consensus	PoW, PoA	PoW, IBFT, PoA	Raft, Kafka, Solo

Smart Contract Language	Go, Solidity	Solidity	Golang, Java, NodeJs
Smart Contracts	Yes	Yes	Yes
API	Web3J	Web3J	Fabric SDK

4.5 Summary

This chapter introduced and compared a number of blockchain platforms, as well as their clients and development libraries. The targeted solution can be implemented using any of the available alternatives. The Fabric client, on the other hand, distinguishes out for its higher modularity, more available information, and the strongest development community. This could be useful not just for the project's development, but also for its long-term upkeep. The Fabric client, along with the Fabric SDK, is thus chosen as the blockchain network's client and interface library. For the ordering service, the network should employ the Raft method.

The solution offered for establishing and connecting the blockchain network with the CloudAnchor platform is discussed in the following chapter.

Chapter 5

Solution Proposal

This chapter describes the network modules and network architecture in order to show the solution model for the given problem.

5.1 Solution Requirements

The CloudAnchor platform creates business entities that must be kept permanently and securely. Blockchain technology will be employed to do this.

The business entities, as well as the network actors who manage them, must be recognised before it can be implemented. The user stories, which describe the behaviours that the system must implement, must then be defined.

5.1.1 Business Entities

SLA entities and trust entities are the two types of business entities that need to be stored on the blockchain.

There are bSLA and rSLA/cSLA contracts under the SLA entities. bSLA contracts are established between the platform and the businesses within the platform Contract layer. rSLA and cSLA contracts are established between businesses when they rent or consume resources, also under the same layer.

The T&R module is used by the platform for the trust entities type to analyse the life-cycle of each celebrated rSLA and calculate reputation grades based on the results of these contracts. Each negotiation produces two sorts of reputation grades: self trust and partner trust. The self trust represents a business's trust in all of its dealings with other businesses, while the partner trust indicates a business's trust in a specific partner. The level of business trust influences whether a company pays more or less to negotiate on the platform. Higher levels

of trust indicate that a company may be trusted and hence receives cheaper brokerage costs.

To summarise, there are four business entities: bSLA contracts, rSLA/cSLA contracts, self trusts and pair trusts.

5.1.2 CloudAnchor Network Actors

The network actors are the software agents that will be used to implement the smart contract logic and are hosted in various platform levels. When company A agrees to provide a service to company B, for example, the software agents representing both organizations work together to create the service agreement. Following the completion of the agreements, the contract data should be addressed to the blockchain by the respective contract agents.

Both bSLA and rSLA/cSLA entities are managed only at the Contract layer. The bSLA contracts are established at the *AgSlaLayer* agents. In the case of rSLA/cSLA contracts, different contract versions are generated for both the consumer and provider identities by the *AgSlaConsumer* and *AgSlaProvider* agents.

Agents from the Enterprise layer and the Contract layer handle the trusts. At the Enterprise layer, the self and partner trusts of a business are updated by *AgEnterpriseTrust*, *AgEnterpriseProvider* and *AgEnterpriseConsumer* agents. Then, at the Contract layer, the *AgSlaLayer* agents consult the values of the business trusts to propose suitable negotiation terms to the businesses.

Figure 5.1 lists the network actors operating each type of business entity.

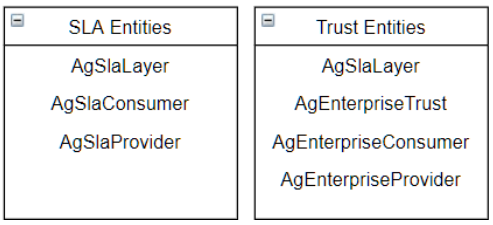


Figure 5.1: Network Actors.

5.1.3 Behaviour User Stories

The behaviour user stories are short functional descriptions that describe the features that will be implemented in the system from the perspective of the user. The network actor and the business entity to be operated must be included in the descriptions. These features are then utilized to drive development and assess the implementation's success.

Table 5.1 lists the behaviour user stories to be implemented.

Table 5.1: Required Behaviours

Number	Name	Description	Priority
US1	Create bSLA	As an AgSlaLayer agent, I wish to create a new bSLA contract whenever the negotiation is successful	High
US2	Renegotiate bSLA	As an AgSlaLayer agent, I wish to update an existent bSLA contract whenever the contract is successfully renegotiated	High
US3	Create/ Update rSLA	As an AgSlaProvider or AgSlaConsumer agent, I wish to insert/update a rSLA each time a contract suffers an update or is established	High
US4	Create/ Update cSLA	As an AgSlaProvider or AgSlaConsumer agent, I wish to insert/update a cSLA each time a contract suffers an update or is established	High
US5	Register Contract Success	As an AgSlaProvider or AgSlaConsumer agent, I wish to register the success of an existent rSLA or cSLA contract	High
US6	Register Contract Failure	As an AgSlaProvider or AgSlaConsumer agent, I wish to register the failure of an existent rSLA or cSLA contract and to register the faulty contractee	High
US7	Update Entity Self Trust	As an AgEnterpriseTrust or AgEnterpriseProvider or AgEnterpriseConsumer agent, I wish to update my own trust (and reputation)	High
US8	Update a Pair Trust	As an AgEnterpriseProvider or AgEnterpriseConsumer agent, I wish to update the trust (and reputation) of my contract partner (pair) based on the final status of a joint contract	High
US9	Read Trust	As an AgSlaLayer agent, I wish to read from the blockchain the trust of a partner when negotiating a bSLA contract with a business identity	High

5.2 Solution Design

The solution must be developed in such a way that all of the components can work together as a single system. CloudAnchor platform agents must be able to transmit requests to Fabric network nodes in the most efficient manner possible.

The CloudAnchor platform and the Fabric network can be connected directly or through an external client that acts as the platform gateway. For this project, it was decided to go with the second alternative, in which the platform is enhanced with an API module that serves as the platform's gateway.

The logic separation allows the gateway implementation to have the least amount of impact on the CloudAnchor platform. As a result, it provides the option of hosting the CloudAnchor platform on a separate machine from the API, potentially improving the efficiency of hardware resource management.

Figure 5.2 presents the system modules designed for the implementation.

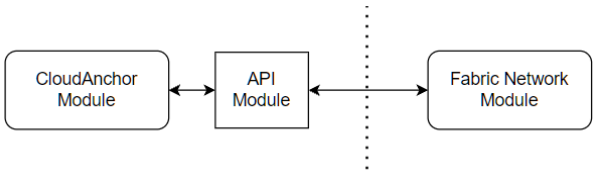


Figure 5.2: Solution Modules.

5.2.1 Fabric Module

The Fabric network module, which represents the Fabric blockchain, has as its major purpose offering an oriented interface for receiving and processing platform requests in the shortest period of time. The network architecture must be well-established to minimise processing time constraints. According to Thakkar et al. [2018], there are two sorts of Fabric blockchain resources that can boost network performance and availability while implementing the network: a large number of endorsement peers and adequate channels to process transactions. Thus, the orderer node selection and configuration may also contribute to this end. Metrics like the time it takes to generate a new block on the ledger, the size of the batch to send in each transaction, and the maximum number of transactions per block are all factors that must be appropriately established, otherwise the network performance would suffer. However, because more resources imply more costs, the goal was to produce the best results with the fewest resources possible.

Table 5.2 presents the configuration made for this network.

Table 5.2: Network Metrics

Name	Description	Logic	Value
Endorsers	The minimum number of endorsement peers.	At least one endorser per smart contract. Condition: More endorsers bring higher throughput.	8 peers
Orgs	The number of organisations.	Organisations can be high volatile and be used to process groups of smart contracts. For easier scalability and management, an organisation can address a group of two peers (8 endorsers / 2 = 4 orgs).	4 orgs
CA	The number of CA.	The Hyperledger recommends to have one CA for the orderer node, and at least one CA per each organisation. Since there are only two peers in each organisation, it is not necessary to have more than one CA per organisation.	5 peers
MSP	The number of MSP.	Each organisation will have its own MSP to manage the identities and certificates of its user.	4 MSP
Channels	The number of communication channels.	Since multiple channels increases the throughput, the network will have 3.	3 channels

Block Timeout	The time for a new block to be generated.	Networks that are constantly being updated and queried should generate more blocks to avoid updating the same key twice.	0.5 s
Block Size	The amount of transactions per block.	It depends on the network needs and the block timeout. The block size can either send 20 rSLA or 100 bSLA.	100 transactions
Message Size	The preferred size of the transaction to be processed.	Transactions that match the message size will be swiftly processed. An rSLA can take up to 350 B, so with a 15 kB message it can process up to 43 rSLA in each batch, or up to 190 80 B bSLA.	15 kB
Orderer Type	The type of ordering service.	Raft is the newer ordering service.	Raft
State Database	The database having the current state of blockchain.	According to [Thakkar et al., 2018], LevelDB performs faster than CouchDB in write operations. CouchDB offers complexity and pagination on querying. For this work, it is preferred writing speed over complex reading queries.	LevelDB
Gossip Protocol	A protocol used by the endorsement peers to spread the updates to other peers.	Because many requests must be handled at the same time, the peers will receive separate requests and may construct blocks with the same id. They must have the state transfer flag activated in order to recover old information from other peers.	State Transfer Enabled

5.2.2 API Module

To have the business logic sent to the blockchain, the CloudAnchor platform must use the API. The functionalities to be accessible via the API interface are the user stories defined in the preceding section.

The API must be updated to match the Fabric network parameters in order to maintain high performance and avoid requests stacking up. As the preferred batch size is defined to 15 kB, the time to process 2 kB or 12 kB is similar. Therefore, the batch requests must be optimised to use its maximum capacity of 15 kB. If the request size is larger than the intended batch size, the processing time will be significantly longer, resulting in a performance degradation.

The requests received by the CloudAnchor platform are added to request queues. This allows write requests to be stacked and batches including numerous requests to be prepared, improving network throughput capacity. Additionally, using queues frees up CloudAnchor request connections by marking requests as successful as soon as they are added to the queues.

