Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Distributed Ledger Technology: analysis of the technology and design of prototypical solution

Relatore: Prof. Claudio Zandron

Co-relatore: Dott. Riccardo Mazzei

Relazione della prova finale di:
Nassim Habbash
Matricola 808292

Anno Accademico 2017-2018

To my family, that has always supported me.
To my friends, that (almost) always endure my stupid jokes.

Abstract

Blockchains have been a disrupting force in the financial market. Popularized by Bitcoin, blockchains, and its underlying technology, Distributed Ledger Technology (DLT), has shaped up to be much more than the foundation for cryptocurrencies.

DLT can have applications in cross-border payments, or financial market infrastructures but its potential is not only limited to the financial sector, as it can enable digital identity solutions, or tamper-proof decentralized records for the flow of any kind of good, service or transaction. More generally, this technology can enable more secure, resilient and efficient systems, making it possible for further decentralization, deintermediation, greater transparency and cost reductions.

However, the technology is still in its infancy and it's rapidly evolving. This coupled with the cost of the migration for existiting longstanding IT infrastructures to a DLT infrastructure poses many new risks and challenges.

The aim of this thesis is to provide an analysis of Distributed Ledger Technologies, with the objective of identifying their risks, opportunities and implementation viability on different scales.

The research work starts from a brief introduction to the technology and how the industry has been adapting to it, with statistical and analytical evidence presented. The thesis then develops on the comparative analysis between the main competing technologies.

In accordance to the research findings, the project also aimed to provide a proof-of-concept design of a solution for a prototypical system answering to at least one identified use-case using the most suited DLT implementation between the analyzed ones.

Objective

The aim of this project was to provide an analysis to Distributed Ledger Technologies, with the objective of identifying their risks, opportunities and implementation viability on different scales. The research work starts from a brief introduction to the technology and how the industry has been adapting to it, with statistical and analytical evidence. The project presents then a brief overview of three main DLT platforms, with considerations on which technology to adopt, and why.

In accordance to the research findings, the project also aimed to provide a proof-of-concept architectural design of a solution for a credits interchange system using the most suited DLT implementation between the analyzed ones.

Structure of the thesis

- In the Introduction chapter a general introduction to the Distributed Ledger Technology is given, expanding on its technical design elements.
- In the Business research chapter, data on the the actual industry growth of the technology is presented, with statistical data and analysis from different sources.
- In the Comparative Analysis chapter, an overview and general comparison of the features of the three DLT technologies is given.
- In the R3 Corda Primer chapter, a primer on how Corda is structured is given, to better understand the choices of the prototypical implementation.
- In the Prototype design and implementation chapter, a prototype design is presented, showcasing how a DLT solution can automatize and implement a seamless distributed voucher payment system.
- In the Conclusions chapter, the author draws the conclusions on the results of the project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Emergence of the Distributed Ledger model

Ledgers have values as *archives*, or, in other words, their value is their capability of being consulted to check, verify and manage records.

Ledgers have been a central element of commerce since ancient times, and are used to record a variety of informations, ranging from financial assets to real estate properties, but most importantly how these change hands, that is, transactions.

The medium on which transactions have been stored may have changed from clay tablets to hardware storage, but in all this time there haven't been notable innovations to the underlying architecture of the system. Each financial institution (i.e. banks, governments, investment funds) manages its own ledgers, each designed differently based on necessities, goals and customers (the would-be *counterparts* in the transaction), and in turn, the counterparts keep recorded their own views of the transactions.

This duplication of information amongst all parties partecipating in the transaction drives a need for costly matching between each copy of the information, reconciliation and error fixing. The plurality of technology platforms upon which financial entities rely adds to that, creating more complexity and operational risks, some of which potentially systematic.

For example, let's consider the need for a party to transfer an asset, be it cash or a stock, to another party. The transaction itself can be executed in microseconds, but the settlement - the ownership transfer of the asset - usually takes more time, from days to weeks. This length is due to different reasons: the parties don't have access to each other's ledgers, and can't automatically verify that the assets about to be transferred are in fact owned and not counterfeit. So a number of intermediaries are needed as guarantors of the assets and to verify the transaction. A number of steps have to be added just for this trusting mechanism, and in addition, the differences between infrastructures and technologies of each party acting in the transaction can be such that there's always a need for reconciliation process between parties (ie adjusting each ledger to the transaction), increasing costs and length of the operations.

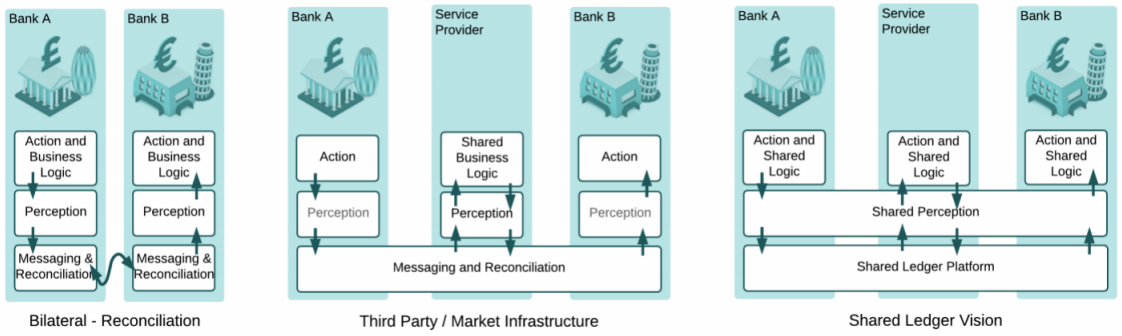

Figure 1.1: Comparison between architectures, **cordawhitepaper**.

Centralized infrastructures were until recently an unavoidable model, as there were few ways to consolidate technologies without effectively consolidating the financial entities themselves. The industry has been moving toward the standardization and sharing of data and some of the business logic behind the architectures through the delegation of some part of the process to third-parties, but these steps are still lagging behind the evolution of the technology.

The term Distributed Ledger Technology refers to the processes, protocols and technologies that enable nodes in a distributed network to share data (propose, validate and record) between multiple synchronized data stores, collectively maintaned. The emergence of this technologies has had a stimulating effect in the FinTech industry, prompting the reconsideration of the entities needed in a financial transaction, how should trust be established, the representation of the transaction, the securing of data, and many more. Even if DLT is not the answer in every case, asking these questions alone can be a force to drive progress forward.

1.2 Brief history

In 2008, a white paper (**bitcoinpaper**) written by an as yet unidentified person using the pseudonym of Satoshi Nakamoto, outlined a novel approach of transferring cash from one party to another without the need for a known and trusted third-party in a P2P manner, claiming, amongst other things, to have solved the issue of double-spend for digitalized currencies.

The technology outlined in the paper was named Blockchain, referring to the way of organizing data and transactions. Bitcoin has soared in terms of popularity and value through its cryptocurrency market, but is just one element in its whole architecture.

The effort of the industry since the introduction of blockchains has been directed to exploring different ways of leveraging this technology beyond Bitcoin, focusing on the core architecture of distributed record management. This use has gathered significant attention, reflecting the financial industry traditional reliance on multiple ledgers to maintain transactions. The use of DLT would be particularly effective for payment, clearing and settling activities because of the potential for simplification of the settling and reconcili-

ation process between the parties involved.

Some of the resulting implementations of DLTs have been on a steady rise, such as Ethereum, that similarly to Bitcoin has seen a steep rise to the value of its cryptocurrency, Ether, and unlike its predecessor, offers a more malleable enviroment (which is the main reason Vitalik Buterin created it), allowing for the transfer and recording of other assets like loans or contracts. Other rising implementations include R3 Corda, which showcases architecture heavily based on financial use-cases, IBM Hyperledger Fabric and Digital Asset Platform.

As the research and development of the technology progresses, real-world applications have highlighted some of the challenges associated with these use-cases, including the need for safe, secure and scalable systems.

As of 2018, the impact of DLTs in the financial sector seems still circumvented. Despite strong progress in the research, it would seem that in the near-to-medium term many of the benefits and efficiency gains of DLT are likely to be reaped by start-ups and financial institutions in the developing countries, such as ABRA, a company that offers instant P2P money transfers with no transaction fees through the Abra Network, combining cryptocurrencies with physical bank tellers; Ripple, that similarly deals in commercial cross-border and inter-bank payments with a peculiar dynamic approach towards transactions, where the flow of funds between a sender and receiver can go through a series of partecipating institutions that offer services (making customers, for example, find better foreign exchange transactions); ShoCard, a digital identity card that stores ID information on the Bitcoin blockchain, with the company currently being in the process of developing solution for different use cases like identity verification, fiancial services credentialing or automated registrations for online purchases.

What is a blockchain?

The term blockchain refers to the most well-known configuration of DLTs, and refers to a distributed ledger architecture where the data is stored in entities called transaction blocks, linked with each other through chained encryption. The blockchain itself is the data structure formed by these linked blocks. Blockchains make use of algorithmic methods and encryption to ensure immutability, security and truthfulness of the information.
New additions are initiated by one of the nodes that creates a new block of data, containing the encrypted transactions. Information about the block is then shared on the network, and all participants collectively try to determine the block's validity according to a pre-defined alghoritmic validation method (consensus). After the validation, all the participant can add the block to their copy of the ledger. With this mechanism, every change to the ledger is replicated across the entire network, and each node has a full, identical copy of the entire ledger at any point in time. As the chain grows and new blocks are added, earlier blocks cannot be altered.

The cryptocurrency aspect is what has made Bitcoin garn the most fame. Bitcoin was designed specifically for creating a digital currency free of government control, while also anonymizing the identity of the participants. The consensus process involves the generation of a reward to the node that validated the last block of the blockchain, that being the currency in itself.

1.3 Distributed Ledger taxonomy

It is emphasized that DLT is not a single, well-defined technology, but as of today there is a plurality of blockchains and distributed ledgers in active development.

DLs can be designed in a number of ways pertaining to main idea behind them and the use-cases they're designed to respond to. Such arrangements usually involve several key technical design concepts that specify how the information has to be kept on the ledger and how the latter has to be updated. There usually are four core attributes of DLTs, these are:

1. The distributed nature of the ledger
2. The cryptographic mechanisms
3. The consensus mechanism
4. The network access permission level

These four elements play are fundamental in ensuring the distributed ledger ability to store and exchange data across different, self-interested parties, without the need for a central record-keeper, without the need for trust amongst the concerned parties, as it is guaranteed by the system itself, and while assuring that no double-spending takes place. Each DLT addresses these attributes in their own specific way, but their abstract taxonomic aspects remain the same.

