UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Exploring the use of blockchain in academic management systems

MESTRADO EM SEGURANÇA INFORMÁTICA

Diogo Edgar Andrade Sousa

Trabalho de projeto orientado por:
Prof. Doutor Alysson Neves Bessani

2020

Agradecimentos

Esta tese é o culminar de cinco anos na FCUL, pautados por muita dedicação e esforço, e que não teriam sido possíveis sem o apoio de muitas pessoas com quem me cruzei neste percurso.

Hoje sou a árvore cuja semente um dia regaram. Como nem todos os visados compreendem inglês, faço questão que consigam ler aqui o meu sentido agradecimento.

Primeiro, tenho de agradecer ao meu pai e à minha mãe, por terem-me dado a oportunidade de apostar na minha educação e voltar ao Ensino Superior. Sem eles não estaria aqui, por mais que uma razão.

Um obrigado ao Prof. Tibúrcio Azevedo, cuja aulas foram essenciais na minha preparação para os exames nacionais e a quem tenho uma dívida a ser saldada em breve.

Agradeço também a todos os docentes que tive nestes cinco anos em Ciências, sem os quais não teria descoberto a minha verdadeira vocação na Informática, e que contribuíram indelevelmente para a minha formação.

Um duplo agradecimento está reservado ao Prof. Alysson Bessani. Não só teve das cadeiras mais interessantes da licenciatura e mestrado, cujos tópicos são abordados nesta tese, como orientou-me ao longo deste trabalho num ano cheio de dificuldades trazidas pela pandemia. O seu apoio foi inestimável e merece todo o meu reconhecimento.

Agradeço também a todos os amigos que fiz e que, de uma forma ou outra, me acompanharam até ao "último nível do jogo". Um abraço especial ao Fred, Jorge, Rui, Miguel e Gonçalo, assim como aos *Amis de Bonjour* pelo incentivo diário durante o processo de escrita.

Um especial obrigado a todos na **qubIT** por me terem acolhido durante a tese e permitido crescer nesta nova aventura pós-faculdade.

Por fim, e porque neste momento está ali à espera que eu acabe de escrever como em tantos outros dias, agradeço à minha esposa Rita pela sua infinita paciência e apoio. Prometo aqui solenemente compensá-la por todos os fins-de-semana que passámos em casa porque havia trabalhos para entregar e prazos a cumprir. O tempo agora é só nosso.

Per aspera ad astra

Resumo

A tecnologia blockchain é um tópico em voga, com a sua visibilidade a estar directamente associada às criptomoedas, cuja grande expansão foi impulsionada pela Bitcoin (BTC).

Apesar desta "explosão" recente, os conceitos base da tecnologia blockchain já têm décadas. O Hashcash, por exemplo, já utilizava um sistema de validação semelhante à Bitcoin para mitigar *spam* e ataques de negação de serviço.

Embora os termos Bitcoin e blockchain tenham uma forte conotação, não são equivalentes. De forma concreta, uma blockchain é uma estrutura de dados que armazena informação numa cadeia de blocos ligados (*block+chain*) e cuja sequência é validada através de criptografia, particularmente funções de síntese (*hash*). Aumentar a cadeia requer que haja consenso entre os participantes sobre qual o próximo bloco a adicionar, consenso esse que é atingido por diversos mecanismos como sendo o *Nakamoto consensus* da Bitcoin, baseado em *Proof-of-Work*. Uma vez adicionado um bloco, essa informação é propagada aos participantes, existindo uma representação global e coesa da blockchain para todos. Um ponto importante é que não é possível apagar dados uma vez inseridos, o que, combinado com a verificação criptográfica e o uso de consenso, torna-a resistente à manipulação do seu conteúdo.

É claro que esta tecnologia tem um grande potencial disruptivo, o que por sua vez tem causado uma "corrida ao ouro" digital, levando a que muitas organizações criem as suas próprias blockchains para posicionar-se no mercado. Os casos de uso já comprovados aproveitam as características da blockchain e variam entre aplicações financeiras baseadas em *tokens* que tiram partido da descentralização e a transposição para blockchain de actividades em que a auditabilidade e resistência à manipulação é altamente valorizada como sendo a gestão de cadeias logísticas.

A **qubIT**, enquanto criadora de soluções de sofware, propôs este projecto para explorar o uso desta tecnologia no seu âmbito principal de negócio, os sistemas de gestão académica. Foi assim ideada uma prova de conceito com o objectivo de permitir validar documentos académicos assim como partilhar dados entre (e através das) instâncias Fenix das instituições da Universidade de Lisboa.

Inspirando-se num dos casos de uso comprovados de blockchain, o rastreio de cadeia logística, a visão desta prova de conceito poderia ser descrita como "rastreio de docu-

mentação académica". A questão a responder é: como podemos avaliar a validade de um documento académico para além da assinatura digital que lhe está aposta? Um exemplo seria o caso em que o documento está assinado por alguém que, naquele momento, não tinha legitimidade para o fazer.

O objectivo é publicar informação numa blockchain e, através desta, estabelecer um registo cronológico imutável que pode ser utilizado para adicionar maior legitimidade e transparência a um documento académico.

Existem dois principais desafios para o uso de blockchain para documentos académicos. O primeiro provém do cumprimento do RGPD. Por exemplo, sabendo que os dados ficam de forma permanente na blockchain, como lidar com a revogação de consentimento sobre dados já publicados. Relacionado com este, surge também a falta de suporte legislativo para valores armazenados na blockchain (não está estabelecido o seu valor probatório), especialmente quando comparada com o suporte existente para as assinaturas digitais.