The API project is a Java 11 Maven application, supported by the Spring Boot framework v2.3.4 tool, and leveraged by the Apache Tomcat HTTP web application server v9.0. It is composed by four layers, namely, (i) the controllers layer, (ii) the services layer, (iii) the model layer, and (iv) the repository layer. The layers are represented in Figure 5.3 and their objectives are described below:

- The controller layer is the entry point to the API. It is the layer where the HTTP requests are mapped, such as GET, POST, PUT, or DELETE.
- The service layer binds the controller layer to the repository layer. If the API supports more than one repository, switching between them is as simple as altering the interface in the service layer. This implementation follows the principle of the inversion of dependency.
- The model layer is where the business logic is shaped to create the data objects. The objects can be serialised to be transported through the network.
- The repository layer is the layer that accesses the blockchain persistent data. It implements the Fabric SDK gateway-logic to invoke the chaincode API methods available in the blockchain endorsement peers.

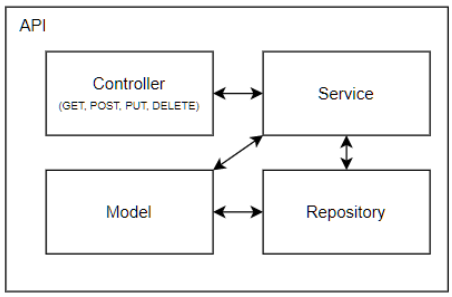


Figure 5.3: API Architecture.

5.2.3 CloudAnchor Module

Apart from usual execution, the platform must learn how to connect with the API module in order to transmit contracts to the blockchain.

A gateway module will enable the agents to easily access the Create, Read, Update and Delete (CRUD) operations available at the API. This module must also share a set of message and model classes with the API so that it could standardise the requests content.

The platform must include three gateway classes that allow agents to send messages to the API based on the types of business entities. As a result, the following classes will need to be created:

- bSLA gateway class - this class must map the received requests to the API endpoints of creating/updating and reading bSLA entities.
- rSLA gateway class - this class must map the received requests to the API endpoints of creating/updating and reading rSLA entities.

- Trust gateway class - this class must map the received requests to the API endpoints of updating and reading own trust and pair trust entities.

5.2.4 Network Architecture

The whole solution will be made up of three parts: the blockchain network, the gateway API, and the CloudAnchor platform integration via the gateway API, as previously stated. These modules work together to create a single output.

Figure 5.4 presents the system architecture where it shows the connection between the CloudAnchor platform and the API Controller layer, and the usage of the fabric-gateway-java to connect the API Repository layer to the blockchain.

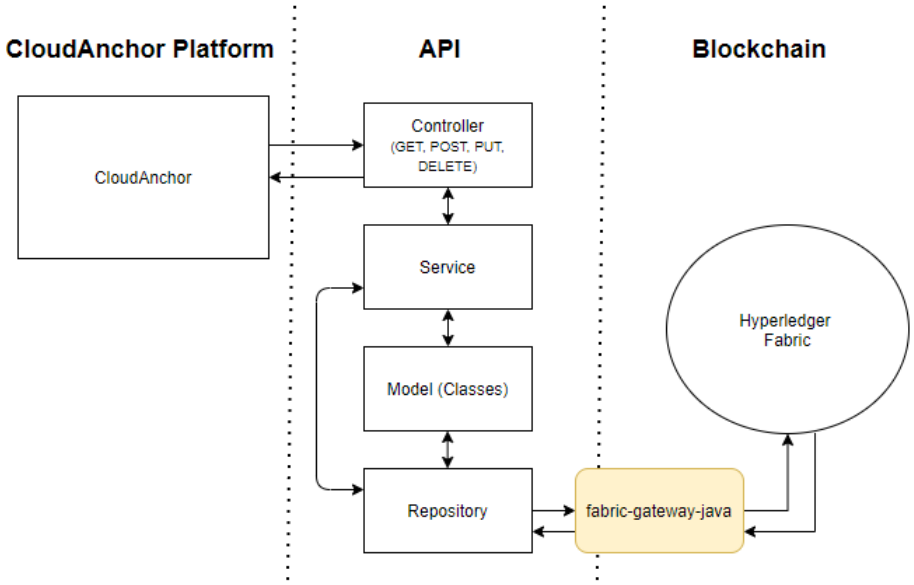


Figure 5.4: Solution Architecture.

5.3 Summary

From an end-to-end perspective, the proposed approach aims to find the optimum technique to ensure an efficient message flow. It includes the specification of the Fabric network module, the Java REST API, and the CloudAnchor platform and Fabric network interaction logic.

The Fabric network deployment and architecture, as well as the system module integration, will be discussed in the following chapter.

Chapter 6

Blockchain and CloudAnchor

This chapter presents the solution implementation, detailing the blockchain network, the Java REST API, and the CloudAnchor integration with the network parts.

6.1 Fabric Network

The Fabric network may be deployed and managed with the help of a collection of tools and configurations. The nodes must be configured, deployed, and maintained in order for the blockchain to function properly for the chaincodes to be instantiated on it.

A collection of configuration options for this blockchain deployment was described in the preceding Chapter. It will now be illustrated how the network components are put together and how the system works.

6.1.1 Nodes Architecture

As defined in Table 5.2, the network is composed by eight endorsement peers, three communication channels, an orderer node, five CA and four MSP. Each organisation has its own MSP that handle its users identities and the certificates. The organisations peers use the channels to reach consensus about the transaction proposals. Each channel is used for submitting different chaincode entities. For example, one channel is dedicated for bSLA entities transactions, while other channel is for rSLA/cSLA transactions, and the last one for the trust types. This helps to manage the channel connections at the API and to submit requests at different rates for each channel, preventing key collisions and other kinds of concurrency issues.

The Fabric network architecture is presented in Figure 6.1.

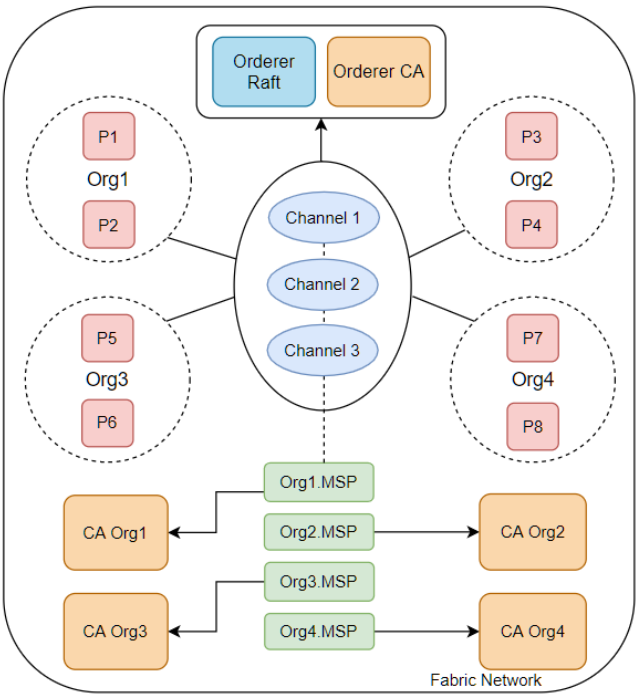


Figure 6.1: Fabric Network Architecture.

6.1.2 Deployment

The network nodes could be deployed on disk or in Docker containers. The second option was chosen for this project.

The nodes are deployed in Docker containers using the Hyperledger Fabric container images when the ledger is first started. Each type of network node is instantiated using the images presented in Table 6.1.

The containerised nodes listen to a configured machine port so that they can be addressed on the network. In total, there is a container for each endorsement peer, for the orderer node and for each of the five CA.

Table 6.1: Docker Images

Node Type	Image Name	Version
Endorser	hyperledger/fabric-peer	1.4.1
Orderer	hyperledger/fabric-orderer	1.4.1
CA	hyperledger/fabric-ca	1.4.7

6.1.3 Project Structure

Hyperledger provides a few sample projects for users to try out the technology and to use as a starting point for their own use cases. A sample project named *fabric-samples* [Hyperledger, 2021] was used to start the blockchain development. This project was first used to test the functionalities of the Fabric blockchain, and then it was adapted to this use case.

The configuration files came with default values for the Fabric sample network. Adapting the configuration to fulfill this network use case included adding more organizations and peers to the network, adding more channels to the network, and changing the default values for batch requests and ledger block creation.

The updated version defines four directories, namely the *bin*, the *chaincode-bsla*, the *config* and the *test-network*.

- *bin* - allocates the executable programs necessary to operate the Fabric network nodes, such as *peer* or *orderer*, which are related to the network endorsement peers and the orderer node.
- *chaincode-bsla* - contains chaincode contract classes and data type classes.
- *config* - contains configuration files such as *core.yaml* or *orderer.yaml*, used to configure the network nodes and the network metrics.
- *test-network* - contains folders such as *scripts* for operating the network, *docker* containing docker configuration files, *organisations* for auto-generating the crypto material for the orgs and the CA authorities, or *channel-artifacts* which contain channels configuration files used by the Fabric network.

6.1.3.1 Network Commands

The network leverage begins with the node containers being started, the channels being created, and the chaincodes being deployed to the channels. These operations are done by executing the respective commands specified in the script files, located under the *test-network* directory. The data in the ledger must always be preserved during the deployment, maintenance, and updating of the network. A network restart is rare. If the chaincode needs to be redeployed for whatever reason, it may be done on the fly without causing the service to go down. In addition, if a peer goes down or begins to malfunction, the docker container that hosts the peer can be restarted without affecting the other peers.

Table 6.2 describes a set of operations that are used for managing the ledger. On it, processes like starting and halting the network, deploying and updating the chaincode in peers, and others are contained.