Double-spending

Double-spending is an issue unique to digital currencies, and is the risk that a digital currency can be spent twice. Physical currencies do not have this issue, as they're not easily reproduced, but digital information, on other hand, is easily replicated.

With digital currency, there is a risk that its holder could make a copy of the digital token and send it to another party, while retaining the original.

1.3.1 Distributed nature of the ledger

In its simplest form, a distributed ledger is a data store held and updated by each participant (or node) in a network. The control over the ledger does not lie within any single entity, but within several, if not all the network participants. This sets the technology apart from cloud computing or data replication, which are commonly used as shared ledgers.

There are different configurations to be analyzed regarding how the data is maintained over the ledger. In blockchains, no single entity of the network can amend past data entries, and no single entity can approve new additions to the ledger, which have to go through a predefined consensus mechanism. At any point in time there exists only one version of the ledger, and each network participant owns a full and up-to-date copy of it. After validation the new transaction(s) are added to all the ledgers to ensure data consistency across the network. In configurations like Corda's, each node maintains a separate ledger. The entirety of the ledger is the union of these ledgers, but isn't public, each peer can only see a subset of the facts on the ledger, and no peer is aware of the ledger in its entirety. This is due to Corda's design, where data is shared only on a need-to-know basis and only to directly involved parties.

Generally, this distributed nature of DLs allows the removal of a trusted central party, increasing speed and potentially removing friction costs and inefficiencies associated with the matching and reconiculiation processes. It also improves security, removing the single point of attack and single point of failure that is represented by the central trusted entity. To potentially gain control over the network, a malicious third party would have to gain control over 50%+1 nodes in the network.

Security risks aren't completely solved: the software layer built over the distributed ledger can become an additional attack surface.

1.3.2 Cryptographic mechanisms

Cryptography is at the core of the DLT. Asymmetric cryptography plays an important role by identifying and authenticating participants, confirming data entries and facilitating ledger updates. Each data entry is hashed, producing the so-called digest. The data is in this way hidden to anyone that is not intended to look at it, as the digest, which looks random and unrelated to the original input, is in fact deterministic, meaning that

from one original input there's only one hash possible. Digital signatures, which are a common and robust method used in a wide array of application are used as a means of authentication. Each network participant has a private key, that is used for signing digital messages and only known to the key owner, and a public key, which is public knowledge and used for validating the identity of the sender of the original message. participants proposing changes will authenticate themselves using digital signatures, and the validators will use cryptographic tools to verify whether the participant has the proper credentials, and so on. The validators can be either a counterpart, a third party, or the whole network depending on the type of DL and operation the change refers to.

In the blockchain subset of DLs in particular, encryption plays a fundamental role, as they're essential in the chain encryption mechanism between the blocks that make up the blockchain itself.

1.3.3 Consensus mechanism

The purpose of the consensus mechanism is to verify that the information being added to the DL is legitimate. It is fundamental in handling conflicts between multiple simultaneous competing entries (ie double spending), or take-overs by bad actors in the network. It's a derivative property of the distributed nature of the ledger, and it requires participants in the network to reach a consensus over the information being added. There exist many different consensus algorithms and mechanisms, with different purposes, advantages and disadvantages.

Consensus usually involves two steps:

1. Validation, where each validator involved identifies that the state change is consistent according to the rules of the ledger. This operation may rely on records of previous states or a last agreed state.
2. Agreement, where each validator agrees to the state changes to the ledger. This step involves the mechanisms to resolve eventual conflicts and ensuring that valid changes are made only once, thus ensuring that the whole network is synchronized.

According to the DLT configuration, the mechanisms to avoid double-spendings fit in either of the two steps.

The Bitcoin blockchain uses Proof-of-Work (PoF) to establish consensus. To add a new block to the blockchain, a node has to provide a proof of work. This is a computationally taxing problem, but easy to verify, and is solved by brute-forcing cyptographic hashing algorithms until the correct string that satisfies certain conditions is generated. This process is called "mining". Each miner that produces a valid PoF is then rewarded Bitcoins, which serves as an economic incentive to maintain system operation and integrity.

The Ethereum blockchain uses Proof-of-Stake (PoF). Its process is quite different from the PoF of Bitcoin, as there's no mathematical problem to solve, but instead, the creator of the new block is chosen in a deterministic way based on their stake, that is, how many coins or tokens they possess. A key advantage to this approach is the energy efficiency.

The Bitcoin network, for example, requires an annual energy consumption comparable to that of Columbia (57.6 TWh annually). Thus PoS systems are well suited to platforms where there is a static coin supply, without inflation from block rewards. The rewards consist only in the transaction fees.

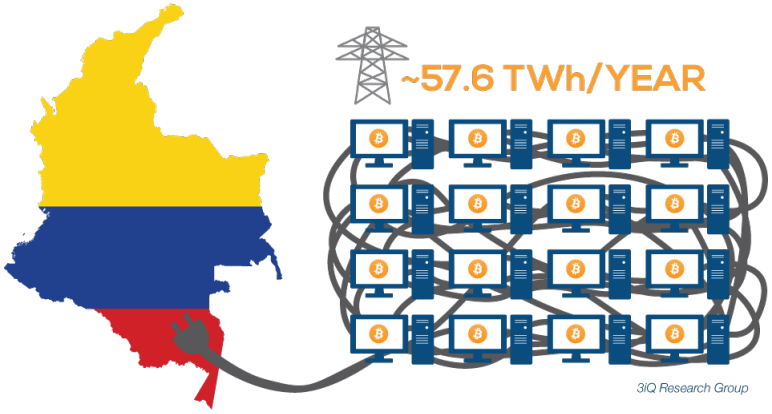

Figure 1.2: Data as at April 3rd, 2018. Retrieved from https://digiconomist.net/bitcoin-energy-consumption

The Corda distributed ledger, utilizes a unique "pluggable" consensus service, due to its peculiar distributed ledger architecture. It divides consensus in two types, validity consensus and uniqueness consensus. Given a proposed transaction, the first type of consensus checks whether the two parties satisfy a set of validity conditions, going back through all the parties' transaction histories, while the latter has the purpose to avoid double spend, and the consensus is provided by a Corda network service called "Notary", by attesting that, for a given transaction, there hasn't been proposed another competing one.

1.3.4 Network access permission level

The partecipation in the DL network can be open (permissionless) or permissioned. Bitcoin and Ethereum are the most prominent examples of completely permissionless blockchain, where participants can join or leave the network at will. This plays as one of their strengths, as a large, open permissionless system with a large number of nodes incentivized to validate new changes to the ledger and accurately and establishing a consensus is directly related to its network security (1.3.3).

In permissioned DLs its members are pre-selected by someone - an owner, or a network service - who controls network access and sets the rules of the ledger. The regulations of network access usually permit the use of a non-computationally expensive consensus mechanism, as there is no need for any trust between participants. This, however, means there's now a centralized trust entity playing a coordinating role and bearing the responsibility over the trusting mechanism. In permissioned DLs it's possible to have different degrees of transparency over the ledger, and faster transaction processing (thanks to the lighter consensus algorithm) allows for higher transaction volumes.

The identity verification needed for the access solves some problems with governments and regulators with concerns about the identity verification and legal ownerships clarifications.

Permissionless DLs have open access to the network, so anyone can join and leave as they wish. There's no central owner or administrator, the ledger is wholly transparent and the security is established by having a large scale network. It is required to have a complex consensus algorithm to guarantee the integrity of the information, and there are some legal concerns over the lack of ownership, as no legal entity owns or controls the ledger.

Some industry players make distinctions between public/private, in term of access, and permissioned/permissionless, in term of roles in the network. For example, Ripple has a permissioned ledger, but the data is validated by all participants, therefore being a public, permissioned ledger. Corda, on the other hand, has a permissioned ledger, but the data is validated only by a set of participants (those which the data concerns), hence being a private, permissioned ledger.

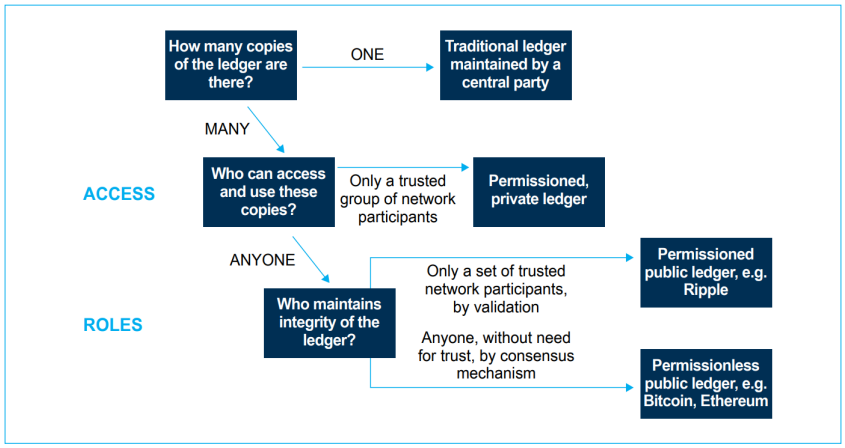

Figure 1.3: Network Access Ledger Taxonomy, **ukgovdltpaper**.

1.3.5 Roles

Nodes in the network may play a variety of roles, depending on the participants intentions or technical arrangement of the DL. The Committee on Payment and Market Infrastruc-

What are smart contracts?

Smart contracts are self-executing contracts with the terms of the agreement between two parties of a transactions, written as code. The code and the agreement exist distributed across a distributed ledger network. Smart contracts allow for trusted transactions and settlements to be carried out among different and possibly anonymous parties without the need for a central authority, legal system or enforcement mechanism. They render transaction transparent, irreversible and traceable.

tures of the Bank for Internation Settlements proposed a generalized framework, with the following different, non-exclusive roles for a node:

1. System Administrator: node controlling access to the system and provides dispute resolution and notary services. This role is not required in permissionless DLs.
2. Asset Issuer: node enabled to issue assets. In the Bitcoin blockchain, there's no entity playing this role as the system creates assets (Bitcoins) by itself, according to its rules.
3. Proposer: node enabled to propose ledger updates.
4. Auditor: node enabled to view the ledger, but not to make updates. Can be used by regulators or supervisors.
5. Validator: node enabled to validate requests for addition of transactions in the ledger. This role is performed by the consensus mechanism in permissionless DLs.

Chapter 2

Business research

2.1 Industry growth

2017 has been a year of exponential growth for the cryptographic asset market (term indicating the cryptocurrency market enabled by certain configurations of DLT). The outlook for 2018 seems to be set on the creation of solid regulatory frameworks required to provide consumer protection and guidelines.

Within less than a decade, the industry of cryptoassets has developed into a thriving ecosystem with a total market capitalisation of over 300 billion USD. The focus the cryptographic asset market is receiving through media and political coverage (Cryptoassets have topped the G20 agenda in March) has put increasing pressure on the actors playing in the market to match the needs and valuations from governance and financial institutions (the latter being players themselves).