O primeiro passo foi escolher uma blockchain para a implementação. Começámos por decidir entre a utilização de uma rede aberta, já existente, ou criar a nossa rede através de uma blockchain permissionada.

Aqui analisámos o uso de Ethereum. Com a sua rede e suporte para várias linguagens de desenvolvimento, era uma opção viável. Tinha a vantagem de podermos beneficiar da infraestrutura já existente, incorrendo apenas nos custos associados ao seu uso. Contudo, devido a esses custos e à inexistência de regulamentação para os enquadrar, optámos por criar a nossa própria infraestrutura através de uma solução permissionada.

Para tal, analisámos várias implementações até que a escolha foi reduzida a dois candidatos, Fabric e Corda, que comparámos extensivamente. A abordagem tomada contrastou-os em três áreas: governança, que dados são visíveis e como são acedidos pelos participantes; suporte, quão activa é a equipa de desenvolvimento e como são resolvidos incidentes; e arquitectura, explorando os aspectos técnicos relacionados com suporte para linguagens de programação ou armazenamento de dados.

No fim, foi escolhido o Corda por ser baseado em Java, podendo assim ser rapidamente integrado com o conjunto de tecnologias já utilizadas e acelerar o desenvolvimento da prova de conceito. Contribuindo para esse processo foi também criado um ambiente de desenvolvimento com as ferramentas necessárias para a criação de aplicações Corda (*CorDapps*).

A salientar, contudo, que o Corda não é estritamente uma blockchain, mas sim um livro-razão (*ledger*) distribuído. Esta distinção aplica-se porque, no Corda, não existe um estado global partilhado mas sim conjuntos de registos que são visíveis para as entidades consoante as transacções em que participam.

Utilizando a versão *open source* do Corda foi construída a Academic CorDapp. Esta aplicação permitia que fosse publicada e partilhada informação entre várias instâncias

do Fenix. Os dados publicados agregavam informação relativa a estudantes, cursos, notas e documentos e, com o seu registo no *ledger*, permitiam estabelecer uma cronologia imutável do progresso académico de um estudante numa instituição.

Esta aplicação funcionava de forma autónoma, através da linha de comandos, com a sua integração no Fenix condicionada a um parecer favorável após análise pela equipa de soluções de negócio.

Nesta avaliação, determinou-se que o Corda tinha alguns aspectos técnicos mais fracos (p.ex., suporte para bases de dados) mas era capaz de cumprir os requisitos para integração. O ponto principal levantando na avaliação foi que a proposta de valor adicionado pelo uso de blockchain era insuficiente quando comparada aos mecanismos de assinatura digital qualificada já existentes, especialmente se utilizado o padrão LTV.

Foram então re-avaliadas as opções que tínhamos previamente excluído no processo de escolha do Corda. A solução em Ethereum continuou inviável pelas razões que tinham determinado a sua exclusão inicial: falta de suporte legal assim como a dificuldade em estimar os custos de operação. Foi discutida a possibilidade de reimplementar a aplicação utilizando o Fabric ou outra solução permissionada. Isto, apesar de resolver as limitações tecnológicas do Corda, não conseguia sanar a comparação com as assinaturas digitais que comprometia a viabilidade comercial.

Assim sendo, a equipa deu um parecer desfavorável e decidiu que o projecto não iria avançar para a fase de integração em produção.

Não obstante este desfecho, este projecto foi frutífero para estabelecer uma base de conhecimento sobre as implementações blockchain existentes, permitindo guiar adequadamente projectos futuros, assim como deixou preparadas ferramentas para acelerar o processo de desenvolvimento e aumentar a competitividade em futuras oportunidades neste espaço de negócio.

Palavras-chave: Blockchain, Fabric, Corda, Governança, Fenix, Documentos, Assinatura Digital

Abstract

The term blockchain is a “hot topic” in technology, with its visibility in the mainstream being boosted by the cryptocurrency boom brought forth by Bitcoin (BTC).

An enterprise solutions developer, **qubIT** proposed this project to explore the use of blockchain technology in their main business scope, academic management systems. For that, a proof of concept was devised that used blockchain with the goal of providing document validation and data sharing across (and through) Fenix instances of ULisboa’s schools. Taking inspiration from a proven use case, supply chain traceability, the project’s vision could be described as “academic document transparency”. The main concerns regarding the project’s outcome were possible GDPR issues (e.g., revoking consent) and a lack of regulatory backing when compared to digital signatures.

First step was to choose a blockchain implementation. We started by deciding between using an open, existing blockchain network or building it using a permissioned blockchain.

For this we considered Ethereum. It was a viable option but, due to regulatory concerns, we decided to opt for a permissioned blockchain.

After exploring different blockchain offerings, we narrowed the choice down to Fabric and Corda, comparing them in depth.

In the end, Corda was chosen and its open source version was used for the Academic CorDapp. This application allowed Fenix instances to publish information on a shared ledger, using it to chronologically track a student’s academic achievements. It functioned in a stand-alone manner, with a favorable evaluation from the Business Solutions team determining its integration into Fenix production systems.

Corda was found lacking in some technical aspects (e.g., database support) but capable. The key issue was that the value added by using blockchain was insufficient when compared to qualified digital signature mechanisms, specifically the LTV standard, which led to the decision of not pursuing the project any further.

Despite this, this project had a positive outcome in building know-how regarding blockchain that will help guide future ventures in this area.