Table 6.2: Network Commands

Name	**Description**	**Usage**
Start the Network	Bring up fabric orderer and peer nodes.	./network.sh up
Create Channels	Bring up fabric network with the three channels.	*./network.sh createChannel*
Deploy Chaincode	Deploy the chaincode in all network peers.	*./network.sh deployCC -ccn bsla -ccv n -ccs n, (n=1,2,3,.., n)*
Update Chaincode	Deploys a newer/updated version of chaincode.	*./network.sh deployCC -ccn bsla -ccv (n+1) -ccs (n+1)*
Restart the Network	Restarts the ledger without clearing the data.	*./network.sh restart*
Reset/Clear the Network	Clears the network and removes docker containers.	*./network.sh down*
Query Chaincode from CLI	Query chaincode from command line.	*peer chaincode invoke –args CC_Name:Method*

Below are the steps used for booting up the network.

1. Network up - brings up the node containers and generates the crypto material for the peers, such as certificates and identities.
2. Create channels - deploys the channels so that the endorsement peers can join.
3. Deploy chaincode - compiles and installs chaincode on peers.
4. Update chaincode - deploys a new version of chaincode.

Then, when the blockchain is up and running, the applications can start using it.

6.1.4 Chaincode Entities

A chaincode is a piece of software that can be used to represent business logic and provide a set of activities that network peers can instantiate and initialise. The entity model is maintained in the blockchain as a chaincode data type, whereas the chaincode contract provides a set of methods to operate the entities.

To be handled using chaincode contracts, the business logic in the CloudAnchor platform must be modelled using chaincode data types. This way, the API can map the platform requests to the defined chaincode operations. Table 6.3 lists the chaincode contracts and their data types.

Table 6.3: Chaincode Mapping.

Entity Type	Business Logic	Contract	Data Type
SLA	Resource	RSlaContractTransfer	RSlaPair
	Brokerage	BSlaContractTransfer	BSlaContract
Trust	Own Trust	OwnTrustTransfer	OwnTrust
	Pair Trust	PairTrustTransfer	PairTrust

- bSLA entities are managed by the BSlaTransfer contract and are defined by the BSlaContract data type.
- rSLA entities are managed by the RSlaPairTransfer contract and are defined by the RSlaPair data type.
- Self trust entities are managed by the OwnTrustTransfer contract and are defined by the OwnTrust data type.
- Pair trust entities are managed by the PairTrustTransfer contract and are defined by the PairTrust data type.

6.1.4.1 Chaincode Data Types

The data types presented in Table 6.3 are the structures to be held in the ledger. Each business entity defines its own data type to hold the relevant data. In this work are defined four data types, naming PairRSla, BSlaContract, OwnTrust and PairTrust, for the rSLA, bSLA, self trust and pair trust respectively (Figure 6.2).

- The *PairRSla* specifies the contract identifier, the provider and consumer contract part, the business entity identifiers, the contract type, whether it is a cSLA or a rSLA, the start and end dates of the contract, and the state of the contract, which can be in one of the following states: empty (E), created (C), active (A), success (S), failed by consumer (M), or failed by provider (P).
- The *BSlaContract* contains the contract identifier, the business entity identifier, the type which distinguishes the bSLA from the bcSLA contracts, the fee, which is the amount the entity has to pay to the platform, the start and end dates of the contract, and the state of the contract, which can be in one of the following states: initialised (I), reinitialised (R), active (A), success (S), failed (F), or terminated (T).
- The *OwnTrust* defines the trust identifier, the business entity identifier, the total number of negotiations, the total number of successful negotiations, and the total number of failed negotiations.

- The *PairTrust*, also defines the trust identifier, the business entity identifiers, the number of successes and failures from rSLA negotiations, and the number of successes and failures from coalition negotiations of each entity.

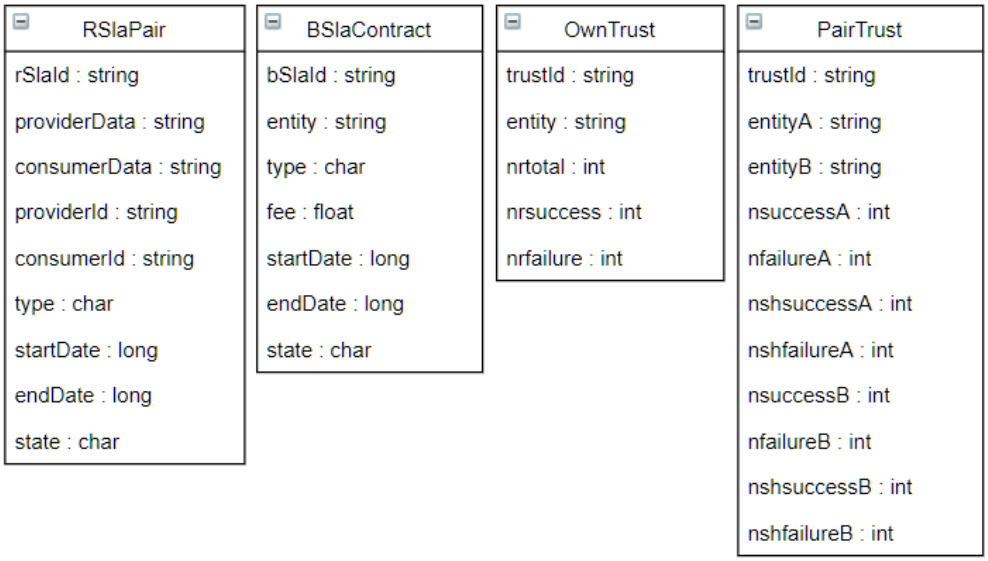


Figure 6.2: Chaincode Entity Data Types.

6.1.4.2 Chaincode Operations

For the API to perform read/write operations associated with the entities data types, the chaincodes must expose an interface called contract. There are two types of contracts entities: SLA contracts and trust entities. The SLA type is supported by the RSlaPairTransfer and BSlaTransfer contracts while the trust type is supported by the the OwnTrustTransfer and PairTrustTransfer contract. Each one contains the necessary operations to manage the respective business entities, however, these operations are differently implemented in each class.

Table 6.4 gathers all the available operations, with each operation description and the operation access mode.

Table 6.4: Smart Contract Operations.

Chaincode	Transaction	Description	Access Mode
SLA	HealthCheck	Checks if chaincode is initiated.	Read
	CreateOrUpdate	Creates or updates an SLA contract.	Read
	ReadSLA	Returns an SLA contract by ID.	Read
	GetAllSLA	Returns all SLA contracts.	Read
	GetByEnterprise	Returns all SLA of an enterprise.	Read
	UpdateStatus	Updates the status of an SLA.	Write
	ValidateContracts	Synchronises the SLA state.	Write
	CountContracts	Counts the number of contracts stored in the ledger.	Read

Trust	HealthCheck	Checks if chaincode is initiated.	Read
	Update	Updates a trust (own/pair).	Write
	ReadTrust	Returns a trust by ID.	Read
	GetAllTrust	Returns all trust contracts.	Read
	CountTrusts	Counts the number of trusts stored in the ledger.	Read

Among the operations presented, there are also system validation and debugging operations. For example, the *HealthCheck* operation can be used to identify if the chaincode is initiated. The *ValidateContracts* validates the contracts state whether they need to be updated. Figure 6.3 presents the four contract available operations according to their implementations.

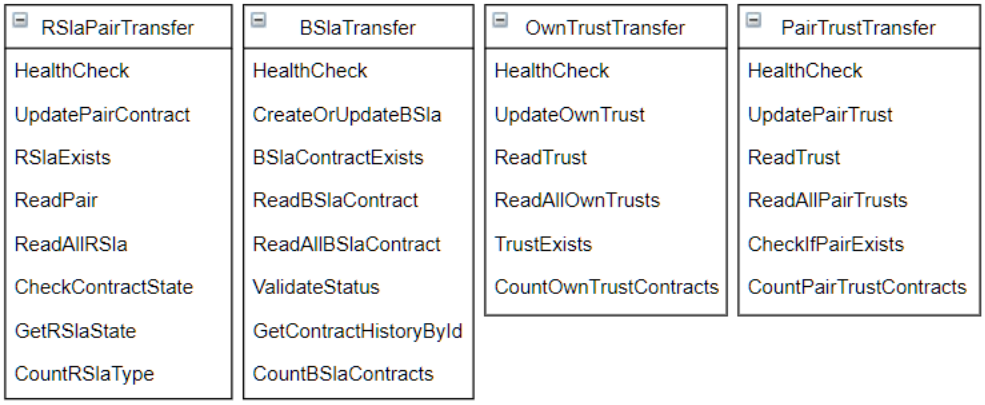


Figure 6.3: Chaincode Entity Contracts.

6.2 API Services

The API is composed of a set of independent services for dealing with the different business entities requests. Its operation consists of verifying, packaging, and delivering the CloudAnchor requests to the blockchain peers. The API services only act when they receive an input (a request), so their architecture follows a request-driven pattern.

The services assume a vertical structure, composed by classes from the different API layers. The communication is isolated, which means that the services do not interact with each other. For example, the self trust entity requests that are sent to the trust controller pass through the trust service before they reach the self trust repository. The operation response is sent in the opposite way to the controller to deliver it back to the CloudAnchor platform.

6.2.1 Controller Layer

The *Controller* layer defines three different controller classes: the *BSlaController*, the *RSlaController*, and the *TrustController*.

- *BSlaController* - defines the operations for dealing with the bSLA entities. These methods are mapped to the "/api/v1/bslacontract" path.
- *RSlaController* - defines the operations for dealing with the rSLA entities. These methods are mapped to the "/api/v1/rslacontract" path.
- *TrustController* - defines the operations for dealing with the self trust and pair trust entities. These methods are mapped to the "/api/v1/trust" path.

Figure 6.4 presents the controllers available methods for each type of class.