It can be identified a drive from the investment banking industry (hereon sell-side) born out by the investment and participation in big consortia like R3 (which developed Corda) and DL platforms like Digital Asset. Institutional investors (hereon, buy-side) seem to be falling behind, and the risks this poses are of being presented with technology architectures with terms dictated by the sell-side self-interest.

The Bank of England identified this threat in a recent paper *"... if a DL (or other) technological solution for settlement ultimately succeeds in replacing existing settlement methods, it is likely to be characterised by network externalities and decreasing average costs. This suggests that the industry may well retain its high degree of concentration. It is possible and likely that a small number of Central Securities Depositories (CSDs) will be replaced by a small number of DL network providers and as a result future settlement services may be associated with some form of monopoly pricing"* **bankofenglandreport**. The buy-side failure to action may end up delegating to the sell-side the most of the definition and development of DLT infrastructure, the sell-side being a party with potentially diverging interests from them, while if it actually ended up participating in the development and research of DLT it could add another strong player in the industry with a sizeable budget to invest in the technology.

2.2 Governments stance

It is fair to say that DLT has been on a steady rise in the governments' agenda in light of its saving, automatization and security advantages. Central banks around the world are exploring DLT-based digital currencies - UK, Russia, Sweden, Canada and China's central banks are assessing risks and benefits of issuing fiat currencies backed by digital currencies, and investigating their potential effects on the economy and on financial stability.

The UK Government's Office of Science published a major report on DLT in January 2016, **ukgovdltpaper**, assessing the possibilities of DLT use in private and public areas. The Department for Work and Pensions in the UK has been testing from June 2016 the use of DLT for welfare benefit payments, that trough a phone application lets welfare claimants manage their benefit money, having transaction recorded on a DL with the aim to create a more solid and efficient welfare infrastructure preventing frauds.

The Estonian government has been experimenting with DLTs for a long time, as apparent from its **eresidency** platform, where it's possible to verify government records like birth or marriage certificates, which are only a couple of the services provided by, as the e-Residency project seem to be trying to encompass a whole variety of utilities, like opening a bank account or starting a company (in Estonia), but most importantly, providing a form of transnational digital identity, so far as to having NASDAQ partnering with the platform to enable secure e-voting in shareholders meetings.

By 2020 Dubai wants to become the first government in the world to conduct all of its transactions using blockchain. The emirate estimates that adding visa payments, license renewals and other documents to the blockchain could save 1.2 billion EUR annually in document processing alone, and also cut CO2 emissions and redistribute 25.1 million hours of economic productivity.

The European Union Intellectual Property Office (EUIPO) is investingating how blockchain could combati counterfeiting, which costs the EU about 60 billion EUR each year according to the agency. In June 2018 the EUIPO organized a Hackathon competition in Brussels to develop a series of anti-counterfeiting blockchain solutions, drawing support

Sell-side and Buy-side

Sell-side and buy-side are terms belonging to the financial investment world.
Sell-side refers primarily to the investment banking industry. It refers to the key function of the investment bank - namely helping companies to raise debt and equity capital and then sell those securities to investors. The investment bank role is that of a seller of corporate securities to insititutional investors.
Buy-side broadly refers to such insititutional investors, such as mutual funds, hedge funds, insurance companies, endowments and pension funds. They raise money from investors and invest that money across various asset classes using a variety of different trading strategy.
As of 2014 the estimated flow of money between the two sides is of approximately 227 trillion USD in global assets.

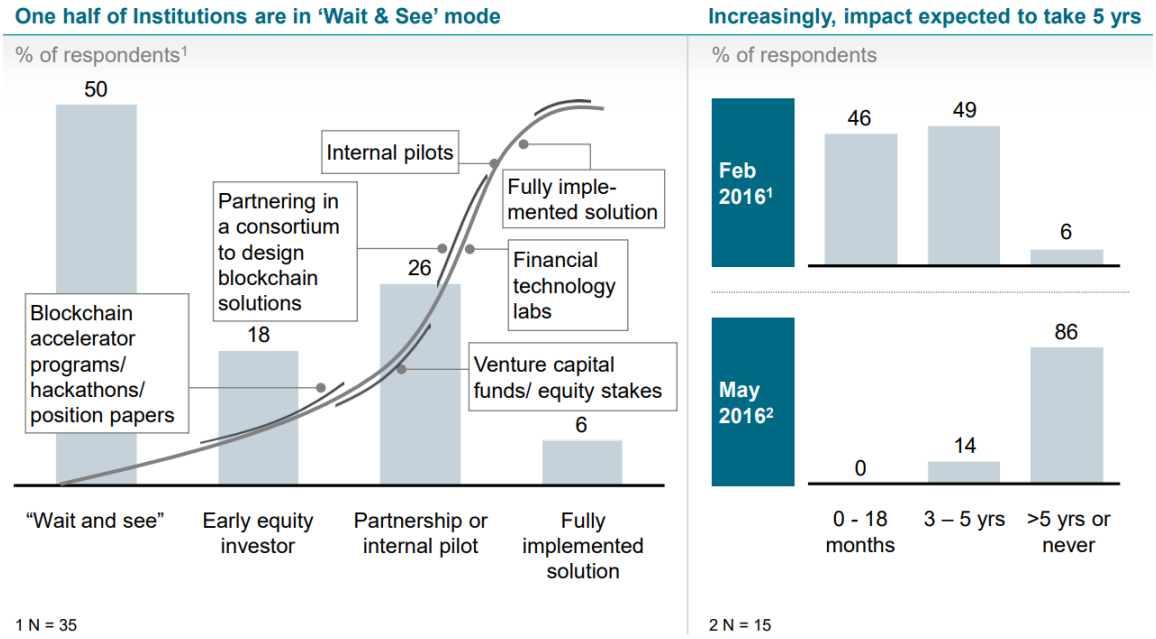

Figure 2.1: Infographic on the industry response to DLT, based on survey of senior executive leadership in financial institutions, Feb 2016 and May 2016, McKinsey & Company.

from specialists in law, IP and anti-counterfeiting.

Briefly, other relevant government-backed applications and researches include the USA's Food and Drug Administration's exploring blockchain solutions for patent sharing, Denmark's Liberal Alliance blockchain voting platform and Georgia's blockchain land registry project.

2.3 Financial stance

The two biggest trends in DLT development seem to be:

1. Commercial FinTech startups developing applications for different purposes utilizing public blockchain infrastructure, like BitCoin and Ethereum.
2. Industry consortiums researching and developing private, permissioned distributed ledgers to address a set of industry-specific enterprise use-cases.

A survey by Greenwitch Associates, **greenwitchdltreport**, gathering 400 market partecipants (as in financial companies, funds, etc) working on DLT has tried to assess opinions on the key trends and issues in the current state of DLT development. The report finds engagement across the sample at 63%, with Market Infrastructure providrs in the high end with 75% engagement, and Asset Managers in the bottom, with 32% engagement.

This presents a picture of lagging in the adoption of the technologies, due to the competition of other competing technologies. According to a recent survey from **multifundssurvey**, big data analytics, AI and robo-advice are currently higher up in the Asset Managers' agenda than DLT, accounting for more than half (55%) of responses, explaining that it is

"probably down to the fact that these new technologies are relatively easier to implement than more revolutionary DLT concepts".

But as **ibmdltreport** reports, recent evidence shows that the industry is notwithstanding operating in the research and development department. Examples (extrapolated from the report), include:

- Schroders announcement that they have joined the Hyperledger Project
- Northern Trust's implementation of a blockchain platform for Private Equity fund administration in partnership with Unigestion and IBM
- BlackRock's announcement of their intention to Blockchain-enable Provider Aladdin, a private platform / dashboard to streamline transactions with BlackRock's Custodians
- The launch of a Blockchain-based solution for syndicated loan servicing by Synaps (a joint venture of Ipreo and Symbiont), with involvement from Asset Managers including Eaton Vance and Alliance Bernstein
- Calastone's launch of Blockchain-based distributed market infrastructure
- FNZ 's development of FNZChain as a private blockchain for Asset Management registers
- SETL's launch of Iznes, in collaboration with various asset managers, as a Pan-European Distribution and Transfer Agent Platform for fund subscriptions, distributions and settlements
- The announcement from Natixis in July 2017 that they had successfully sold funds directly to clients through FundsDLT, a fund distribution platform developed by a partnership of the Luxembourg Stock Exchange / Fundsquare, InTech and KPMG;
- The news that SEB are working with NASDAQ on the development of a trading platform for Swedish mutual funds
- The rise of crypto fund launches in 2017, representing the fastest growth of any hedge fund sector in the industry's history, **hedgefundcryptoreport**

2.4 Adoption considerations

It is clear that many market players and public authorities have embarked on a learning process that has familiarised many of them with the foundations of DLT. Investment in the technology has started to gain momentum, and is expected to grow at a very high pace in the near future.

The technology has the potential to support attractive models for the tech-savvy generation of consumers who wants more control over their investments and finances, delivered via their favoured media, thus providing a strong and agreeable user base.

The effective use case execution of said solution will depend highly on the collaboration among players in an ecosystem. In the financial services case, a strong interest has been outlined by banks and financial institutions and fintech companies, with a 400 million USD estimated capital market spending by 2019. It is believed that right now the biggest challenge for DLT is going to be shaping a regulatory environment and the agreement on key standards and active collaboration across all required players.

It is hence believed that entering the market in this moment seems to be the prime time to ripe the benefits provided by the assessing ecosystem to solidify a strong and longstanding position among current and future competing solutions. Entering the market at this point in time - when the industry hasn't yet transformed itself - offers real potential that isn't ignorable, as most of the highest commercial gains expected for DLT will take place in the first transitional period after its stabilization in the market.