Keywords: Blockchain, Fabric, Corda, Governance, Fenix, Documents, Digital Signature

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The term blockchain is a “hot topic” in technology, with its visibility in the mainstream being boosted by the cryptocurrency boom brought forth by Bitcoin (BTC) [4]. Accompanying that visibility has been investment, a bet on the “next big (disruptive) thing” which blockchain technology is positioned to be. Companies now offer a myriad of “Powered by Blockchain” [5, 6] solutions. Consider, for instance, the Forbes “Blockchain 50” [7], which lists companies valued at 1 billion dollars or more, filled with every big cloud player and smaller contenders that are fighting it out to find their space in the market.

We must, however, separate the marketing hype from the technology itself. There are several products layered on top of blockchain, but some of those offerings do not hinge on a blockchain *per se* being used. Just because something is “Powered by X” is not equivalent to “Requires X to function”.

Despite its recency [8], blockchain technology is based on already well-known computer science elements. Merkle trees [9], used for block verification, have been around since the 1970s. Hashcash [10], which used similar principles to validate its tokens, was developed in the 90s as a proof-of-work token to curb Denial-of-Service (DoS) attacks and spam emails.

Bitcoin[1] propelled the concept of a blockchain into the limelight by promising to solve the number one problem with digital currencies: how to prevent a token from being spent twice. This is known as “double-spending” and is a major issue for any digital asset. Even for assets with a centralized entity, an arbiter of spent vs. unspent, this may occur. Latency and concurrency are some possible causes, due to the distributed nature of those systems. Any token that can be spent more than once undermines its value. Solving it properly is a necessary condition for token viability.

With the Bitcoin paper, ongoing blockchain research received a lot of attention and funding, leading up to now where just Bitcoin alone is valued at around 118 billion dollars [11], representing more than 60% of the global market cap for cryptocurrencies.

[1]The network, not the cryptocurrency itself.

Bitcoin and blockchain still retain a strong connotation to this day but are not equivalent. In strict terms, a blockchain is a data structure used to implement a distributed ledger, a facsimile of accounting ledgers, that stores information in a chain of blocks, akin to a linked list (hence the name), that are validated using cryptography. This validation mechanism, paired with the linked data structure, imbue it with desirable properties such as immutability and tamper evidence/resistance.

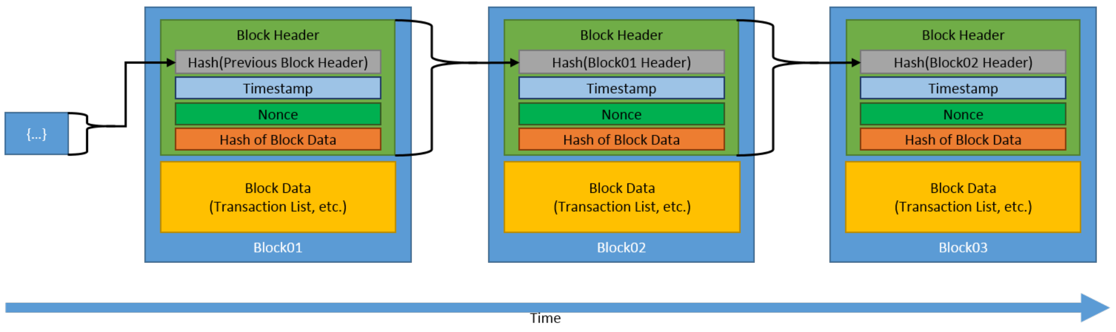

Figure 1.1: Blockchain data structure example [1].

Digital assets are the most visible part of the blockchain ecosystem. Seeing the success of Bitcoin, several other companies and individuals launched their own cryptocurrency offerings: Coinbase lists more than 500 tokens currently being traded [12].

This resulted in several public blockchain implementations, many of them short-lived as a public blockchain is only as strong as its adopting base. Catastrophic events have not been rare: consider the YAM token implosion [13], where a bug in a smart contract plummeted a token value to zero, “erasing” a valuation of 500 million dollars in the span of two days. High profile initial coin offerings (ICO) such as the Venezuelan Petro [14] are also a source of volatility and mistrust in the concept. But there is more to blockchain than cryptocurrencies.

Other use cases [15] for blockchains range from asset tracking (supply chain transparency [16] and land registries [17]) to pure fintech applications based on the exchange of custom tokens. For these classes of problems, generally an existing network such as Ethereum is favored, due to its position as the second most popular blockchain (behind Bitcoin) and its powerful smart contract language, and is used as a base instead of creating an implementation with a specific “twist” to cater to it. Ethereum has even created the ERC-20 [18] standard to support these token endeavours. One such token is the Basic Attention Token (BAT) used by the Brave Browser to reward users that watch ads [19]. They differ from cryptocurrencies by pegging their value to goods and services (as a kind of loyalty card points) instead of fiat currencies such as the euro or dollar.

Most of the major cloud service providers, Google being the notable exception, now offer managed blockchain services. However, their cost may be prohibitive when you are still trying to figure out if blockchain is the best/right answer for a given problem: it sometimes is not and you get “Blockchain for Blockchain’s Sake”[2] applications popping up.

No matter what is being done with it, it is clear that blockchain is strongly disruptive, here to stay and, with its appeal to ideals such as decentralization and anonymity in a progressively more mobile and privacy-weary society, will continue to mature and produce valuable business ventures.

1.1 Motivation

When we look at mainstream blockchains,[3] most of the readily available implementations nudge (and some push) you towards using a managed solution such as Amazon Web Services (AWS) (e.g., Corda) or IBM Cloud (e.g., Hyperledger Fabric[4]). If a user wants to avoid using a managed solution, for reasons such has having full control of their data (sometimes even legally mandated to do so) or trying out the technology without incurring in any costs by using infrastructure they already have available, they must understand how to configure their networks beyond what the basic “Your First Network” tutorials explain.