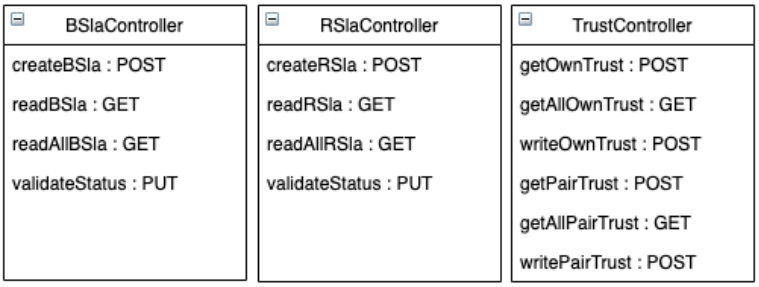


Figure 6.4: API Controllers.

6.2.2 Service Layer

At the *Service* layer there are also three services defined: the *BSlaContractService*, the *RSlaContractService*, and the *TrustService*.

- *BSlaContractService* - establishes the bridge between the *BSlaController* to the bSLA repository.
- *RSlaContractService* - establishes the bridge between the *RSlaController* to the rSLA repository.
- *TrustService* - establishes the bridge between the *TrustController* to the self trust and pair trust repositories.

These classes map the controllers operations to the repositories operations.

6.2.3 Model Layer

The *Model* layer contains two types of classes: the entity classes and the message classes. The entity classes are the entity data types which contain the necessary data for defining an entity. The message classes are standard structures shared between the system parts that transport the necessary information for creating/updating the business entities. Its objective is to guarantee that a fixed request structure is used for transmitting the data through the network.

There are two entity classes: the *BSlaContract* and the *RSlaEntityContract*. The *BSlaContract* entity defines the data type to be stored at the blockchain, similar to the chaincode data type *BSlaContract* previously presented. The *RSlaEntityContract* is the business rSLA contract version to be stored at the *RSlaPair* chaincode data type at the *providerData* and *consumerData* attributes. For the SLA typed entities, the messages contain a business entity and other relevant attributes, while the trust messages only define attributes (Figure 6.5). All of this data is generated at the CloudAnchor platform.

- *BSlaContractMessage* - defines a bSLA entity and a flag to indicate whether it is a renegotiation or not.
- *RSlaPairMessage* - defines the rSLA identifier, the *entity* to indicate if it is the consumer (C) or the provider (P) contract part, the entity identifier, the contract version object, the dates and the status which can be defined as empty (E), failed by provider (P), or failed by consumer (M).
- *OwnTrustMessage* - defines the entity identifier, the negotiation result, and the multiplier to know how much resources have failed (0) or succeeded (1).
- *PairTrustMessage* - defines the two business entities, an indicator that distinguishes whether is a coalition contract or not, the negotiation result, the multiplier, and the parts to be updated.

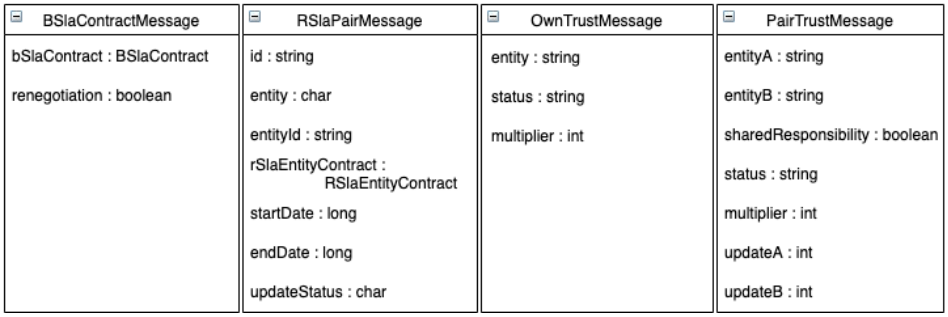


Figure 6.5: Message Structures.

6.2.4 Repository Layer

The *Repository* layer classes have two different functions: to receive the messages delivered by the service classes and to prepare the messages to be submitted to the blockchain endorsement peers. Therefore, the repository classes extend the Java *Runnable* class, which allows to run a second thread in the class so that the responsibilities can be separated.

These classes use message queues to stack the service request messages. It allows the service classes to drop the messages in the queues and to proceed with their execution. The second thread verifies the presence of new messages in the queues and starts preparing the batch requests. A good operation rate for the API is to keep the messages flowing and not have them stack up for too long in queues. The more messages it can send in a single transaction, the better, as long as it does not exceed the batch preferred size limit, because it would slow down the execution.

The existent classes at the *Repository* layer are: the *BSlaContractDataAccessRepository*, the *RSlaContractDataAccessRepository*, the *OwnTrustDataAccessRepository*, and the *PairTrustDataAccessRepository*.

- The *BSlaContractDataAccessRepository* class manages the requests concerning the bSLA entities and uses the Fabric *bslachannel* for submitting the requests.
- The *RSlaContractDataAccessRepository* class manages the requests concerning the rSLA entities and uses the Fabric *rslachannel* for submitting the requests.
- The *OwnTrustDataAccessRepository* class manages the requests concerning the self trust entities and uses the Fabric *trustchannel* for submitting the requests.
- The *PairTrustDataAccessRepository* class manages the requests concerning the pair trust entities and uses the Fabric *trustchannel* for submitting the requests.

6.2.4.1 Peers Connection

The API has to build a connection to the Fabric peers so that it can submit the transaction proposals. The connection to a Fabric peer is established by sending an authenticating request with the generated identity for the respective org. These identities are generated once after the application boot, and are stored in the identity wallet directory under the API project. After the peers accept the connection, the API selects the channel it wants to use for the identity and

prepares the batch for sending in the transaction proposal. The peer's connection is kept alive to maintain a fast operation rate for future transactions.

6.2.4.2 Retry Mechanism

Another thing to consider about the batches is the possibility of an error occurring during the submission, such as a key collision. As the database works in a key-value scheme, a key collision may occur when different peers simultaneously submit an alteration for the same key. These cases are highly likely to happen in high-throughput networks, where the messages are sent concurrently and shortly time-spaced. The blockchain blocks these events from happening to avoid double-spending situations. When it occurs, the API has a retry mechanism which allows it to detect whether a transaction has been declined and it retries the submission of the batch request. The batch submission only ends after the request has been accepted on the blockchain, or if exceeds the predefined number of retries configured at the API. This way, there is a possibility for the API to deal with those unexpected events and to try to guarantee the data recovery.

6.3 CloudAnchor Integration

For the requests to be addressed to the API endpoints, the platform defines an internal module to serve as the requests gateway. This module, named *Blockchain*, provides the necessary tools for the CloudAnchor agents to easily send the data requests to the API. It defines three distinguished class types: (i) the API request gateways; (ii) the message classes; and (iii) the entity models. The message classes and the entity models are the same as the classes presented in the API Services Section. The API request gateway classes map the requests to the API controller endpoints. There are three request gateway classes: the *BSlaContractsHttp*, the *RSlaContractsHttp*, and the *TrustHttp*.

- The *BSlaContractsHttp* class contains the methods for creating and reading the bSLA entities mapped to the *BSlaController*. These methods are invoked by the *AgSlaLayer* agents.
- The *RSlaContractsHttp* class contains the methods for creating and reading the rSLA entities mapped to the *RSlaController*. These methods are invoked by the *AgSlaConsumer*, and *AgSlaProvider* agents.
- The *TrustHttp* class contains the methods for creating and reading self trust and pair trust entities mapped to the *TrustController*. These methods are invoked by the *AgEnterpriseConsumer*, *AgEnterpriseProvider*, *AgEnterpriseTrust*, and *AgSlaLayer* agents.

The agents invoke the gateway methods directly as Java public static methods, which means that they can be accessed by other classes in the project without having to initialise a new object. Figure 6.6 presents the flow chart of a bSLA create/update operation request.

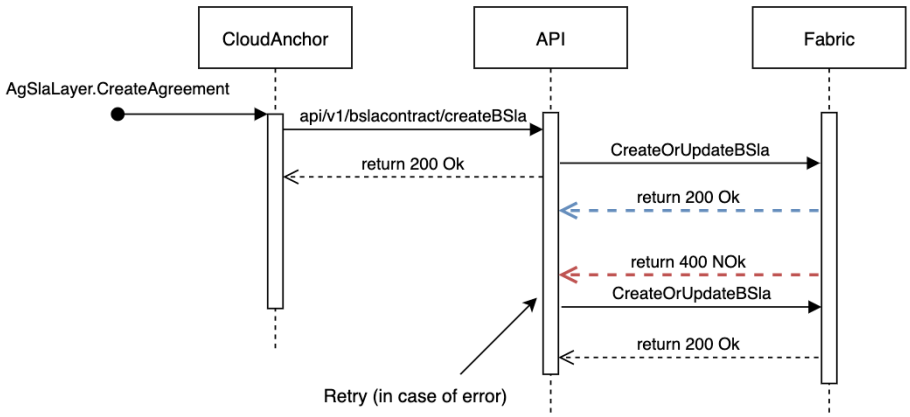


Figure 6.6: Create/Update bSLA.

6.4 Summary

The total solution implementation was provided in this chapter, which included the fabric network composition, API services, and the CloudAnchor platform integration module. It explained the Fabric network deployment method, as well as the management tools and chaincode classes. Then it presented the API layers responsibilities and classes. Finally, it demonstrated the CloudAnchor module, which allows agents to send requests directly to API endpoints over a direct connection.

The tests performed on the developed solution will be presented in the following chapter.

Chapter 7

Experiments and Results

The experiments described in this chapter were carried out to validate and measure the impact of the CloudAnchor platform's adoption of smart contracts.

7.1 Blockchain Impact

These tests are intended to assess the impact of blockchain integration on the CloudAnchor platform's operation. With and without smart contracts, the purpose is to compare platform execution time and generated data performance indicators. While the blockchain integration is likely to effect run-time latency, or overall processing time, the other business logic indicators should only show minor differences.