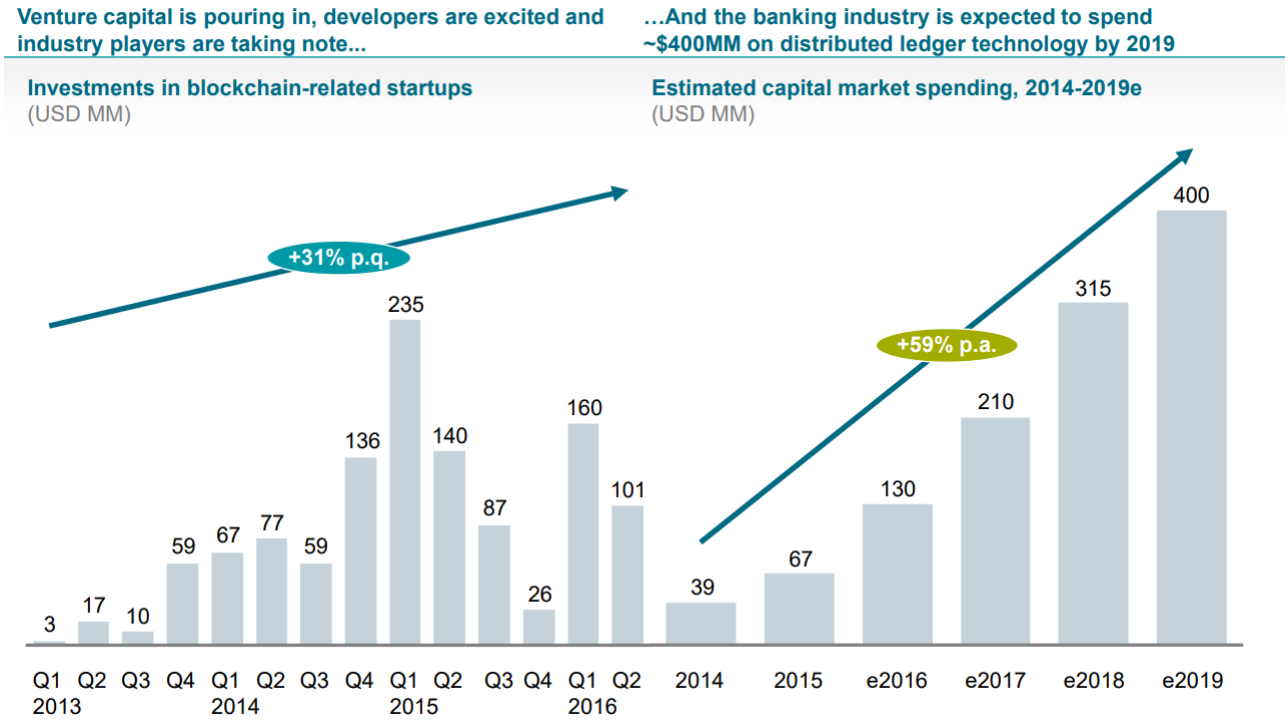

Figure 2.2: Infographic on the investments in DLT development, based on data from AITE Group, Tabb Group, CoinDesk

Chapter 3

Comparative Analysis

The adoption and popularization of DLT has created many new platforms that offer the possibility of building a distributed application. This creates a peculiar situation in which developers and companies need to decide which platform to choose to develop their applications, with or without any kind of experience in the field.

As these platform's architecture can wildly vary between one and another, it seems to be the case that the industry misses at the moment clear research and understanding of how to separate platforms use cases and practical examples on how such platforms have been employed and why. This is to be expected, as the industry is experiencing rapid growth and evolution, and no standardization and regulamentation has been put into place yet.

This chapter aims to analyze three major DLTs: Ethereum, Corda and Hyperledger. It will provide an overview of each DLT, and then dive into a comparison between the technical aspects of each, in order to showcase the major differences among them.

3.1 Ethereum

Ethereum has one of the biggest following among users and developers, with more than a thousand applications already built on top of the blockchain. It was developed as a permissionless public blockchain where anyone can build application or write smart contracts using its own programming language, Solidity. Ethereum's development team's goal was to provide a generic toolbox for providing support to a wide range of decentralized applications, with a particular emphasis on situations where rapid development time, security and continuous execution of concurring different applications are important. Ethereum boasts the second highest market cap in the cryptocurrency world, which explains how the blockchain project continues to elicit strong interest from investors.

3.1.1 Architecture overview

Ethereum is a decentralized blockchain platform for building "unstoppable applications", as the whitepaper puts it, while Ether is the cryptocurrency used in the platform.

Ethereum is an open, permissionless blockchain, hence any developer has free access to the platform to build distributed application, using the built-in programming language. These applications are called Dapps. End users interact with the blockchain through these Dapps, with miners creating blocks in which user's transactions are hashed and stored.

The structure of the Ethereum blockchain is very similar to Bitcoin's. Every node on the network stores a copy of the entire transaction history. With Ethereum, every node also stores the most recent state of each smart contract, in addition to all the transactions.

There's no centralized control in Ethereum: node operators decide which software their nodes run, miner operatores decide how to mine - solo or in a pool.

Consensus in the blockchain still uses Proof-of-Work, but the upcoming Casper implementation will introduce Proof-of-Stake, which works similarily to PoW, but as explained in 1, the partecipants in the consensus process are limited to parties which have a real stake in the blockchain, removing the more taxing computation of the hash function with just a digital signature that proves the ownership of the stake in the system.

Ethereum, being a public blockchain, can experience scalability issues. The change to a PoS algorithm might benefit the platform in this regard, but its real effects are still to be explored.

Smart contracts in Ethereum are akin as "autonomous agents", as **ethereumwhitepaper** puts it. They're programs stored and created in the blockchain that execute only when specific criteria are met, and are immutable, once deployed they can't be changed. The code consists in a seres of operations that runs until it reaches the end, an error, or one of the conditions is violated.

Ethereum has the concept of "gas". Gas is the execution fee for the execution of a contract, and usually maps to one step of its computation. Its price is expressed in ether, which represent Ethereum's internal currency. Gas is hence the amount of ether a sender is willing to spend on every unit of gas.

3.1.2 Use case

Ethereum itself is commonly used as a digital currency or digital asset to store and transfer ethers between accounts. Its most common type of application is custom token systems.

3.2 R3 Corda

Corda is a DLT backed by R3, a consortium made up of many big financial institution. It was developed as a global ledger, with its main goal is to provide an architecture to enable frictionless, well-regulated, reliable and private agreements between parties. Corda was developed specifically on financial use-cases, its main field of application being the financial services industry.

Corda's main goal is to create a global logic ledger, that is, evolving from authoritative systems of records centrally maintained within institutions to a global distribuited authoritative system of records shared between firms, in which all the economic actors will interact in a secure, consistent and private manner.

This has been implemented including the three major visions - as expressed in the **cordawhitepaper**: records managed by the system have to be accessible only on a need-to-know basis by only actors involved in the records; agreements managed by the system and described by their corrispettive contracts gain legitimacy from its over-arching legal prose, which is a mechanism to regulamentate transactions; and to gain adoption across the financial community, portions of the system must be open source and up to industry standards.

3.2.1 Architecture overview

Corda is a private, permissioned distributed ledger. To ensure consistency in a global system where not all data is visible to everyone, Corda relies heavily on secured cryptographic hashes to identify parties and data.

The core concept model of Corda is built by state objects, representing agreements between two or more parties governed by smart contracts, transactions and flows, which are automatized transaction sequences which enable parties to coordinate actions without a central controller.

A Corda distributed ledger is formed by the union of nodes participating in a Corda network. It differs greatly from a blockchain structure, as the data stored in the distributed ledger isn't accessible to anyone but the interested parties.

All nodes participating in the network require their identity to be verified. This passage is guaranteed by the "doorman", which is a node regulating entry in the network.

In Corda transactions involve the evolution of state objects between the distributed ledger of the parties involved in it.

In Corda there are two types of consensus: validity consensus and uniqueness consensus, as noted in 1.3.3. This type of consensus service means that the guarantor of the consistency of the data isn't the whole architecture itself, but the notary service. This means also that these services are not required to see the full contents of any transaction, significantly improving privacy and scalability of the system compared to other blockchain designs which employ the whole architecture to guarantee consistency.

In Corda there smart contracts are agreements between two parties under the form of programs. These smart contracts and their associated legal prose link business logic and business data, and ensure that financial agreements on the platform are rooted firmly in law and can be enforced in the event of ambiguity.

Distributed applications in Corda are called CorDapps.

3.2.2 Use case

Corda's use case is fundamentally rooted in finance. Use cases include automate auditing of records, payments applications and financial trading applications.

3.3 Hyperledger Fabric

Hyperledger Fabric is a blockchain framework developed by the Linux Foundation. It was intended for developing solutions with a modular architecture, as hyperledger allows components to be plug-and-play. As stated on Hyperledger's website: "Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing, and Technology.". Hyperledger's project tries to incubate different business blockchain technologies and frameworks under what it calls its "Umbrella strategy". Fabric is one of the platforms falling inside the Hyperledger project, and its intended use is that of a generic DLT framwork to enable many different kinds of solutions, not only limited to finance.

3.3.1 Architecture overview

Fabric's blockchain is a private permissioned distributed ledger consisting of many nodes running programs called chaincodes. Chaincodes are Fabric's version of smart contracts, and are the central element as transaction themselves are invoked by chaincodes. Participating in the network are also special validating and non-validating nodes, which are run by white listed organisations and transactors.

Fabric permissioned network entry is due to its need to guarantee privacy of the operation for participants in the network. This does not exclude the possibility of identification and point-to-point audit done by regulators. Within Hyperledger, content confidentiality is achieved by encrypting the transactions such that only the stakeholders can decrypt and execute them.

Transactions may be of two types, deploy transactions create new chaincode and take a program as parameter. When a deploy transaction executes successfully, the chaincode has been installed "on" the blockchain. Invoke transactions on other hand perform an operation in the context of previously deployed chaincode. An invoke transaction refers to a chaincode and to one of its provided functions. When successful, the chaincode executes the specified function.

Consensus in Hyperledger is pluggable, like Corda, allowing user to select which implementation or algorithm to use during deployment.

As Fabric is permissioned, it has higher scalability than permissionless distributed ledgers.

3.3.2 Use case

Fabric's use case is broad, as it provides a way to store confidential data that is required for any supply chain. Its use can range from supply chain logistics to trade and financial, or just secure data storing.

3.4 Comparison

From their respective white papers and documentations it becomes obvious that each of these platforms have very different visions and applications in mind. Corda is driven by concrete use cases from the financial services industry, in contrast, Fabric intends to provide an extendable architecture that can be employed in various different industries. Ethereum also present itself as independant from any specific field of application, but in contrast to Fabric, it's architecture provides a generic platform for transactions, rather than a modular platform for securing data.

It is possible to place these platform on a circular spectrum. On one end, there are Fabric and relatively close, Fabric, which provide a highly flexible and generic platform. However, Ethereum's permissionless nature makes the issues of scalability, cost of performance and privacy arise for use cases that actually demand it. Fabric addresses performance scalability and privacy by its permissioned mode of operation and encryption. On the other side of the spectrum there's Corda. It has been consciously designed as a DLT for the financial services industries, taking the highly regulated environment into account. This use case focus provides an architectural design much simpler than Fabric and Ethereum at the cost of flexibility.

The only DLT featuring a cryptocurrency between the three is Ethereum, which uses Ether. Ether is emplyed to pay rewards for nodes that reach consensus. As Fabric and Corda do not reach consensus via mining, they do not need to feature a cryptocurrency in their systems.