Consider Fabric: it has a “build your first network” tutorial [20][5] that depends on complex bash scripts being executed at specific moments. Running those is easy, but adapting them for end users’ needs is not so simple. This means that the barrier for entry is still higher than desirable, but obviously not insurmountable.

qubIT, as an enterprise solutions developer, wants to be prepared for the occasion when blockchain is the correct answer for a given problem. Such problems are, for now, few and far between but it wants to seize business opportunities as they come up in emerging technologies. As such, a way to quickly deploy the required infrastructure to support and onboard developers, while abstracting away most of the configuration work and learning curve, is an important tool to have. **qubIT** embraces the idea of having the abstractions already in-place and letting the developer focus only on creating the end product.

This project is, first and foremost, **an exploration of the blockchain ecosystem**, to ascertain if it is viable enough to invest further. For this purpose, we are looking to **qubIT**’s core business, academic management systems, and evaluating the use of blockchain by building a proof of concept around the validation of academic documents. We also want to simplify the developer’s work, offering a capable toolbox that can be used for experimentation.

Academic documents are, for the most part, still paper based. While being created by digital means and stored as PDF templates, the exception being official forms already printed in bulk, it makes no sense to us that institutions continue printing them to man-

[2]Using blockchain just to claim its use for marketing.

[3]Actively developed and well backed.

[4]Henceforth referred as Fabric.

[5]This tutorial has now been removed from Fabric documentation.

ually stamp and sign. In most cases, that document is then immediately digitized for safekeeping and portability. **qubIT** has already implemented solutions for signing digital documents that are legally compliant.[6] We will now try to tackle the question: *when we receive a digitally signed document, how can we determine its legitimacy beyond the signature?*

Full disclosure: for the duration of this project, I was and currently am a full-time employee of **qubIT**, working on the R&D team and responsible by driving the expansion into new business areas. I was fully involved in the development and business decisions for this project.

1.2 Goals

This project has the following specific goals:

1. Create tooling to facilitate development of blockchain applications;
2. Evaluate blockchain offerings to choose a base framework;
3. Produce a proof of concept application for the validation of digital documents targeting the Fenix ecosystem in ULisboa;
4. Evaluate that proof of concept with regard to its technological and business aspects;
5. If proven viable, integrate the tooling into the **OMNIS** software platform and flesh out the proof of concept into a fully featured solution.

We will follow a "product first" instead of a "technology first" philosophy (built as a product from the get-go) so being commercially viable is a *sine qua non* condition. The Business Solutions team will be consulted regarding project decisions.

Every scope within the **OMNIS** software platform has a prefix, used for content aggregation, so this project and associated modules were placed under the name **PACTUS**, meaning "compromise".

1.3 Contributions

With this project, we explored the blockchain ecosystem by analysing and contrasting multiple frameworks, with emphasis being placed on Fabric and Corda, building necessary know-how about existing blockchain implementations for future projects.

Having chosen Corda as our base framework, we created a development environment, bundled with all the essential tools, and produced documented templates to help developers create their own Corda projects.

[6]This solution is currently being used in Ciências.

Using those tools, we created a proof of concept CorDapp to share information across Fenix instances and validate academic documents that could be integrated into a production system.

1.4 Document Structure

The contents of this dissertation are organized as follows:

Chapter 2 - Background and Related Work - provides an overview of the related technologies and main stakeholders for this project.

Chapter 3 - Blockchain Frameworks Comparison - presents the problem statement, purpose and approach taken. Documents the choice of a blockchain from the viable candidates.

Chapter 4 - Academic CorDapp - details the development process of the CorDapp, as well as auxiliary tasks.

Chapter 5 - Analysis - explains the challenges faced and solved during the development process, as well as the business focused analysis of the Academic CorDapp, with regard to determining its viability as a product.

Chapter 6 - Conclusion and Future Work - concludes with the final results and personal takeaways from the project.

Chapter 2

Background and Related Work

In this chapter, we will define key concepts, starting with the distinction between a distributed ledger and a blockchain, followed by the major axis concerning the later implementations. Then a brief description of mainstream protocols and, by extension, managed blockchain solutions.

Finally, pertaining to our main challenges, we discuss digital signature mechanisms and their legislative backing, in Portugal and in the European Union, as well as data protection issues.

2.1 Key concepts

2.1.1 Distributed Ledger

A ledger is an accounting term for a book or registry recording transactions, debits or credits, amongst a group of participants, while keeping a tally of the total balance, be it current or past.

A distributed ledger, commonly referred as Distributed Ledger Technology (DLT), is an electronic extension of that concept [21]. Instead of a physical book, there is a database, distributed across a P2P network, that operates in a decentralized manner. Every participant uses that ledger to record transactions, public or private, involving itself and/or more parties with the ledger state being then synchronized across the network. The decentralized nature contrasts with a centralized ledger, a single central registry for every operation, which is functionally equivalent to a single-writer database.

In the absence of a centralized point of control, such data structures require a consensus operation to maintain consistency. It does not, however, require it to be consistent across every participant, just the relevant parties for that portion of the ledger. What matters is that eventually there will be a consistent view of the ledger state across all nodes [22]. After ordering, ledger updates form a valid sequence that can be fully chronological traced.[1]

[1]Knowing exactly what was the value recorded at an exact moment.