These tests are based on Veloso et al. [2020] study, which presented a set of KPI for monitoring platform outputs such as global turnover, expenses, and average negotiation time per resource. The global turnover represents the total value paid for the supplied resources, including resource provisioning, brokerage and coalition fees, within the platform. The global loss corresponds to the brokerage and coalition fees paid regarding faulty services. These tests will be run in a VM that hosts the CloudAnchor platform project, the Fabric API, and the Fabric ledger all at the same time. This means that the projects are vying for the same system resources, causing overall performance to suffer. The virtual machine used for these tests has a quad-core CPU with 2 threads per core, 20 GiB of RAM, and a 70 GiB disk for storage.

The experiments involve thirty consumers and thirty providers with renegotiation and brokerage fees, where both consumers and providers are grouped into three sets of 10 businesses with 60 %, 80 %, and 100 % of average SLA enforcement trustworthiness ($\overline{T_{\mathrm{E}}}$). This means that a business that has 60 % of $\overline{T_{\mathrm{E}}}$

complies with its contract obligations on approximately 60 % of all the rented resources and fails on about 40 %. The platform charges higher brokerage fees to businesses which fail more.

To analyse the platform's behaviour, a series of four tests was run. These tests are summarised in Table 7.1. The platform first negotiates single resources for a short and long duration, then federated resources for a short and long period as well. A month is equal to a short term lease, and five months is equal to a long term lease. Each provider has 1000 single resources, but the platform can negotiate 1000 single resources or forty bundles of 25 federated resources depending on the test. The metrics are collected after a five-month warm-up period to avoid a platform cold start. Short-term experiments last 10 months, while long-term experiments last 30 months, yielding 5 and 25 months of warm operation, respectively. Each of the scenarios is executed using the Trust and Reputation (T&R) module and bSLA renegotiation. Finally, all tests are redone with and without blockchain integration, and the outcomes of the two executions are compared to determine the influence of blockchain.

Table 7.1: Tests Scenarios

Scenario	**Businesses**	**VM/Business**	$\overline{T_{\mathrm{E}}}$ (%)
1	30 consumers 30 providers	1000 x 1 Short Term	10 x [60, 80, 100]
2	30 consumers 30 providers	1000 x 1 Long Term	10 x [60, 80, 100]
3	30 consumers 30 providers	1000 x 25 Short Term	10 x [60, 80, 100]
4	30 consumers 30 providers	1000 x 25 Long Term	10 x [60, 80, 100]

7.1.1 Real Set-up

The average number of SLA, VM and smart contracts with (w/) and without (w/o) using the blockchain network are presented in Table 7.2. Each experiment was carried out five times. The reported values are the average and standard deviation that were calculated. Table 7.3 shows that the impact of blockchain in these values is residual – lower than 1 %. This residual variance is expected, but not substantial, given that the platform algorithms are non-deterministic, meaning that they produce different results each time they are run. The conclusion is that the blockchain integration has little impact on these measures.

Table 7.2: Total of SLA, Smart Contracts and Faultless VM

Lease	Metric	Single Resources		Federated Resources	
		w/o BC	w/ BC	w/o BC	w/ BC
Short Term	bSLA contracts	60 ± 0	60 ± 0	60 ± 0	60 ± 0
	bSLA (renegotiated)	137 ± 1	136 ± 1	230 ± 2	228 ± 2
	rSLA contracts	$85\,512 \pm 394$	$84\,819 \pm 427$	$18\,589 \pm 168$	$18\,497 \pm 161$
	cSLA contracts			$81\,750 \pm 339$	$81\,309 \pm 321$
	Faultless VM	$75\,862 \pm 239$	$75\,218 \pm 256$	$80\,648 \pm 280$	$80\,004 \pm 266$
	bSLA Smart Contracts		136 ± 1		228 ± 2
	rSLA Smart Contracts		$84\,819 \pm 427$		$18\,497 \pm 161$
	cSLA Smart Contracts				$81\,309 \pm 321$
Long Term	bSLA contracts	60 ± 0	60 ± 0	60 ± 0	60 ± 0
	bSLA (renegotiated)	145 ± 1	144 ± 1	227 ± 2	228 ± 2
	rSLA contracts	$86\,838 \pm 390$	$86\,527 \pm 417$	$19\,012 \pm 172$	$19\,058 \pm 178$
	cSLA contracts			$81\,582 \pm 351$	$81\,672 \pm 383$
	Faultless VM	$76\,721 \pm 249$	$76\,601 \pm 263$	$80\,281 \pm 218$	$80\,466 \pm 235$
	bSLA Smart Contracts		145 ± 1		228 ± 2
	rSLA Smart Contracts		$86\,527 \pm 417$		$19\,058 \pm 178$
	cSLA Smart Contracts				$81\,672 \pm 383$

Table 7.3: Variation of SLA, and Faultless VM

Lease	Metric	Single Resources Δ (%)	Federated Resources Δ (%)
Short Term	bSLA contracts	0.0	0.0
	bSLA (renegotiated)	−0.7	−0.9
	rSLA contracts	−0.9	−0.5
	cSLA contracts		−0.5
	Faultless VM	−0.8	−0.8
Long Term	bSLA contracts	0.0	0.0
	bSLA (renegotiated)	−0.7	+0.4
	rSLA contracts	−0.6	+0.2
	cSLA contracts		+0.1
	Faultless VM	−0.2	+0.2

Table 7.4 displays the average negotiation latency ($\overline{\Delta t_N}$), average invitation latency ($\overline{\Delta t_I}$) and average transacted value per resource ($\overline{TV}$) with and without blockchain. While the average transaction value per resource stays unchanged, average invitation and negotiation times have increased significantly.

In the short term scenario, the overall negotiation time is 26 % higher for single resources and 68 % higher for federated, and the invitation time is 125 % higher for the single resources and 89 % higher for the federated.

In the long term scenario, the overall negotiation time is 26 % higher for single resources and 68 % higher for federated, and the invitation time is 125 % higher for the single resources and 89 % higher for the federated.

While the business KPI (volume of negotiated and flawless resources) is unaffected by blockchain, the data demonstrate that it has a significant influence on platform run-time. Furthermore, the parallel execution of the entire network on the same machine degrades run-time. The major bottleneck of these experiments was the limited CPU capacity of the VM, with a CPU usage always above 90 %. For a more conclusive evaluation, these experiments were repeated in a simulated distributed scenario.

Table 7.4: Average Latency and Average Transacted Value per Resource

Lease	**Metric**	**Single Resources**			**Federated Resources**		
		w/o BC	w/ BC	Δ (%)	w/o BC	w/ BC	Δ (%)
Short Term	$\overline{TV}$ (€)	30.73	30.65	−0.3	46.29	46.43	+0.3
	$\overline{\Delta t_N}$ (ms)	16.86	21.32	+26.4	9.96	15.85	+59.2
	$\overline{\Delta t_I}$ (ms)	3.87	8.70	+124.5	2.05	5.94	+89.4
Long Term	$\overline{TV}$ (€)	30.94	30.95	0.0	46.21	46.60	+0.8
	$\overline{\Delta t_N}$ (ms)	15.99	20.17	+26.2	9.41	15.79	+67.9
	$\overline{\Delta t_I}$ (ms)	3.77	8.69	+130.2	1.73	5.63	+225.1

7.1.2 Simulated Set-up

This test simulates the blockchain and CloudAnchor being physically separated. By avoiding the blockchain operations, it assumes that both components are hosted on separate machines. In this alternative set-up, the previous set of experiments were repeated, taking into consideration the platform's run-time and amount of API queries. It is possible to measure the overtime introduced by the blockchain by tracking the run-time and volume of requests made to the blockchain. The run-time due to blockchain should be the same as CloudAnchor's default run-time plus the time it takes to process API requests.

The API performs chaincode logic in new developed test classes and keeps the records in memory to simulate the blockchain network being hosted on a separate system. Only reading operations are sent to the blockchain because CloudAnchor waits for the results of reading queries.

7.1.3 Run-time

To negotiate the 30 000 resources, the platform sent nearly 120 000 requests to the API, taking approximately 122 min (Table 7.5). The corresponding operations were carried out in numerous threads asynchronously. This means that the total time should be divided by the number of CPU cores available at the VM, resulting in approximately 15 min per core. So, the overall tests execution had to take more 15 min with the blockchain module, when negotiating 30 000 resources.

Table 7.6 presents the results of 12 executions made in this set-up. Without the blockchain the platform takes between 30 min to 35 min, while with the blockchain it lasts between 45 min to 50 min. The run-time without the blockchain is approximately 15 min less than with blockchain. Therefore, it shows that the platform takes about [43 %, 50 %] longer to execute with blockchain when running on a VM with 4 CPU cores with 2 threads per core. This is significantly less than the results of the first set-up. Because the requests are auto-generated at a pre-determined frequency, these findings can be presumed to be linear. In a real-world scenario, however, if a large number of requests arrive in a short amount of time, performance may suffer, affecting run-time. The amount and qualities of available hardware resources have a significant impact on execution time. The lower the time it takes to process the same number of requests, the more actions are executed in parallel. More CPU capacity means less execution time.

Table 7.5: Blockchain Requests Load per 30 000 rSLA Contracts

Type of Contract	**Operation**	**Message Count**	**Time per Operation** (ms)
Self trust	Read	163	125
bSLA	Write	163	8
rSLA/cSLA	Write	40 831	150
Self trust	Write	36 695	15
Pair trust	Write	40 628	15
	Total:	118 480	**121 min 46 s**
	CPU core threads:	**8**	**15 min 10 s / thread**

Table 7.6: Execution Time

Test Type	**Time**
Without blockchain:	30 min to 35 min
With blockchain:	45 min to 50 min

7.2 Block Rate Impact

The blockchain setup was created with one of the worst-case scenarios in mind: many requests for the same contract in a short amount of time. However, in the real world, the rSLA contracts should not expire in 5 min nor 10 min, since their duration extends for months or even years. This is important to determine the pace at which the blocks are generated. In this work, blocks of 15 kB are generated every half second. In a year, at this rate, the blockchain would occupy up to 925 GB of disk – 15 kB × 2 block/s × 3600 s × 24 h × 365 d. If the blocks

are configured to be created every 2 s, the disk space would be reduced by factor of 4. This means that in a production environment, the impact of longer block production periods on API and Fabric network performance should be analysed.