The architectures between the three platforms are quite different,hence the comparison will be driven along one shared axis that can highlight each platform's strength and weakness: how data is managed within the system.

3.4.1 Data comparison

Data coordination

Public blockchain do not have anywhere near the same throughput and data processing capabilities as private distributed ledgers. The same holds true for **Ethereum**, so a comparison on those factors would be certainly be heavily in favor of Corda's or Fabric's distributed ledgers. Still, Ethereum features a wide range of forks which maintain the same core architecture, but make key changes to increase throughput and processing capabilities, like Quorum. Of note is the change from Proof-of-Work to Proof-of-Stake that is going to be implemented in the Casper version of Ethereum, which will substatially

increase throughput via a lighter consensus mechanism.

Corda's architecture relies on a nodal structure and submodules called notaries to help maintain the validity of the network. Each node has a relational database, and their union makes up the distributed storage platform. The data is only accessible by individual parties directly concerned in the transaction, so the actual global distributed ledger isn't accessible by anyone. Coordination of data is carried out through flows, which are programmable and predefined functions that allow communication between nodes. Additionaly, Corda features a layer of identity control (the doorman) which allows for different degrees of access control to the network.

In **Fabric** the movement of chaincode between clients and the network of distributed peer nodes, along with the transacion mechanisms and transfer of receipts that satisfy endorsement policies is the backbone of the coordination of data in the closed system, while the gossip protocol that propagates transactions within private channels allows for the coordination of larger datasets. The architecture was specifically designed to allow multilateral coordination structures. In this setup, data is distributed across clusters to make up a distributed storage platform. Fabric makes use of a ledger-type structure that deviates from the blockchain structure. It is in fact an hash liked data structure, but these data blocks, while transitioning through what are called deliver events, do not necessarily transition data into a modification of the system's state, rather, the information is stored in a database-like structure with different instances of hashes. The configuration of the system allows for a transaction throughput that would be expected of the distributed database architecture, but it should be made present that challenges regarding coordination between data and code haven't been completely solved yet.

As a conclusion to this section regarding data coordination, it can be said that Fabric has the edge over the other two platforms thanks to its superior and extensible data coordination toolset. Corda is still defining its capabilities offering coordinating service which are of use specifically to financial institutions, while Ethereum, being designed as a public blockchain, does not have the same raw database processing capabilities as the former two. Forked architecture to Ethereum are designed to close the gap in terms of processing capabilities while allowing the features of a blockchain.

Trust and data immutability

Data immutability has been used as a synonymous concept of trust, that is, *since the system is immutable, it is trusted that tampering will not go unnoticed*. It is though important to asses how a trusted system is implemented, to ensure a safe business model.

For **Ethereum**, different layers of trust and immutability have been enstablished in the platform and protocols - in particular for data, within a subprotocol of public blockchain derived state roots from Patricia Merkle Trees. The immutability if the blockchain as a whole is guaranteed because of the number of miners keeping track of the blockchain's ledger. Altering data in a client's ledger will result into a mismatched version between

that node and every other node in the network, thus resulting in the tampered node to be ignored by the whole network. Ethereum and its enterprise forks are hence able to fully substantiate immutability.

For **Fabric**, in order to guarantee that no party in the channel has tampared data in its favor, a sophisticated endorsement policy has been put into place. It is necessary for the client which issues a new transaction to get endorsement from all interested parties, thus ensuring that everyone have a consistent state. It must be understood that the architecture itself, utilizing Apache Kafka for its channel communication protocol, has inherent read/write access to data, so the immutability aspects are somewhat limited to architectural choices.

For **Corda**, the aspect of immutability is preserved differently withing the confines of the system. The trust is in fact enstablished due to the architecture of the system through, the combined acts of the identity verification and regulatory nodes action. Data entries - called states - cannot be modified, but only consumed.

Each platform is able to guarantee immutability and trust to various degrees. The edge is given to Ethereum, for which the whole platform is able to guarantee different layers of trust and immutability to the data. Fabric and Corda are somewhat limited to their approach to guarantee immutability and trust due to their architectures, but nevertheless are able to secure data.

3.5 Project adoption considerations

In order to have a robust enough platform that can manage transaction in an easy and secure way, the system must be able to satisfy business requirements for efficient processing of data, trust and immutability of the information. It is apparante that each platforms can achieve the same goal, to various degrees.

Cost-resource considerations have been made in regards to which technology adopt for the project, and Corda has been chosen as the technology to use for the development of the prototype for this project work. This is due to its simpler architectural design, which would require a definitely shorter amount of resources - time, developers - to understand compared to its competitors, and to the focus on financial use cases, which the underlying project this project work takes place in is set up. The issues of scalability, privacy and efficiency have also been taken in consideration, thus ruling out a public, permissionless environment.

Chapter 4

R3 Corda Primer

This chapter is meant as a short primer on Corda to better understand its architecture and functionalities in preparation for the prototype implementation details.

4.1 Principal features

Corda provides a framework specialized for use with regulated financial institutions. It is a distributed ledger system inspired by blockchains, but its design set it apart from traditional blockchain architectures.

Its main features, extracted from the **cordawhitepaper** are:

1. Recording and managing the evolution of financial agreements and other shared data between two or more identifiable parties in a way that is grounded in existing legal constructs and compatible with existing and emerging regulation
2. Choreographing workflow between firms without a central controller.
3. Supporting consensus between firms at the level of individual deals, not a global system.
4. Supporting the inclusion of regulatory and supervisory observer nodes.
5. Validating transactions solely between parties to the transaction.
6. Supporting a variety of consensus mechanisms.
7. Recording explicit links between human-language legal prose documents and smart contract code.
8. Using industry-standard tools.
9. Restricting access to the data within an agreement to only those explicitly entitled or logically privileged to it.

4.2 Network topology

A Corda network is made up of authenticated nodes engaging in peer-to-peer communication, where each node is a Java Virtual Machine runtime hosting Corda services and executing CorDapps. All comunication is direct and TLS-encrypted: the data is shared only on a need-to-know basis between nodes, and there are no global broadcasts.

A Corda network has a figure known as the doorman. Such service enforces identification at network entry and based on the rules it was set up with, admits nodes to the network - this is what makes Corda a permissioned, private distributed ledger.

Identities in Corda are certificates (X.509) which represent the legal identity of an organisation or service identity of a network service.

4.3 Core mechanics

In Corda there's no single central store of data. Instead each node mantains a separate database of states. The entirety of the distributed ledger is made up of the union of such databases - which are called vaults. The union of states strictly regarding a transaction between two nodes makes up the ledger between two nodes.

States are what are recorded on ledgers, and represent facts such as funds possessed by the node's owner, loans, or any kind of financial agreement between parties. States are immutable and evolve. A state consumed by a transaction becomes historic, and can be replaced by a new state generated by the same transaction.

Transactions are regulated by contracts: a contract verifies if a transaction abides by its rules, and if it does, is verified and can be finalized in the ledgers of the parties engaging in it. Transactions take in a set of input states and generate a set of output states. Transaction validity is given by the digital signatures of the parties engaging in it and the smart contract itself.

Consensus is computed through a third node, the Notary, which attests that for a given transaction it has not already signed other transactions that consumes any of the proposed transaction's input states - thus preventing double-spends. As the consensus is pluggable in Corda, any of the given algorithms can be used for computing the consensus. Furthermore, every input state is walked through its history, to verify if every state that brought to the final and current one is valid - this is called validity consensus.

Flows are automatization of transactions. The flow framework allows developers to automate the process rather than specify the transaction steps manually. Nodes in Corda are made to communicate through flows.

4.4 CorDapp

CorDapps are distributed applications running on the Corda platform, allwoing nodes to communicate. This is achieved by defining flows that Corda node owners can invoke through RPC calls or HTTP APIs.

CorDapp in Corda can be written either in Java or Kotlin. And their main components are:

1. State classes, defining the facts over which agreement is reached
2. Contract classes, defining what constitutes a valid ledger update
3. Flow classes, defining flows for transactions

Between other components, we list Serialization whitelists classes, which restrict what types a node can receive off the wire, and Service subclasses that provide long-lived utilites within the node.

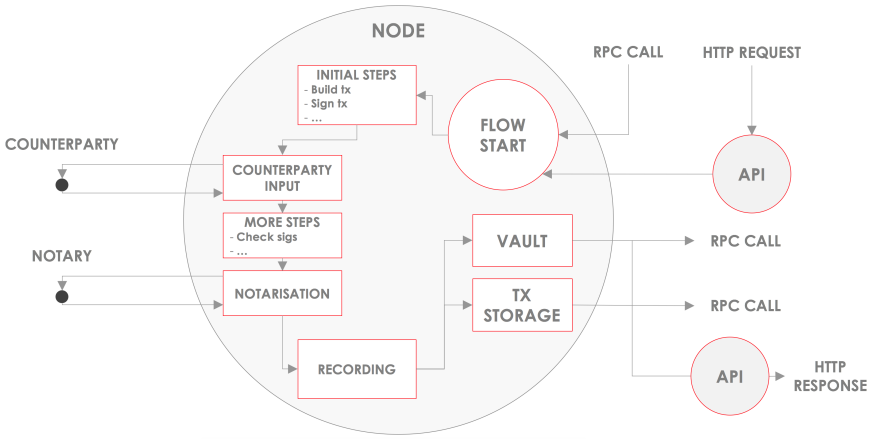

Figure 4.1: CorDapp structure diagram

Chapter 5

Prototype design and implementation

5.1 Background

5.1.1 Scope

This prototype has been designed as a viability test for a DLT solution utilizing R3 Corda. The planning of this project includes the choice of the methodologies used and the description of the development process. This project is part of a bigger project that involves the complete development of a platform that could integrate some of the functionalities offered by this design.

The pratical objective of this project is to estabilsh know-how and a foundation for the use of this technology, providing a template and roadmap for future developments. As agile methodologies are preferred over heavyweight and longer ones, the design here presented might not represent the last iteration of development, and not every functionality might be implemented.

Although all the documentation will be required by the end of the development process, it has been decided to exclude or not analyze further marginal functionalities not pertaining the DLT architecture, as drafting what would be a large documentation without direct influence on the software implementation has been deemed unnecessary.

The sections here presented describe the requirements of the solution and the proposed design in the form of simplified diagrams.

5.2 Description

The project aims to develop a platform enabling transactions, mainly commercial ones, through different kind of credits and currencies.

As the project main focus has been the integration of voucher usage in the system, the requirements and design will focus on such use case.