Distributed ledgers are append-only, meaning you cannot remove information after it is inserted (e.g., setting the current status of something by appending a new record). It can replicate any database where data can be removed by marking data as outdated or unnecessary instead of deleting it. The rationale for this behaviour is simple if you consider the real world analogue: accounting ledgers must be balanced and subject to audit at any time, so no information must ever be removed. It is an essential point for applying proper governance.

Being append-only grants it high resistance to tampering as an adversary trying to change the ledger in its favor must change a significant portion of it, across all copies. This makes DLT useful in untrusted environments where Byzantine faults, malicious or not, may occur.

We note that the terms DLT and blockchain are used interchangeably but a distributed ledger can be implemented in various ways, with blockchain being the most popular option. Not every distributed ledger uses blockchain though. IOTA [23], for instance, uses a Directed Acyclic Graph (DAG) to store ledger information. These alternative implementations have some advantages (e.g., lower transaction fees) and disadvantages (e.g., harder to reach consensus) [24], as expected, but have gained little traction when compared to blockchain.

2.1.2 Blockchain

A blockchain is a data structure implementing a distributed ledger. A chain starts with a block called the seed or genesis block. From that, blocks are appended to form the chain. Each new block is composed of several records, a timestamp and the hash of the previous block (see Figure 1.1). A record is an arbitrary collection of data: it can be details of a financial transaction, a PDF document, an hash value or even a digital cat.[2]

Participants create transactions and through some mechanism or incentive (one point where blockchain implementations vary), transactions are selected for the next block to be added onto the chain. Generally, a blockchain network has built-in incentives so participants keep creating the next blocks, essential to keep the network running. This operation requires participants to agree on what is the next block to be added, as there are multiple proposals, which is achieved through consensus (another point where there are several variations). The chain of blocks represents a shared global state that is stored across all the nodes. Since every participant must be aware of every update and agree on the state, messages are broadcast amongst them using a gossip based protocol as all members are not known upfront by each other (in the public version of blockchain).

[2]https://www.cryptokitties.co/

The data contained in the chain can be validated through the hash values. To validate block $i+1$ we take block i, calculate its hash and see if it matches to what is in block $i+1$. The previous block's hash is used for calculating the current block's value, propagating it across each link. As long as the hash function is sound [25], computing and comparing the hash demonstrates the ordering of the blocks as replacing a block with another would require creating a hash collision, something known to be difficult for most cryptographic hash functions in current use.

Besides this, there are other ways to attack a blockchain that take advantage of the operating conditions of the network. An example is selfish mining [26]: miners collude to hide newly-generated blocks and create a fork that eventually takes over networks where the main chain is automatically the longest, receiving all the block rewards. Attacks are as diverse as the incentives and consensus mechanisms in the blockchain landscape [27].

We can further characterize blockchains by two major axis: how they manage participants and how they achieve consensus, each with its own sub-categorizations.

Participants

Regarding participants, the major division is between **permissionless** and **permissioned**.

The permissionless model is the one used by Bitcoin and Ethereum (among many others) and is pretty simple: anyone can join the network at any time and actively participate in it by proposing transactions or mining blocks. Participants are anonymous, identified only by their address on the network, something that makes this extremely attractive for cryptocurrencies.[3] Trust is also distributed across the system, requiring the use of Byzantine Fault Tolerance (BFT) consensus (malicious participants are expected due to the financial incentive). As a blockchain collapses if everyone leaves, these networks require an incentive model to encourage sustained participation. As examples, both Bitcoin and Ethereum offer transaction fees and block rewards to miners [28,29] while IOTA will only validate your transaction if you validate two others (no fees are present) [30]. Blockchains using a permissionless model are described as **Open**.

Permissioned blockchains, by contrast, are networks where participants are all known and identified. These are generally maintained by a company or group of companies to leverage blockchain-based services. Within this concept, we can further split it into three sub-categories: private, where membership is closed and block generation is centralized, generally for internal company use (e.g., blockchain for logging); consortium, an extension of the private model where a number of entities control block creation (e.g., different banking institutions using a common blockchain); and hybrid, where there is a mix of private and public facing blockchain, like Kadena [31] which offers a public network and a private one (Kuro) that can intercommunicate.

[3]Decentralized currency remains a major theme and driver in blockchain usage.

Consensus

In a permissioned network where participants are well known, consensus can use more traditional mechanisms such as PBFT [32] or even eschew BFT (consider the case for private networks) in favor of simpler, non-BFT algorithms, such as Raft [33]. Fabric, the most popular permissioned blockchain, uses a plugable consensus module [34], allowing the end-user to adjust it to its security needs. One such option is a BFT orderer built upon BFT-SMaRt [35].

There are, however, more unusual approaches. Hyperledger Sawtooth uses Proof-of-Elapsed-Time (PoET) [36], whereby a participant generates a random waiting time in a secure manner (generally hardware)[4] and waits for that time to elapse. First one to "wake up" claims the rights to produce next block.

When we have an open blockchain, where participants come and go as they please, all assumptions provided by a permissioned network are gone. There is a need to solve consensus through a protocol with unknown participants, over an unstable and unreliable network. There are several proposed solutions to this issue, with the two most common being Proof-of-Work (PoW) and Proof-of-Stake (PoS).

Proof-of-Work, which is the mechanism used in most open blockchains, consists in solving a computationally difficult puzzle. In the case of Bitcoin, the miner must vary an integer counter on the proposed block until the hash produced has a certain number of leading zeros. With every leading zero, the difficulty rises, allowing for adjustment on the block creation rate. Every participant tries continuously to generate the desired hash and the first node to achieve it broadcasts that information across the network, adding the next block to the chain and receiving the reward.