7.3 Summary

The influence of the blockchain on the CloudAnchor platform execution was measured in this chapter. First, it was determined that the blockchain integration had no impact on the platform's business logic results. The network was then discovered as being delayed and bottlenecked due to a shortage of hardware resources. As a result, a second experimental setup was created in which only blockchain read activities were successfully conducted. The findings revealed that the blockchain's influence is less than that shown in the initial testing, amounting to a 50 % boost in run-time. Finally, with more and better CPU resources, the blockchain's influence might be lowered even further.

Chapter 8

Conclusions

This chapter details the contributions and benefits gained by the CloudAnchor platform as a result of the project's completion.

8.1 Outcomes

The adoption of blockchain technology with smart contracts has a number of implications for the CloudAnchor platform. It is evident that the blockchain enhances the platform's data security. Furthermore, it records previous changes to the recorded contracts, allowing the platform to track the evolution of contracts and business trusts.

However, the implementation of the blockchain also has its downsides. The addition of complexity to the network results in higher maintenance expenses, along with a run-time surplus of about 50 % longer.

When comparing the platform performance indicators with the prior version, they remain identical with the exception of the run-time. This means that the blockchain network is unaffected by the platform business logic.

To summarise, smart contracts provide extra run-time protection for important business logic involving potentially untrustworthy parties.

8.2 Improvements and Future Work

This solution can be enhanced and become more user-friendly. Here are some ideas about how to enhance the solution:

- Fabric network management - The network is currently deployed in docker containers, and all actions must be performed manually. As a result, a

graphical interface, such as Kubernetes (or a similar external API with a front-end interface), would allow the user to administer and monitor the network.

- CloudAnchor and Fabric - At the moment, the CloudAnchor platform only consults the blockchain to double-check the SLA and pairwise trust contracts. In the future the blockchain could be used as the unique contract storage database for the CloudAnchor platform activities.
- Self-update operations - The state of the contracts stored in the ledger could be updated using chaincode self-executing mechanisms. To update the contracts state, the mechanisms should automatically validate start and end dates, negotiation result and previous state.
- Fabric Dashboard - A dashboard could be added to easily monitor the contracts in the ledger, such as the SLA state and the inter-business trust.

Bibliography

Imran Bashir. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018. [cited on p. v, 5, 6, 26, 31]

Imed Bouchrika. Service models for distributed systems, 2013. URL `https://www.ejbtutorial.com/distributed-systems/service-models-for-distributed-systems`. [cited on p. v, 7]

Luís Paulo Reis. *Coordenação em Sistemas Multi-Agente*. PhD thesis, PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003. [cited on p. v, 8, 14, 16]

Hyacinth S Nwana. Software agents: An overview. *The knowledge engineering review*, 11(3):205–244, 1996. [cited on p. v, 9, 10, 11]

Katia Sycara. Intelligent agents. [Online] Available: `https://slideplayer.com/slide/5352764/`, 2016. [cited on p. v, 12]

Rafael Bordini, Alvaro Moreira, Renata Vieira, and Michael Wooldridge. On the formal semantics of speech-act based communication in an agent-oriented programming language. *J. Artif. Intell. Res. (JAIR)*, 29, 10 2011. doi: 10.1613/jair.2221. [cited on p. v, 13]

Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007. [cited on p. v, 9, 10, 11, 13]

Bruno Miguel Delindro Veloso. *Transacção de componentes multimédia suportada por agentes*. PhD thesis, Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto, 2012. [cited on p. v, 17, 38]

CNIP FIPA. Fipa contract net interaction protocol specification. [Online] Available: `http://www.fipa.org/specs/fipa00029/SC00029H.html`, 2002a. [cited on p. v, 18, 19]

ICNIP FIPA. Fipa iterated contract net interaction protocol specification. [Online] Available: http://www.fipa.org/specs/fipa00030/SC00030H.html, 2002b. [cited on p. v, 20]

Rafael Brundo Uriarte. *Supporting Autonomic Management of Clouds: Service-Level-Agreement, Cloud Monitoring and Similarity Learning*. PhD thesis, IMT Institute for Advanced Studies, 03 2015. [cited on p. v, 21]

David Creer. How to choose the best platform for your blockchain based system? [Online] Available: https://medium.com/gft-engineering/how-to-choose-the-best-platform-for-your-blockchain-based-system-8b9c57862225, Mar 2020. [cited on p. v, 23]

Anastasiia Lastovetska. Blockchain architecture basics: Components, structure, benefits & creation, Jan 2018. URL https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture. [cited on p. v, 25]

Lina.Network. Generations of blockchain (part1): Blockchain 1.0. [Online] Available: https://medium.com/@lina.network/generations-of-blockchain-part1-blockchain-1-0-ba6cd931cd30, 2019. [cited on p. v, 28]

Bruno Miguel Delindro Veloso. *Media Content Personalisation Brokerage Platform*. PhD thesis, University of Vigo, 2017. [cited on p. v, 38, 39, 41]

Flentas. Ethereum: Architectural overview. [Online] Available: https://www.flentas.com/ethereum-architectural-overview, 2019. [cited on p. v, vii, 47, 51]

Rob Dawson. Announcing hyperledger besu – hyperledger. [Online] Available: https://www.hyperledger.org/blog/2019/08/29/announcing-hyperledger-besu, Aug 2019. [cited on p. v, 48]

Hyperledger. Smart contracts and chaincode. [Online] Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/smartcontract/smartcontract.html, 2020a. [cited on p. v, 49]

Shyam Pratap Singh. Detail analysis of raft & its implementation in hyperledger fabric. [Online] Available: https://medium.com/@spsingh559/detail-analysis-of-raft-its-implementation-in-hyperledger-fabric-d269367a79c0, May 2019. [cited on p. v, 50]

web3j. web3j/web3j. [Online] Available: https://github.com/web3j/web3j, May 2020. [cited on p. vi, 51]

The Bitcoin News. The world's first public blockchain with secret contracts is making its case. [Online] Available: https://cryptonewsmonitor.com/2019/

07/30/the-worlds-first-public-blockchain-with-secret-contracts-is-making-its-case/, 2019. [cited on p. vii, 28]

A.D. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2011. ISBN 9781139470315. URL https://books.google.pt/books?id=G7SZ32dPuLgC. [cited on p. 5, 6]

Roberto Infante. *Building Ethereum DApps: Decentralized Applications on the Ethereum Blockchain*. Manning Publications, 2019. ISBN 9781617295157. URL https://books.google.pt/books?id=wdh2tgEACAAJ. [cited on p. 5, 32]

Sandeep. Network models peer to peer and client and server. [Online] Available: https://www.vskills.in/certification/tutorial/basic-network-support/network-models-peer-to-peer-and-client-and-server/, Jan 2013. [cited on p. 6]

Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999. [cited on p. 7, 16]

Avron Barr and Edward A Feigenbaum. The handbook of artificial intelligence. william kaufmann. *Inc., Los Altos, CA*, pages 163–171, 1981. [cited on p. 7]

Alan H Bond and Les Gasser. Distributed artificial intelligence, 1988. [cited on p. 7]

Edmund H Durfee. Distributed problem solving and planning. In *ECCAI Advanced Course on Artificial Intelligence*, pages 118–149. Springer, 2001. [cited on p. 7]

Keith S. Decker and Katia Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3):239–260, Nov 1997. ISSN 1573-7675. doi: 10.1023/A:1008654019654. URL https://doi.org/10.1023/A:1008654019654. [cited on p. 8]

Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1): 7–38, 1998. [cited on p. 8]

Henri Avancini and Analía Amandi. A java framework for multi-agent systems. *Electronic Journal of SADIO [electronic only]*, 01 2000. [cited on p. 8]

Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge & Data Engineering*, 1:63–83, 1989. [cited on p. 8]

Katia P Sycara. Multiagent systems. *AI magazine*, 19(2):79–79, 1998. [cited on p. 8, 9, 10]

Donna L. Hudson and Maurice E. Cohen. *Neural networks and artificial intelligence for biomedical engineering*. IEEE New York, NY, USA:, 1999. [cited on p. 9]

Giordano Lanzola and Harold Boley. Experience with a functional-logic multiagent architecture for medical problem solving. In *Knowledge media in healthcare: opportunities and challenges*, pages 17–37. IGI Global, 2002. [cited on p. 9]

Virginia Dignum and Frank Dignum. A logic of agent organizations. *Logic Journal of the IGPL*, 20(1):283–316, 2012. [cited on p. 9, 14]

Andrea Omicini and Agostino Poggi. Multiagent systems. *Intelligenza Artificiale*, 3(1-2):79–86, 2006. [cited on p. 10, 15]

Kim On Chin, Kim Soon Gan, Rayner Alfred, Patricia Anthony, and Dickson Lukose. Agent architecture: An overviews. *Transactions on science and technology*, 1(1):18–35, 2014. [cited on p. 10, 12, 13]

Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986. [cited on p. 12]

Anand S Rao, Michael P Georgeff, et al. Bdi agents: from theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995. [cited on p. 12]

Innes A Ferguson. Toward an architecture for adaptive, rational, mobile agents. *ACM SIGOIS Bulletin*, 13(3):15, 1992. [cited on p. 12]

Jörg P Müller, Markus Pischel, and Michael Thiel. Modeling reactive behaviour in vertically layered agent architectures. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 261–276. Springer, 1994. [cited on p. 12]

R Scott Cost, Yannis Labrou, and Tim Finin. Coordinating agents using agent communication languages conversations. In *Coordination of Internet agents*, pages 183–196. Springer, 2001. [cited on p. 14]

Michael N Huhns and Larry M Stephens. Multiagent systems and societies of agents. *Multiagent systems: a modern approach to distributed artificial intelligence*, 1:79–114, 1999. [cited on p. 14]

Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, page 456–463, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916743. doi: 10.1145/191246.191322. URL `https://doi.org/10.1145/191246.191322`. [cited on p. 15]

Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald Mckay, James McGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck. Draft specification of the kqml agent-communication language. *Elsevier*, 1993. [cited on p. 15]

FIPA. FIPA ACL. [Online] Available: `http://www.fipa.org/specs/fipa00061/SC00061G.html#_Toc26669700`, 2002c. [cited on p. 15]

JADE. JAVA Agent DEvelopment Framework is an open source platform for peer-to-peer agent based applications. [Online] Available: `https://jade.tilab.com/`, 2020. [cited on p. 15]

Giovanni Caire and David Cabanillas. Jade tutorial. *Application-Defined Content Languages and Ontologies*, 2002. [cited on p. 15, 16]

Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995. [cited on p. 16]

Michael N. Huhns and Munindar P. Singh. Ontologies for agents. *IEEE Internet computing*, 1(6):81–83, 1997. [cited on p. 16]

P. Faratin, C. Sierra, and N.R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142(2): 205 – 237, 2002. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(02)00290-4. URL `http://www.sciencedirect.com/science/article/pii/S0004370202002904`. International Conference on MultiAgent Systems 2000. [cited on p. 16]

G. Kecskemeti. *Developing Interoperable and Federated Cloud Architecture*. Advances in Systems Analysis, Software Engineering, and High Performance Computing (2327-3453). IGI Global, 2016. ISBN 9781522501541. URL `https://books.google.pt/books?id=dhL4CwAAQBAJ`. [cited on p. 16]

Nicholas R Jennings, Peyman Faratin, Alessio R Lomuscio, Simon Parsons, Carles Sierra, and Michael Wooldridge. Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10 (2):199–215, 2001. [cited on p. 16]

Marco Alberti, Davide Daolio, Paolo Torroni, Marco Gavanelli, Evelina Lamma, and Paola Mello. Specification and verification of agent interaction protocols in a logic-based system. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 72–78, 2004. [cited on p. 17]

David J Farber. A distributed computer system. *ICS Technical Reports*, 1970. [cited on p. 17]

Reid G Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, 1(12):1104–1113, 1980. [cited on p. 17]

Muhamad Hariz Muhamad Adnan, Mohd Fadzil Hassan, Izzatdin Aziz, and Irving V Paputungan. Protocols for agent-based autonomous negotiations: a review. In *2016 3rd International Conference on Computer and information sciences (ICCOINS)*, pages 622–626. IEEE, 2016. [cited on p. 17]

Zafeer Alibhai. What is contract net interaction protocol. *IRMS Lab, SFU*, 2003. [cited on p. 18]

Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *AAAI*, volume 93, pages 256–262, 1993. [cited on p. 18]

Egon M Verharen, Frank Dignum, and Sander Bos. Implementation of a cooperative agent architecture based on the language-action perspective. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 31–44. Springer, 1997. [cited on p. 18]

Lai Xu and Hans Weigand. The evolution of the contract net protocol. In *International Conference on Web-Age Information Management*, pages 257–264. Springer, 2001. [cited on p. 18]

FIPA. Fipa iterated contract net interaction protocol specification. *Foundation for Intelligent Physical Agents*, 2000. [cited on p. 19]

S. Omatu, J. Neves, J.M.C. Rodriguez, J.F.P. Santana, and S.R. Gonzalez. *Distributed Computing and Artificial Intelligence: 10th International Conference*. Advances in Intelligent Systems and Computing. Springer International Publishing, 2013. ISBN 9783319005515. URL `https://books.google.pt/books?id=tu5KdI4XGJMC`. [cited on p. 20]

Benjamin Gâteau. A multi-agent system to monitor sla for cloud computing. In *ICEIS (2)*, pages 647–652, 2014. [cited on p. 20, 22]

Patrick CK Hung, Haifei Li, and Jun-Jang Jeng. Ws-negotiation: an overview of research issues. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10–pp. IEEE, 2004. [cited on p. 21]

Heiko Ludwig, Alexander Keller, Asit Dan, Richard P King, and Richard Franck. Web service level agreement (wsla) language specification. *Ibm corporation*, pages 815–824, 2003. [cited on p. 21]

Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad Van Moorsel, and Fabio Casati. Automated sla monitoring for web services. In *International Workshop on Distributed Systems: Operations and Management*, pages 28–41. Springer, 2002. [cited on p. 21]

Frank S. de Boer, Elena Giachino, Stijn de Gouw, Reiner Hähnle, Einar Broch Johnsen, Cosimo Laneve, Ka I Pun, and Gianluigi Zavattaro. Analysis of sla compliance in the cloud - an automated, model-based approach. *ArXiv*, abs/1908.10040:1–15, 2018. [cited on p. 22]

Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. *Kno.e.sis Publications*, 2009. [cited on p. 22]

Mark Gates. *Blockchain: Ultimate Guide to Understanding Blockchain, Bitcoin, Cryptocurrencies, Smart Contracts and the Future of Money*. CreateSpace Independent Publishing Platform, 2017. ISBN 9781547090686. URL `https://books.google.pt/books?id=KmPltAEACAAJ`. [cited on p. 22]

Maher Alharby and Aad Van Moorsel. Blockchain-based smart contracts: A systematic mapping study. *arXiv preprint arXiv:1710.06372*, 2017. [cited on p. 22, 31]

Saifedean Ammous. Blockchain technology: What is it good for? *Available at SSRN 2832751*, 2016. [cited on p. 22]

Lou Carlozo. What is blockchain? *Journal of Accountancy*, 224(1):29, 2017. [cited on p. 23]

BitOrb. Dlt technology: Blockchains vs dags vs tempo. [Online] Available: `https://www.bitorb.com/campus/dlt-technology-blockchains-vs-dags-vs-tempo/`, 2019. [cited on p. 23, 25]

Hemang Subramanian. Decentralized blockchain-based electronic marketplaces. *Communications of the ACM*, 61(1):78–84, 2017. [cited on p. 23]

Binance Academy. Blockchain advantages and disadvantages. [Online] Available: `https://www.binance.vision/blockchain/positives-and-negatives-of-blockchain`, 2017. [cited on p. 24]

Solidity. Solidity. [Online] Available: `https://solidity.readthedocs.io/en/v0.6.2/`, 2014. [cited on p. 26, 43]

Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017. [cited on p. 27]

Gareth W Peters and Efstathios Panayi. Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. In *Banking beyond banks and money*, pages 239–278. Springer, 2016. [cited on p. 27]

Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018. [cited on p. 27]

Blesson Varghese, Massimo Villari, Omer Rana, Philip James, Tejal Shah, Maria Fazio, and Rajiv Ranjan. Realizing edge marketplaces: challenges and opportunities. *IEEE Cloud Computing*, 5(6):9–20, 2018. [cited on p. 27]

Anastasiia Lastovetska. Blockchain architecture basics: Components, structure, benefits & creation. [Online] Available: `https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture`, 2019. [cited on p. 27]

Arati Baliga. Understanding blockchain consensus models. *Persistent*, 2017(4): 1–14, 2017. [cited on p. 29]

Andrew Poelstra. On stake and consensus. *WP Software*, 22, 2015. [cited on p. 29]

Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*, pages 557–564. IEEE, 2017. [cited on p. 29, 30]

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982. URL `https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/`. [cited on p. 29]

Michael Miller. *The ultimate guide to Bitcoin.* Pearson Education, 2014. [cited on p. 29]

Fahad Saleh. Blockchain without waste: Proof-of-stake. *Available at SSRN 3183935*, 2020. [cited on p. 30]

Fabian Schuh and Daniel Larimer. Bitshares 2.0: general overview. *accessed June-2017.[Online]. Available: http://docs. bitshares. org/downloads/bitshares-general. pdf*, 2017. [cited on p. 30]

Brent Xu, Dhruv Luthra, Zak Cole, and Nate Blakely. Eos: An architectural, performance, and economic analysis, 2018. [cited on p. 30, 45]

Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. *arXiv preprint arXiv:1902.10244*, 2019. [cited on p. 30]

Khahulin Pavel, Igor Barinov, and Viktor Baranov. Poa network whitepaper. [Online] Available: `https://github.com/charlespwd/project-title`, 2018. [cited on p. 30]

Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain. *Research Center of Cyber Intelligence and Information Security*, 2018. [cited on p. 30]

Metadium. Similarities and differences - dpos and poa. [Online] Available: `https://medium.com/metadium/similarities-and-differences-dpos-and-poa-eb03bc23dde8`, Jul 2018. [cited on p. 30]

Nick Szabo. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials*, 6, 1997. [cited on p. 31]

Yining Hu, Madhusanka Liyanage, Ahsan Mansoor, Kanchana Thilakarathna, Guillaume Jourjon, and Aruna Seneviratne. Blockchain-based smart contracts-applications and challenges. *arXiv preprint arXiv:1810.04699*, 2018. [cited on p. 31, 32]

Hyperledger. Implementation of service level agreement management feature using hyperledger fabric. [Online] Available: `https://wiki.hyperledger.org/display/INTERN/Implementation+of+Service+Level+Agreement+Management+feature+using+Hyperledger+Fabric`, 2019a. [cited on p. 32]

Huan Zhou, Cees de Laat, and Zhiming Zhao. Trustworthy cloud service level agreement enforcement with blockchain based smart contract. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 255–260. IEEE, 2018. [cited on p. 32]

Eder J Scheid, Bruno B Rodrigues, Lisandro Z Granville, and Burkhard Stiller. Enabling dynamic sla compensation using blockchain-based smart contracts. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 53–61. IEEE, 2019. [cited on p. 32]

David Johnston, Sam Onat Yilmaz, Jeremy Kandah, Nikos Bentenitis, Farzad Hashemi, Ron Gross, Shawn Wilkinson, and Steven Mason. The general theory of decentralized applications, dapps. *BitAngels*, 2014. [cited on p. 32]

Dapp. What are dapps? [Online] Available: `https://www.leewayhertz.com/what-are-dapps/`, Jun 2019. [cited on p. 32]