Meal vouchers are redeemable tickets given by an employer to its employee as a form of benefit. Vouchers have a fixed value and have to be spent in their entirety. Meal vouchers can be used only in a affiliated shops or premises.

When cost and voucher value do not add up the issue of unspent change arises, making payments problematic and more cumbersome than they need to be. The regulatory framework legislates that vouchers cannot be converted into money, but the decision to whether give change to the voucher user is left to the merchant.

The purpose of this platform (ExChange) is to provide a platform for payments in which vouchers are a form of credit that can be used to seamlessly execute transactions between parties. The platform addresses the issue of leftover change through two hypothetical solutions: it can either provide back real change or issue IOUs (discount states) to the merchants receiving the vouchers, granting the voucher user the leftover change for future purchases from that same merchant. The platform also supports pooling of fixed values credits from multiple buyers for batch purchases, minimizing leftover change.

Potental users of this project include end-users (or costumers) searching for a seamless payment platform that supports different kinds of vouchers or fixed value credits.

Potential users of this project also include companies utilizing vouchers as a form of empleyee benefit which seek to automatize voucher delivery, or take advantage of the system as a payment platform.

5.3 Requirements

The application should have the common features of an e-wallet. It should provide visibility to a user's funds, transaction history, and provide other features such as credit (currency) fund addition and fund transfer.

The application should permit a user A to initiate a payment transaction, and another user B to accept such transaction and proceed with its completion. For example, a seller might send a payment request to a buyer over a commercial transaction, and the buyer should be able to accept it and complete it.

Payments can be done through vouchers or other forms of credits. Vouchers will require to be specially handled.

The application should provide a way for a user to manage its own account, and should provide different user profiles according to one's intended use of the application: as a costumer, as an employer or as a seller.

A seller and company profile should require additional legal information and require (in this scenario) a voucher originator affiliation, to either receive vouchers or be able to redeem them.

Voucher originators are voucher issuing companies. They are part of the system as they issue vouchers directly on the accounts of voucher beneficiaries according to their contracts with the beneficiaries employers.

The application should be accesible via smartphone or browser.

5.4 System model

5.4.1 Actors

1. **Generic user**: A generic user is a user that has in his account a certain amount of credits. A generic user can either receive or transfer his credits as part of a transaction. contemplated transactions involve selling or buying goods - hence a generic user can either be a buyer or a seller.
2. **Company**: A company is a legal entity made up of an association of people. A company might represent a seller or a conglomeration of employees pooling their credits together for batch purchases. Only a registered company can redeem fixed value credits from its originator.
3. **Voucher originator**: A voucher originator is the entity that issues vouchers. Vouchers are directly issued in the system as a form of fixed value credit.
4. **ExChange application**: This actor represents the system and the actions it takes. The platform acts as a gateway between the actions the user takes and the underlying CorDapp. It is responsible for presentation of the main interfaces to the user.
5. **CorDapp**: This actor represents the CorDapp, that being the node of the distributed ledger. This actor is responsible for committing and securing every transaction, and for new users joining the system.

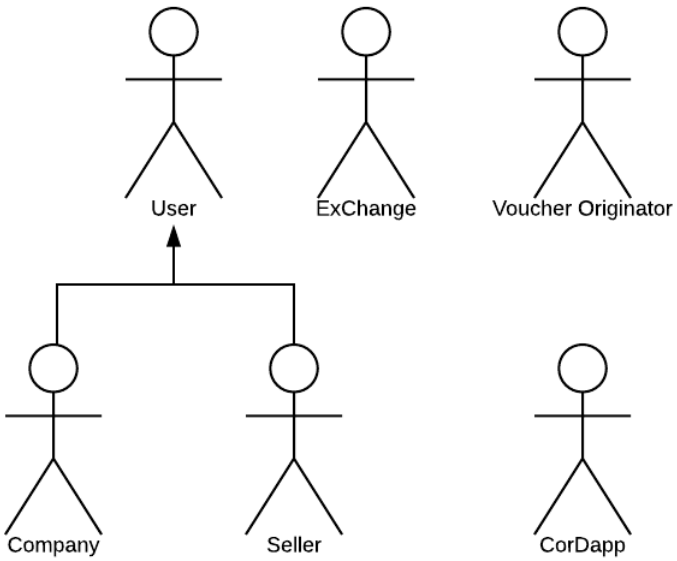

Figure 5.1: Diagram of the actors involved.

5.4.2 Use Cases

1. **Generic user use cases**:

- **Manage account**: A user can manage their own informations recorded on the application.
- **Adding credits**: A user can add credits, be it fixed value or currency, to their account.
- **Send payment request**: A user can initiate a transaction over the purchase of a good. At the end of the transaction, the value of the good will be added to their account. The user in this case is a seller.
- **Accept payment request**: A user can respond to a transaction over the request of purchase of a good. At the end of the transaction, the value of the good will be detracted from their account. The user in this case is a buyer.

2. **Company use cases**:
   - **Pooling credits**: A company can pool credits from its (willing) employee for a batch order. These orders can be meal orders. The pooling tries to address change leftovers by minimizing it over larger purchases.
   - **Redeem a fixed value credit**: Companies with a selling profile can redeem fixed value credits, such as vouchers, from their originators.
   - **Send payment request**
   - **Accept payment reques**
3. **Voucher originator**
   - **Issue credit**: Voucher originators, according to its contracts with companies, issue vouchers to users.

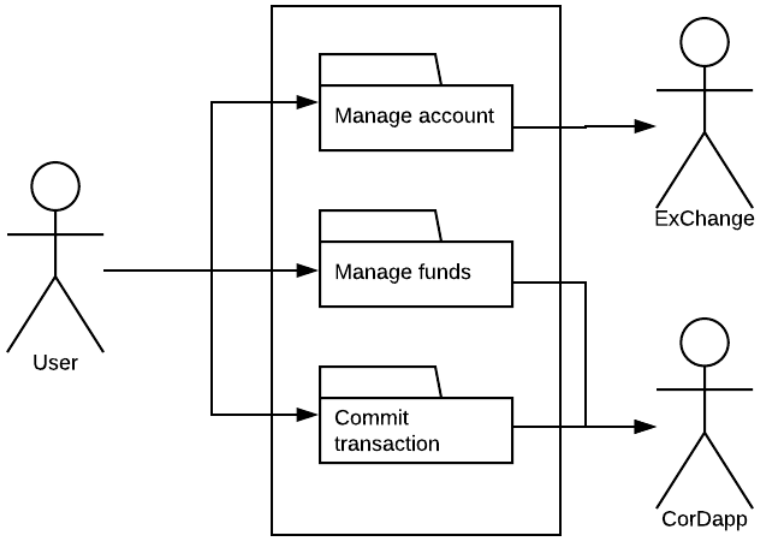

Figure 5.2: Diagram of the use case packages for a generic user.

Figures 4.2 and 4.3 show simplified diagrams of a generic user (a buyer or a seller) pertaining use cases, with a focus on the transaction use case package.

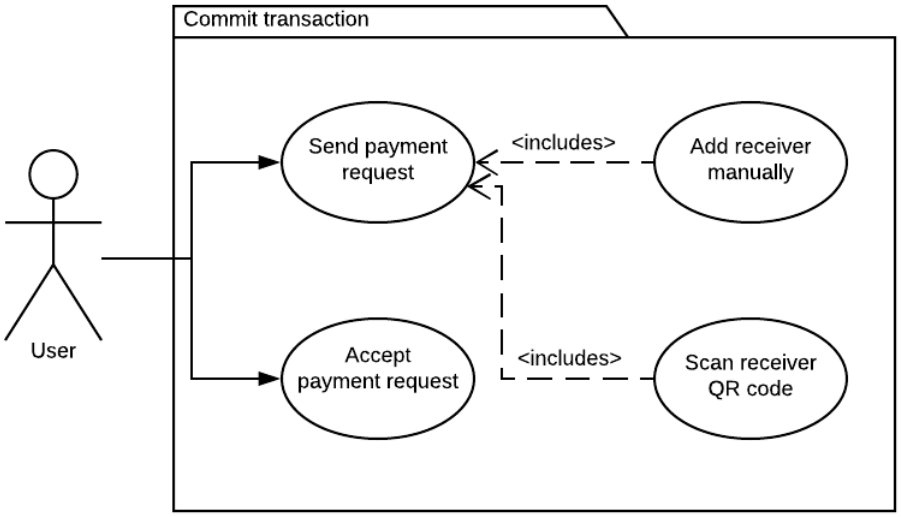

Figure 5.3: Diagram of a single use case package for a generic user.

The use cases can be clearly partitioned between account management actions, such as editing user information, associated credit card or company, fund management action, such as adding funds and tracking transaction history, and transaction committal actions, such as sending payment requests and accepting payment requests.

We will here on focus on the transactional procedures, as it's the one most closely related to the workings of distributed ledger architecture and Corda itself.

5.4.3 System architecture

Before delving into the behavior of the system, it can be useful to consider the architecture of the system. While we're relatively free to choose how to design the application, or the higher layer "interface" through which the user communicates, we have to take choices about how the application is linked to the distributed ledger underlying architecture provided by Corda. (Note: by high layer interface, it is not meant front-end, as the application can be a built as a complete stand-alone application interfacing the distributed ledger network). Thus, it is necessary at this point of the process to consider some more technical elements to choose how the system behaves and communicates as a whole.

We will consider the whole system as comprised of two separate subsystems. One is the ExChange application, that acts as an interface for user interaction, and the other is the CorDapp, that is the subsystem that manages the node interactions in the distributed ledger network. The CorDapp and its higher level interface do not have to reside in the same enviroment. In this prototype a more coupled solution has been preferred for an easier implementation, so the ExChange application resides in the same enviroment as the CorDapp.

The ExChange network in its entirety can be considered as the network comprised of users (buyers, sellers, companies, and voucher originators). In addition, the network can present different permissioning services that can be interfaced through the application, such as notaries, oracles or doorman (useful to provide a network map that would work as a map of affiliated shops or premises). Note that at this point in time Corda still does not provide complete implementation for some of this services (such as the network map), so their design and implementation will be postponed until a further date.

A CorDapp possesses a vault, that contains all the current and historic states that a node is aware of. The vault contains the credits (vouchers, currency) under the form of states, that make up the fund of an user account, and their history - how each state evolved.

States come in different types. Some states can be transferred through ownership change and some states can be splittable, for example a state describing the ownership of 20 vouchers can be split on necessity in two voucher states (19+1). A historic state is a state that has been consumed - it evolved from one (or more) state to another.

States evolve through transactions. Each transaction is validated by a smart contract. Smart contracts verify transaction validity, and are a set of rules a transaction has to satisfy to be valid. Different type of transactions are validated by different type of smart contracts.