One criticism of this mechanism is that it involves significant computations that consume a lot of energy. The BTC network consumes as much power as a small country [37], while generating exahashes (10^{18}) of computational power dedicated to solving such puzzles.

To improve upon this, Proof-of-Stake was created [38] as an alternative. Under this model, the next block is chosen from a combination of random number generation plus a function of the stake held by a participant on the network. We can look at it like a lottery draw: whomever has the largest stake has more "lottery tickets" and has more chances to mine the next block. This removes unnecessary computations but has the clear disadvantage, requiring adequate balance, that whoever owns more tokens has a better chance of receiving more tokens.

Apart from these protocols used by the mainstream networks, there are several others currently in use in niche blockchains, like Proof of Burn [39], where participants show that they sent tokens to some unrecoverable address, proving that they have legitimate

[4]Sawtooth is backed by Intel and used Intel Software Guard Extensions (SGX) features in PoeT-SGX.

interest in adding a block (or risk wasting their tokens).[5]

2.2 Public Blockchains

2.2.1 Bitcoin

Bitcoin's first block was mined in 2009 and is the most visible blockchain network. It is a permissionless network that uses Proof-of-Work (dubbed "Nakamoto Consensus" [4]) as a consensus mechanism.

Bitcoin has an interesting feature built-in: there is an announced collapse, as only 21 million bitcoins can be mined. This limit is hardcoded and attempts to fork the network to solve it have been thwarted by the largest mining pools that hold a strong position in the currency and expect its value to go up as a limited commodity for financial operations. When this limit is reached, block rewards will reach zero and only transaction fees will be transferred.

Although Bitcoin allows for smart contracts, its support is very limited, so the network is almost entirely focused on currency trading. Bitcoin's anonymity made it the currency of choice of more nefarious purposes, being linked to money laundering, extortion and *ransomware* [40], but that scenario is changing rapidly. Efforts are taking place in several countries, Portugal included [41], to regulate cryptocurrencies and mainstream payment processors such as Paypal are starting to accept them as means of payment [42].

2.2.2 Ethereum

Ethereum [29] is the *de facto* standard for mainstream blockchain applications. It is a permissionless network that uses Greedy Heaviest Observed SubTree - GHOST, a Proof-of-Work algorithm for consensus [43]. GHOST differs from the Bitcoin PoW by choosing the main chain not from the longest chain (vulnerable to selfish mining) but the one where most CPU power has been applied by looking at the subchains as branches in a tree. In turn, rewards are given not only to whomever solves the puzzle but also the previous blocks' generators (called "Uncles"), brought along as a subchain. Launched in 2015, it quickly grew in popularity and, in May 2020, its chain reached 10 million blocks.

The reason for this popularity is two-fold. On one hand, the Ethereum cryptocurrency, named Ether (ETH), learned from Bitcoin's mistakes and created a mechanism to update itself without requiring hard forks. It also removed a hard limit on the number of Ether available, controlling currency inflation. Its governing organization has repeatedly taken steps to ensure that the network keeps operating at a constant step (difficulty is adjusted such that a block is mined every 10 minutes).

[5]This only works if the network has infinite token capacity to keep running (e.g., Ethereum). Bitcoin could never use this model.

Ethereum also positions itself as a distributed computation platform (a “World Computer”) and allows for the creation of smart contracts using several languages.[6]

Commonly used are Solidity, a Turing complete language similar to JavaScript, and Vyper, a more Pythonic language. A smart contract is simply a piece of code that executes in an autonomous fashion, with the terms of the agreement specified as code. Since the Ethereum Virtual Machine is itself Turing complete, smart contracts are unbound on what operations they can perform, which is a risk (e.g., compute a fork bomb). To limit this, Ethereum uses the concept of “gas”, a fee that must be paid to use the computation power of the network. There is a maximum amount of gas that can be used by any contract, restricting the kinds of computation that can be performed. The user cannot create a Denial-of-Service (DoS) as they will eventually run out of gas or hit the gas limit after spending all their money. They also worked to create standards that make their network attractive for third parties, such as the ERC-20 [18] standard for custom tokens.

2.3 Permissioned Blockchains

2.3.1 Tendermint

Tendermint is not a blockchain in itself but a group of three products. Tendermint Core [44] is a modular BFT consensus engine that can be used in your blockchain of choice and is the default choice of Hyperledger Burrow [45]. The Cosmos SDK [46] is their blockchain offering, packaging the tools needed to build applications operating under the Cosmos Network. Finally, they have the IBC protocol [47] to ensure messaging between distributed ledgers and allow different blockchains to interact.

Its activities revolve around the Cosmos Hub, a group of connected blockchain networks that communicate between them using the IBC protocol.

2.3.2 Ripple

Ripple is an open-source [48] semi-permissioned blockchain that was designed mainly for currency transfers. The semi-permissioned model is defined as follows: anyone can participate and exchange XRP tokens but transaction validators are appointed by the network operator. This model made Ripple very attractive to banking institutions and novel business ventures such as peer-to-peer lending, which was Ripple Labs’ main goal. This model is also similar to a consortium blockchain.

For consensus, Ripple uses its own Ripple Protocol Consensus Algorithm (RPCA) [49], a multi-round mechanism where each participant votes on the transactions that should be committed to the ledger, with the rounds repeating until 4/5 of the participants agree on an update.

[6]https://ethereum.org/en/developers/docs/smart-contracts/languages/

They also maintain RippleNet, a global network where that exchange takes place and also offers their software to whomever decides to implement it.

2.3.3 Fabric

Fabric [34] comes from the Hyperledger project, an open-source initiative, supported by The Linux Foundation and several corporate partners, that develops distributed ledgers and associated tooling. It has produced several implementations such as Sawtooth or Iroha, with the most prominent of those being Fabric, initially developed by IBM.