David Lucking-Reiley and Daniel F Spulber. Business-to-business electronic commerce. *Journal of Economic Perspectives*, 15(1):55–68, 2001. [cited on p. 35]

Sam Mire. Blockchain for e-commerce: 12 possible use cases. [Online] Available: `https://www.disruptordaily.com/blockchain-use-cases-ecommerce/`, 2018. [cited on p. 35]

Rúben Cunha, Bruno Veloso, and Benedita Malheiro. Renegotiation of Electronic Brokerage Contracts. In Álvaro Rocha, Ana Maria Correia, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *Recent Advances in Information Systems and Technologies*, pages 41–50, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56538-5. [cited on p. 36, 38]

Electra Tamani and Paraskevas Evripidou. Applying trust mechanisms in an agent-based p2p network of service providers and requestors. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 2, pages 13–13. IEEE, 2006. [cited on p. 36, 42]

Sergei Chichin, Mohan Baruwal Chhetri, Quoc Bao Vo, Ryszard Kowalczyk, and Marcin Stepniak. Smart cloud marketplace-agent-based platform for trading cloud services. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 388–395. IEEE, 2014. [cited on p. 37, 42]

Adina-Georgeta Cretan. Intelligent multi-agent platform within collaborative networked environment. *Challenges of the Knowledge Society*, page 975, 2016. [cited on p. 37, 42]

Diego Alejandro Cocconi, Jorge Roa, and Pablo David Villarreal. Collaborative business process management through a platform based on cloud computing. *CLEI Electronic Journal*, 2018. [cited on p. 37, 42]

Diego Cocconi, Jorge Roa, and Pablo Villarreal. Cloud-based platform for collaborative business process management. In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–10. IEEE, 2017. [cited on p. 37]

Kei-Leo Brousmichc, Andra Anoaica, Omar Dib, Tesnim Abdellatif, and Gilles Deleuze. Blockchain energy market place evaluation: an agent-based approach. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 321–327. IEEE, 2018. [cited on p. 37, 42]

Luís Ventura de Sousa. *Plataforma de negociação de componentes multimédia*. PhD thesis, Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto, 2012. [cited on p. 38]

Bruno Veloso, Benedita Malheiro, and Juan Carlos Burguillo. A multi-agent brokerage platform for media content recommendation. *International Journal of Applied Mathematics and Computer Science*, 25(3):513 – 527, 2015a. doi: https://doi.org/10.1515/amcs-2015-0038. URL `https://content.sciendo.com/view/journals/amcs/25/3/article-p513.xml`. [cited on p. 38]

Bruno Veloso, Benedita Malheiro, and Juan C. Burguillo. Media Brokerage: Agent-Based SLA Negotiation. In Alvaro Rocha, Ana Maria Correia, Sandra Costanzo, and Luis Paulo Reis, editors, *New Contributions in Information Systems and Technologies*, pages 575–584, Cham, 2015b. Springer International Publishing. ISBN 978-3-319-16486-1. [cited on p. 38]

Bruno Veloso, Fernando Meireles, Benedita Malheiro, and Juan Carlos Burguillo. Federated IaaS Resource Brokerage. In Gabor Kecskemeti, Attila Kertesz, and Zsolt Nemeth, editors, *Developing Interoperable and Federated Cloud Architecture*, chapter 9, pages 252–280. IGI Global, Hershey, PA, USA, 2016. doi: 10.4018/978-1-5225-0153-4.ch009. [cited on p. 38]

Fernando Meireles and Benedita Malheiro. Integrated Management of IaaS Resources. In Luís Lopes, Julius Žilinskas, Alexandru Costan, Roberto G. Cascella, Gabor Kecskemeti, Emmanuel Jeannot, Mario Cannataro, Laura Ricci, Siegfried Benkner, Salvador Petit, Vittorio Scarano, José Gracia, Sascha Hunold, Stephen L. Scott, Stefan Lankes, Christian Lengauer, Jesús Carretero, Jens Breitbart, and Michael Alexander, editors, *Euro-Par 2014: Parallel Processing Workshops*, pages 73–84, Cham, 2014. Springer International Publishing. ISBN 978-3-319-14313-2. [cited on p. 38]

Bruno Veloso, Benedita Malheiro, Juan Carlos Burguillo, and João Gama. Impact of trust and reputation based brokerage on the cloudanchor platform. In Yves Demazeau, Tom Holvoet, Juan M. Corchado, and Stefania Costantini, editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness. The PAAMS Collection*, pages 303–314, Cham, 2020. Springer International Publishing. ISBN 978-3-030-49778-1. [cited on p. 39, 75]

Apache Software Foundation and RedHat. Apache deltacloud. [Online] Available: `https://github.com/deltacloud`, 2009. [cited on p. 39]

Ethereum. Ethereum. [Online] Available: `https://www.ethereum.org/`, 2020. [cited on p. 43]

Hyperledger. Hyperledger – open source blockchain technologies. [Online] Available: https://www.hyperledger.org/, 2019b. [cited on p. 43]

R3 Corda. Corda - open source blockchain platform for business. [Online] Available: https://www.corda.net/, 2019. [cited on p. 43]

EOS. Eosio - blockchain software architecture. [Online] Available: https://eos.io/, May 2020. [cited on p. 43]

Hyperledger Besu. Hyperledger besu. [Online] Available: https://www.hyperledger.org/use/besu, Apr 2020. [cited on p. 44, 46]

Hyperledger Burrow. Hyperledger burrow. [Online] Available: https://www.hyperledger.org/use/burrow, Apr 2020. [cited on p. 44]

Hyperledger Fabric. Hyperledger fabric. [Online] Available: https://www.hyperledger.org/use/fabric, Apr 2020a. [cited on p. 44]

Hyperledger Indy. Hyperledger indy. [Online] Available: https://www.hyperledger.org/use/indy, Apr 2020. [cited on p. 44]

Hyperledger Iroha. Hyperledger iroha. [Online] Available: https://www.hyperledger.org/use/iroha, Apr 2020. [cited on p. 44]

Hyperledger Sawtooth. Hyperledger sawtooth. [Online] Available: https://www.hyperledger.org/use/sawtooth, Apr 2020. [cited on p. 44]

Hyperledger. *Hyperledger Architecture, Volume 1.* Hyperledger Org., Aug 2017. [cited on p. 44]

Arogyalokesh. Ethereum vs hyperledger. [Online] Available: https://mindmajix.com/ethereum-vs-hyperledger, Mar 2018. [cited on p. 44]

Corda R3 Consortium. Enterprise blockchain platform. [Online] Available: https://www.r3.com/corda-platform/, 2013. [cited on p. 45]

Docs Corda CorDapp. What is a cordapp? [Online] Available: https://docs.corda.net/docs/corda-os/4.4/cordapp-overview.html, Jan 2020. [cited on p. 45]

Docs Corda. Notaries. [Online] Available: https://docs.corda.net/docs/corda-os/4.4/key-concepts-notaries.html, Jan 2020. [cited on p. 45]

Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: an introduction. *R3 CEV, August*, 1:15, 2016. [cited on p. 45]

EOS Block One. Block.one - high performance blockchain solutions. [Online] Available: https://block.one/, 2019. [cited on p. 45]

Medium. The jungle community demonstrated a new max tps for eosio — 9656. [Online] Available: `https://medium.com/@cryptolions/the-jungle-community-demonstrated-a-new-max-tps-for-eosio-9656-ea8330c75516`, Feb 2020. [cited on p. 46]

Ethereum Geth. Go ethereum. [Online] Available: `https://geth.ethereum.org/`, 2013. [cited on p. 46, 47]

Nethermind. Nethermind .net client. [Online] Available: `https://nethermind.io/`, 2020. [cited on p. 46]

OpenEthereum. openethereum/openethereum. [Online] Available: `https://github.com/openethereum/openethereum`, Jun 2020. [cited on p. 46]

Hyperledger. Frameworks and tools - learning materials development working group. [Online] Available: `https://wiki.hyperledger.org/display/LMDWG/Frameworks+and+Tools`, 2020b. [cited on p. 46]

Cleos. Cleos | eosio developer docs. [Online] Available: `https://developers.eos.io/manuals/eos/latest/cleos/index`, 2020. [cited on p. 46]

Eoslime.js. Limechain/eoslime. [Online] Available: `https://github.com/LimeChain/eoslime`, May 2020. [cited on p. 46]

Jim Zhang. Consensus algorithms: Poa, ibft or raft. [Online] Available: `https://www.kaleido.io/blockchain-blog/consensus-algorithms-poa-ibft-or-raft`, 2018. [cited on p. 47, 48]

EthSigner. Ethsigner transaction signer. [Online] Available: `https://docs.ethsigner.pegasys.tech/en/latest/`, 2020. [cited on p. 48]

Fabric. *Open, Proven, Enterprise-grade DLT*. Hyperledger, 2020b. [cited on p. 48]

Hyperledger. Peers. [Online] Available: `https://hyperledger-fabric.readthedocs.io/en/release-2.0/peers/peers.html`, 2020c. [cited on p. 49]

Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Annual Technical Conference*, pages 305–319, 2014. [cited on p. 50]

Hyperledger. The ordering service. [Online] Available: `https://hyperledger-fabric.readthedocs.io/en/release-2.0/orderer/ordering_service.html#phase-two-ordering-and-packaging-transactions-into-blocks`, 2020d. [cited on p. 50]

Christopher Ferris. Hyperledger fabric performance and scale - blockchain pulse. [Online] Available: `https://www.ibm.com/blogs/blockchain/2019/01/answering-your-questions-on-hyperledger-fabric-performance-and-scale/`, Jan 2019. [cited on p. 50, 52]

IBM. Ibm/blockchain-application-using-fabric-java-sdk. [Online] Available: `https://github.com/IBM/blockchain-application-using-fabric-java-sdk`, Aug 2019. [cited on p. 52]

Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276. IEEE, 2018. [cited on p. 58, 59]

Hyperledger. Hyperledger fabric sampels. [Online] Available: `https://github.com/hyperledger/fabric-samples`, 2021. [cited on p. 65]