A flow is the automatization of a transaction - different flows pertain to different transaction types. This architecture expects the creation of a Voucher Payment flow that deals with voucher payments and handling its leftover changes, when existing.

Flows and transaction are regulated with notaries nodes. Notaries guarantee uniqueness consensus and prevent double-spend in the system by running validity controls on the history of states.

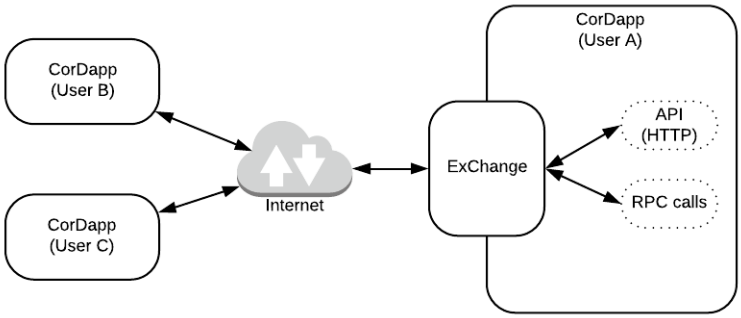

Figure 5.4: Simplified architecture of the system.

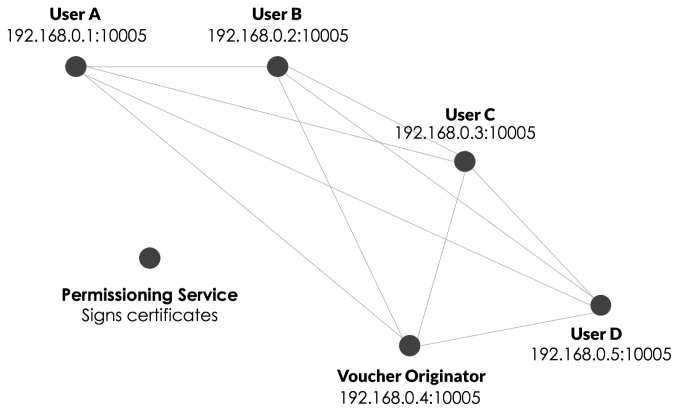

Figure 5.5: Diagram of the ExChange network.

5.4.4 System behavior

Every interaction that involves transactions is managed through the CorDapp. The ExChange application works as a mediator between the requests of the user and the underlying CorDapp. This premise makes it so every action pertaining such use case has to go through two layers: the ExChange app and the underlying CorDapp. For the following diagrams we will consider the ExChange application and CorDapp as one single object to underline how the system behaves and communication flows between nodes to commit a transaction.

Figure 4.4 shows the sequence diagram for a would-be purchase using vouchers (it includes also the initial voucher issuance from the originator). Note that each object in the diagram is a node, and the exchange of messages on a higher level happens through the ExChange application, while the flow is the real transaction that happens in the CorDapp and updates the ledgers of both parties with the transaction.

To understand the transactional process better, we will consider in detail how a voucher payment works and how the states in the ledger change.

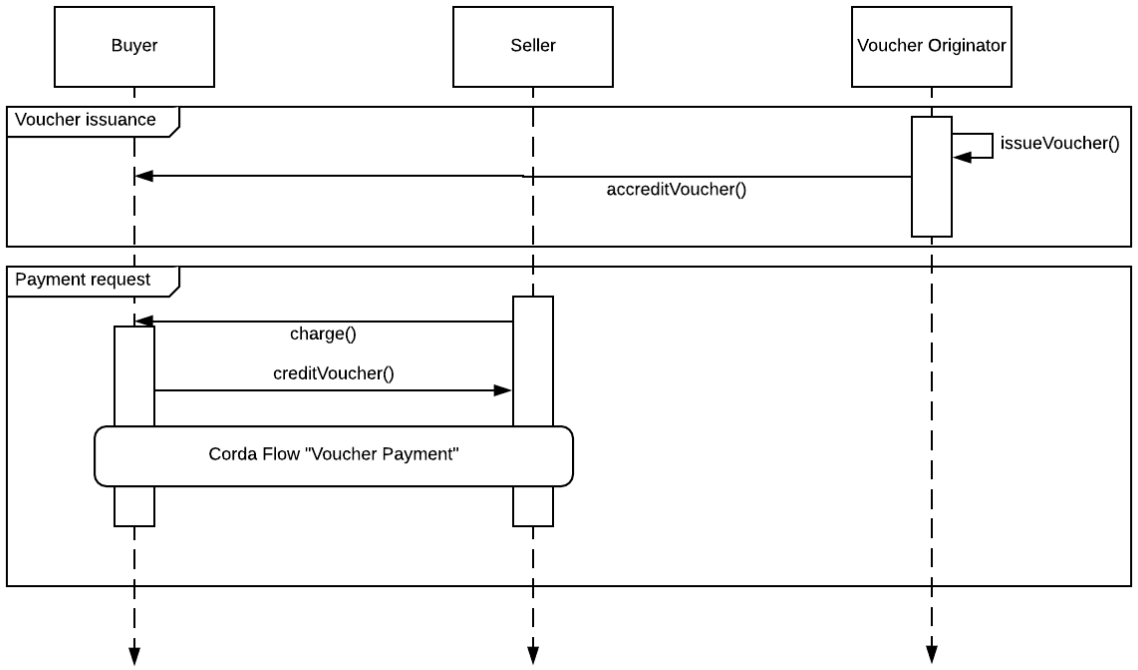

Figure 5.6: Sequence diagram comprising of voucher issuance and voucher payment.

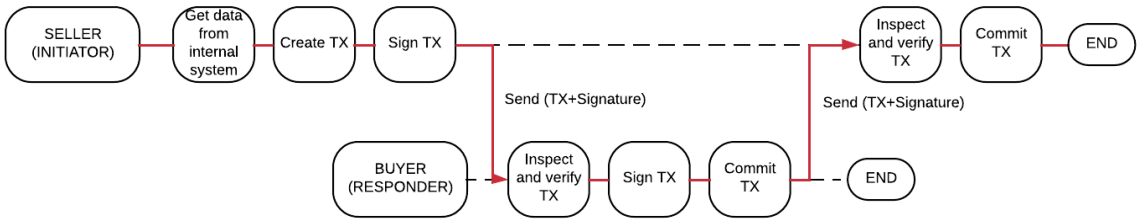

Figure 5.7: Diagram of the Voucher Payment flow.

5.4.5 Voucher payments

Two solutions have been elaborated to solve the issue of voucher payments and leftover change.

Both solutions provide advantages in terms of automatization of the voucher payment process and change processing - not considering the advantages and securities offered by the DLT itself.

The first solution, in case of mismatch between payment request and voucher face value presupposes the creation of IOUs in the vault of the seller. IOUs in this case are discount states representing the leftover change from the voucher.

Figure 4.8 represents the distributed ledger contents between the buyer and seller nodes, before and after a payment transaction of 4€ with a voucher with a face value of 5.20€ according to the first solution.

The buyer will retain the change for future payments and purchases from the same seller.

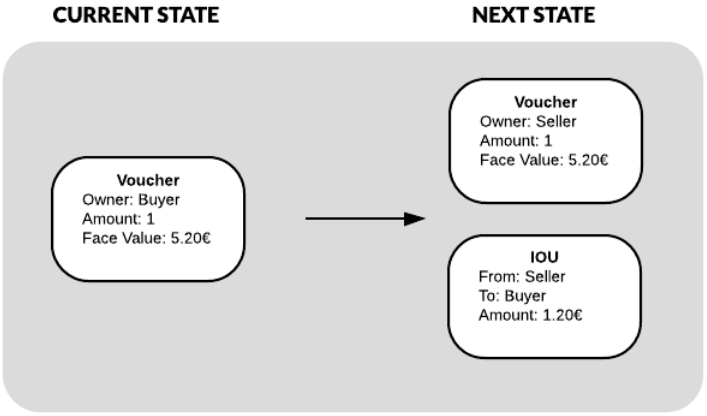

Figure 5.8: Input and output states by the "Voucher Payment" flow in the distributed ledger viewed by the buyer and the seller.

How the states are consumed in the transaction depends on the final implementation of the flow. The Voucher Payment flow can be setup so it consumes the most optimal combination of states, giving priority to IOU states - if existing in the seller vault - and then voucher states.

This solution presents no added risks from either parties compared to a normal transaction, and automatizes the handling of voucher's leftover change. It is somewhat more rigid than preferred, as the change's future usage is limited to the seller where it was generated.

The second solution enables transferrable change. After a voucher payment, a change state is generated in the buyer's vault, corrisponding to the change left over by the voucher. It is here necessary to introduce a second type of voucher state, called "Redeemable Voucher". Redeemable Voucher states represent list of vouchers detained by a seller. A seller can effectively redeem only the redeemable amount of voucher they own.

Figure 4.9 represents the distributed ledger before and after a payment transaction of 4€ with a voucher with a face value of 5.20€ according to the second solution. If a buyer uses a 5.20€ voucher to pay for a 4€ good, he is given a 1.20€ change state, which they can use without any limitation to the seller. The seller, on other hand, cannot redeem the 5.20€ voucher in its entirety, as he only obtained 4€ worth of it.

This arises the issue of how to deal with voucher fragmentation, as this leaves the seller open to eventual malicious "voucher exchange" attacks - that is an attack where a buyer might use a voucher for a small purchase, for example of 0.10€, to cash in the change and leave to the seller the burden of accumulating the 5.10€ of difference needed to redeem the voucher.

Voucher fragmentation can be dealt easily as Sellers can use their own funds in their accounts to issue change free from vouchers - the change can be any other kind of transferrable credit accepted by the platform, for example, currency.

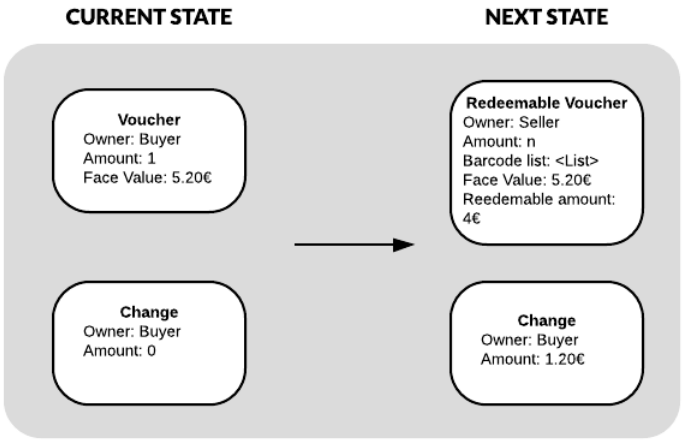

Figure 5.9: Input and output states by the "Voucher Payment" flow in the distributed ledger viewed by the buyer and the seller.