Each of those implementations focuses on a specific element (e.g., Sawtooth was built around Intel's SGX hardware). For Fabric, that focus is on extensibility by allowing most of the key components to be modular. This comprises the ordering service (where the consensus protocol is used [35]), the membership service which dictates governance and the gossip protocol for information spreading (e.g., choose between Apache Cassandra [50] or other implementations).

Another strong point of Fabric is that it was the first to consider smart contracts written in general purpose languages such as JavaScript, Java or Go, a different take from the domain specific languages used in platforms such as Ethereum. This allowed for a much faster development cycle and a lower barrier of entry, promoting fast adoption.

Fabric follows a permissioned model and is the choice implementation of leading cloud service providers like AWS [51] and IBM Cloud [52].

2.3.4 Corda

Developed by R3, Corda [53] is described as a business-focused platform with a marketplace where CorDapps, distributed applications similar to smart contracts, can be sold and installed by clients. For this purpose, R3 maintains its own Corda network, similar to RippleNet, while offering an open source version of Corda as an introduction to its paid enterprise counterpart.

Corda is a special case in this section as it is not strictly a blockchain. It is a distributed ledger that, by sharing many characteristics with blockchains (e.g., validating transactions by recursing over the transaction chain until its origin), competes in the same business space. One key distinction is that there is no shared global state.

Like Fabric, they decided to use a general purpose language to power its platform, in this case the languages running on the Java Virtual Machine or equivalent (e.g., GraalVM [54]). As a consequence, CorDapps can be written in Kotlin, Java or any other JVM compatible language.

We will delve more in depth about Corda (and Fabric) in Chapter 3.

2.4 Managed Solutions

Cloud providers have moved into the blockchain space, with Amazon, Azure, Oracle and IBM all offering managed solutions [51,52,55,56], be it as a pure blockchain endeavour or a “blockchain powered“ product such as Amazon’s Quantum Ledger Database [57]. One notable absence is Google Cloud, which does not offer any blockchain related products at the time of writing.

Amazon’s offering are the most prominent and diverse. One option is the Quantum Ledger Database (QLDB), positioned as a fast, scalable ledger with a centralized authority that provides access to a cryptographically verifiable complete history of changes to the stored data. It also offers a managed blockchain solution that can use Fabric or Ethereum, so you can build an open blockchain on top of a managed solution. Both these options integrate with other AWS products such as AWS Key Management Service for participant management and even uses QLDB to keep track of block ordering.

IBM Cloud, on the other hand, only offers support for Fabric but leverages its extensive knowledge of the platform (it was incubated by IBM and they are still major contributors) to offer a fully integrated solution for blockchain management.

It seems clear to us that managed solutions will continue to grow and further integrate with blockchain frameworks to the point where deploying your own infrastructure will rarely be considered due to the ease and low cost of quickly creating a fully functioning network on any chosen provider.

2.5 Main Challenges

In this section we discuss two important concerns regarding the use of blockchain for academic documentation, the key use case considered in this work: digital signatures and legislative backing; and GDPR compliance.

2.5.1 Legal basis for document digital signatures

Asserting a fact, any piece of information, on a public blockchain allows you to have a timestamped proof of such assertion that, after block confirmation, can never be changed and is valid for as long as that blockchain keeps operating.

A digital signature is a legally recognized instrument that can be put on an electronic document to assert who signed it, that it was done without coercion, and the content has not been changed. It is an extension of a physical signature, whereby someone agrees and validates the content of something or shows its identity by signing it [58].

Digital signatures, paired with a timestamping service [59], give you the same properties (integrity, non-repudiation) without the need for a blockchain and associated cost of operation. However, the signature may not be independently verifiable if being viewed on a machine where trusted certificates have not been bootstrapped and therefore cannot validate the signature chain. A more catastrophic scenario would be that one of the certificates in the chain gets compromised and added to a revocation list or confirmed by the Online Certificate Status Protocol (OCSP), breaking a signature that was valid at the time.

This is were Long Term Validation (LTV) comes into play. LTV is a standard set by PDF Advanced Electronic Signatures (PAdES) [60]. That standard, transposed to portuguese law [58], captures the full state of the certificates at the time of the signing, effectively timestamping the signature and certificate chain.

Since the status is captured and embedded in the document, the signature can be validated on a much later date, even after that certificate has been revoked or expired. The document can, by itself and using the information within, ascertain if the signature was valid at the time it was created. LTV's security is backed by the strength of digital signature algorithms, which we can consider equivalent to the blockchain guarantees, but without the need for maintaining a costly network [61].

It also solves one key compliance issue: extensions can be added to specify who authorized the signing, under what pretenses was the document signed or even what position does that person occupy in an institution. This is extremely relevant for academic institutions as signing a document backed by that institution can only be done, by force of law, by authorized personnel such as department heads or service chiefs. That information can be tagged to a document (e.g., "Signed on DD/MM/YY by person A, who holds position B") and protected by the LTV signature.

Establishing an extreme scenario where our chosen blockchain suffers a hard fork (e.g., new genesis block) or stops being used, LTV signed documents retain their properties regardless of the world state.

2.5.2 General Data Protection Regulation

General Data Protection Regulation (GDPR) [62] is, as the name states, the EU data protection regulation that, albeit already transposed into portuguese law, is in a transition period as it is a very complex legislation and imposes a compliance burden on most organizations. ULisboa is still under the three year transition period, but is expected to be fully compliant in 2021.