A note has to be said in regards of the second solution here shown. The entire implementation of such solution is only a thought experiment in regards to how to deal with voucher leftover changes, as the conversion of change leftover into an effective digital credit to be used in a transactional platform isn't regulated.

5.5 Implementation

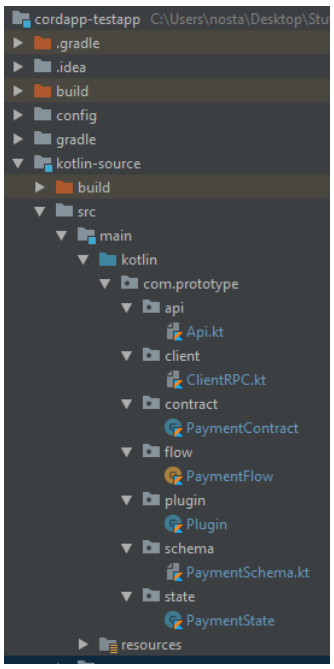

Figure 5.10: Structure of the Corda implementation

A test application implementing the basic communication between nodes and exchange of a "Payment" state has been implemented in Kotlin using the IntelliJ IDEA environment.

The test application regards only the CorDapp subsystem, and allows a seller node and a buyer node to agree on a payment, if it follows the following contract rules:

- The payment's value is strictly positive
- A node is not trying to pay itself

The CorDapp can be deployed as three nodes, Buyer, Seller, and Notary. The Corda Kotlin template has been used for the development of this test application.

5.5.1 State

The first class to define is the state class, representing the shared facts on the ledger. The class is immutable, meaning that the values of one instance of the class cannot be modified. Instead, consumed states become historic, and a new state with new values is put into its sequence.

```
package com.prototype.state

import ...

/**
 * The state object recording a payment agreement between two parties.
 *
 * A state must implement [ContractState] or one of its descendants.
 *
 * @param value the value of the payment.
 * @param seller the party issuing the payment.
 * @param buyer the party receiving and approving the payment to the seller.
 */
data class PaymentState(val value: Int,
                        val seller: Party,
                        val buyer: Party,
                        override val linearId: UniqueIdentifier = UniqueIdentifier()):
        LinearState, QueryableState {
    /** The public keys of the involved parties. */
    override val participants: List<AbstractParty> get() = listOf(seller, buyer)

    override fun generateMappedObject(schema: MappedSchema): PersistentState {
        return when (schema) {
            is PaymentSchemaV1 -> PaymentSchemaV1.PersistentPayment(
                    this.seller.name.toString(),
                    this.buyer.name.toString(),
                    this.value,
                    this.linearId.id
            )
            else -> throw IllegalArgumentException("Unrecognised schema $schema")
        }
    }

    override fun supportedSchemas(): Iterable<MappedSchema> = listOf(PaymentSchemaV1)
}
```

Figure 5.11: Payment state class

5.5.2 Contract

The contract class is used to define how an associated state evolves over time. In Corda, contracts also are able to impose what kind of transactions are to be allowed, so when input and output states do not satisfy a contracts rules, the transaction is considered invalid.

```
package com.prototype.contract

import ...

/**
 * A implementation of a basic smart contract in Corda.
 *
 * This contract enforces rules regarding the creation of a valid [PaymentState], which in turn encapsulates an [Payment].
 *
 * For a new [Payment] to be issued onto the ledger, a transaction is required which takes:
 * - Zero input states.
 * - One output state: the new [Payment].
 * - An Create() command with the public keys of both the lender and the borrower.
 *
 * All contracts must sub-class the [Contract] interface.
 */
class PaymentContract : Contract {
    companion object {
        @JvmStatic
        val PAYMENT_CONTRACT_ID = "com.prototype.contract.PaymentContract"
    }

    /**
     * The verify() function of all the states' contracts must not throw an exception for a transaction to be
     * considered valid.
     */
    override fun verify(tx: LedgerTransaction) {
        val command = tx.commands.requireSingleCommand<Commands.Create>()
        requireThat { this: Requirements
            // Generic constraints around the Payment transaction.
            "No inputs should be consumed when issuing an Payment." using (tx.inputs.isEmpty())
            "Only one output state should be created." using (tx.outputs.size == 1)
            val out = tx.outputsOfType<PaymentState>().single()
            "The lender and the borrower cannot be the same entity." using (out.seller != out.buyer)
            "All of the participants must be signers." using (command.signers.containsAll(out.participants.map { it.owningKey }))

            // Payment-specific constraints.
            "The Payment's value must be non-negative." using (out.value > 0)
        }
    }

    /**
     * This contract only implements one command, Create.
     */
    interface Commands : CommandData {
        class Create : Commands
    }
}
```

Figure 5.12: Payment contract class

5.5.3 Flow

The flow class allows to define the steps needed for a Corda node to perform an update on the ledger. In other words, flows allows nodes to issue new PaymentStates to the ledger. Every flow has two sides, the Initiator side, which will initiate the request to update the ledger, and the Acceptor side, which will respond to said request.

```
package com.prototype.flow

import ...

/**
 * This flow allows two parties (the [Initiator] and the [Acceptor]) to come to an agreement about the Payment encapsulated
 * within an [PaymentState].
 *
 * All methods called within the [FlowLogic] sub-class need to be annotated with the @Suspendable annotation.
 */
object PaymentFlow {
    @InitiatingFlow
    @StartableByRPC
    class Initiator(val iouValue: Int,
                    val otherParty: Party) : FlowLogic<SignedTransaction>() {
        /**
         * The progress tracker checkpoints each stage of the flow and outputs the specified messages when each
         * checkpoint is reached in the code.
         */
        companion object {
            object GENERATING_TRANSACTION : Step( label: "Generating transaction based on new Payment.")
            object VERIFYING_TRANSACTION : Step( label: "Verifying contract constraints.")
            object SIGNING_TRANSACTION : Step( label: "Signing transaction with our private key.")
            object GATHERING_SIGS : Step( label: "Gathering the counterparty's signature.") {
                override fun childProgressTracker() = CollectSignaturesFlow.tracker()
            }

            object FINALISING_TRANSACTION : Step( label: "Obtaining notary signature and recording transaction.") {
                override fun childProgressTracker() = FinalityFlow.tracker()
            }

            fun tracker() = ProgressTracker(
                    GENERATING_TRANSACTION,
                    VERIFYING_TRANSACTION,
                    SIGNING_TRANSACTION,
                    GATHERING_SIGS,
                    FINALISING_TRANSACTION
            )
        }

        override val progressTracker = tracker()

        /**
         * The flow logic is encapsulated within the call() method.
         */
```

Figure 5.13: Payment flow class

```
    /**
     * The flow logic is encapsulated within the call() method.
     */
    @Suspendable
    override fun call(): SignedTransaction {
        // Obtain a reference to the notary we want to use.
        val notary = serviceHub.networkMapCache.notaryIdentities[0]

        // Stage 1.
        progressTracker.currentStep = GENERATING_TRANSACTION
        // Generate an unsigned transaction.
        val PaymentState = PaymentState(iouValue, serviceHub.myInfo.legalIdentities.first(), otherParty)
        val txCommand = Command(PaymentContract.Commands.Create(), PaymentState.participants.map { it.owningKey })
        val txBuilder = TransactionBuilder(notary)
                .addOutputState(PaymentState, PAYMENT_CONTRACT_ID)
                .addCommand(txCommand)

        // Stage 2.
        progressTracker.currentStep = VERIFYING_TRANSACTION
        // Verify that the transaction is valid.
        txBuilder.verify(serviceHub)

        // Stage 3.
        progressTracker.currentStep = SIGNING_TRANSACTION
        // Sign the transaction.
        val partSignedTx = serviceHub.signInitialTransaction(txBuilder)

        // Stage 4.
        progressTracker.currentStep = GATHERING_SIGS
        // Send the state to the counterparty, and receive it back with their signature.
        val otherPartyFlow = initiateFlow(otherParty)
        val fullySignedTx = subFlow(CollectSignaturesFlow(partSignedTx, setOf(otherPartyFlow), GATHERING_SIGS.childProgressTracker()))

        // Stage 5.
        progressTracker.currentStep = FINALISING_TRANSACTION
        // Notarise and record the transaction in both parties' vaults.
        return subFlow(FinalityFlow(fullySignedTx, FINALISING_TRANSACTION.childProgressTracker()))
    }
}

@InitiatedBy(Initiator::class)
class Acceptor(val otherPartyFlow: FlowSession) : FlowLogic<SignedTransaction>() {
    @Suspendable
    override fun call(): SignedTransaction {
        val signTransactionFlow = object : SignTransactionFlow(otherPartyFlow) {
            override fun checkTransaction(stx: SignedTransaction) = requireThat { this: Requirements
                val output = stx.tx.outputs.single().data
                "This must be an Payment transaction." using (output is PaymentState)
                val payment = output as PaymentState
                "I won't accept Payments with a value over 100." using (payment.value <= 100)
            }
        }

        return subFlow(signTransactionFlow)
```

Figure 5.14: Payment flow class

5.5.4 Interaction

The front-end and API haven't been fully implemented (ExChange application), as they are outside the scope of this project. A command line interface has been implemented to interact with the network via command line. It is possible to run the client on the terminal window, with the commands:

```
cd project
./gradlew clean deployNodes
./build/nodes/runnodes
```

Figure 5.15: deployNodes task in the build.gradle file

With that it's possible to verify that the nodes and implemented classes are running correctly.

Chapter 6

Conclusions

The focus of this project work has been the exploration and viability research of Distributed Ledger Technology in the current market environment, to understand if a solution can carve itself a place in the market, can be profitable, how the landscape of DLTs is formed, and in what form could such a solution be presented.

We started from a general primer on the technology itself and came to understand what position in the market the technology currently holds, and how the major market players have been reacting to it. Evidence has shown a moderate interest in the technology, but with a growing engagement and flux of investments that is expected to grow exponentially in the next years.

After that a comparative analysis between three main technologies, Ethereum, Hyperledger and Corda revealed their major characteristics and difference in approach, showcasing which technology is more adequate for which kind of solution.

Corda emerged as the preferred technology for a distributed solution with strong focus on financial use-cases that require a certain degree of privacy, regulations and security.

Finally, we evaluated a possible solution design and its implementation, focusing on the usage of vouchers in transactions. In this, the issues about leftover change for voucher usage have been addressed by two different solutions. This design has shown how a distributed ledger architecture can be implemented to enable a seamless payment platform, automatizating every transaction and data recording thereof through distributed ledger technology. It has also shown how such an architecture can solve the issues of cost, latency, single-point-of-failure, eventual reconciliation and need for a trusted intermediary for a payment use case.