One example of such complexity in academic information systems is grades: schools across ULisboa do not agree if a student's grade should be public information, required for transparency. Some schools, Ciências included, keep the grades private unless they have explicit consent from the student.

There are two possible compliance issues at the forefront of our blockchain plans that we must always keep in mind. First of them is consent [63]. GDPR has strict requirements for consent, including that every party is identified, what is the purpose of the data and what systems will be used to process it. This consent also needs to be kept up to date whenever a new party is added to the system or a new process is created. This is nothing new as Fenix already has those notices implemented (e.g., sending information to a banking institution for identification card purposes). The issue is that consent may be withdrawn at any time and smart contracts, which run in a fully autonomous fashion, must take this into account lest they incur in compliance violations.

Paired with this is the so-called "right to be forgotten" that clashes with the append-only nature of blockchain. It is a "what if" scenario as, for the moment, information stored in the systems is considered essential for rendering the service and is therefore protected from removal. But, consider the scenario where someone does not want its thesis to be public or that it graduated and there is jurisprudence allowing such request. ULisboa may not be able to comply without compromising the whole system as blockchain is append-only.

2.6 Final Remarks

One trend that we found while analyzing the technologies is that most of the blockchain frameworks which offer open source versions use them to create traction for their paid/-managed counterparts (e.g., Corda and Corda Enterprise) or visibility for the creator's network (e.g., Ripple and RippleNet).

For our purposes, the main advantage of using DLT is that complete traceability and auditability is already built-in. For academic management systems, transparency is important and a strong selling point, much more than data security, provided it can be offered while keeping GDPR compliance.

Chapter 3

Blockchain Frameworks Comparison

In this chapter, we first give a brief description of the host institution to contextualize the project. Following that, we present the problem that we are trying to solve through the use of blockchain. Furthermore, we provide a comparison between Corda and Fabric that guided our choice of implementation platform for the proof of concept.

3.1 qubIT - Software Solutions

qubIT is an enterprise solutions developer, specialized in higher education academic management systems. Its main product is the FenixEdu-based Fenix solution used every day by students, teachers and administrative staff to manage all aspects of the academic process. This solution is also multi-channel, offering a mobile app called myFenix.

Its most visible customer is, of course, ULisboa, with Fenix being currently deployed in most of its schools, including Ciências.

More than a “one size fits all solution”, **qubIT** has been continually improving its offerings by adding new features, offered as Fenix modules, and provides tools such as the Workflow engine, a state machine augmented with several logic operations and actions, that allows end-users to model processes like signing up to optional classes or requesting a certificate. The company also provides training and support regarding its modules so users can use them independently.

Tying everything together is **qubIT**’s low-code software platform, **OMNIS**[1], built to empower developers, either in-house or partners, and allow them to build software tailored to their needs. Proper tooling and abstractions allows them to focus on what they do best, developing, and letting the platform do all of the heavy lifting.

This project touches on both of these aspects as adding a new scope, in this case blockchain, would allow **qubIT** to offer new Fenix modules as well as make its low-code platform attractive to potential partners in the blockchain space.

[1]https://omnis.cloud/

3.2 Problem Statement

Continuing from Section 3.1, ULisboa is comprised of eighteen schools with sixteen of them using a FenixEdu-based solution (Fenix) as its academic management system, each with its own instance. They share a common core and are configured according to the school's specifications, ranging from the grading scales to how degrees are chronologically organized. Each instance has its own application and database servers, running in complete isolation from the others.

When deciding what would be the key use case for the proof of concept, ULisboa was, at the time, finishing their implementation of Erasmus without Paper [64]. This system allows for the exchange of information between home and host institutions about a student in three key moments: registering the student, signing the learning agreement and, after its concluded, sending the grade transcripts, all through the use of digitally signed documents. Those documents are exchanged using the EWP Network so both the home and host institutions can easily validate their provenance.

We had the idea of expanding that concept in two ways. First, adding support for other kinds of documents besides grade transcripts such as achievement certificates or attendance records. As an example, external institutions could request attendance records before approving credits transfers. Second, allowing for external entities, without the need for the establishment of formal protocols, to validate those documents.

This is to hedge against the case where a student, current or former, produces a signed document that is not valid: it may have been revoked or signed by someone who did not have the authority to do so.

The validation mechanism being considered is that, when generating a digital document, a unique code would be placed on it. That code could then be used in a service request that would output the transactions that led up to that document being created, establishing a chronology of events, and also validating that the document matches with our records.

In a future phase, this could also be used to link physical documents to their digital representations (e.g., through a QR code printed on the document), augmenting physical watermarks beyond only capturing the instant at which they are placed.

Knowing that supply chain transparency as one of the proven blockchain use cases [65,66], **qubIT** decided to pursue this as a proof of concept under the vision of "academic document transparency".

There is also another idea being explored in the background. If a student chose to do a Minor in another institution within ULisboa, the grades associated with that minor are sent to the base institution, generally by email, and introduced into Fenix manually. This is, of course, error prone and would break any academic tracking. We even have a more notable scenario in the General Studies degree offered by FLUL. Students can freely choose classes in any of the partnered institutions, with information then having to be centralized and synchronized before producing relevant documents such as enrolment certificates. If using blockchain for Fenix interactions proves to be sound, we could consider using the same mechanism to augment current workflows to work in a distributed manner, using blockchain as an exchange medium. Since blockchain provides a consensus mechanism and coherent representation, concerns about duplicated messages or command ordering would be abstracted away. Although this favors a permissioned network, it was not a must-have condition when we evaluated the use of public blockchain.

This concept ties into a planed Fenix feature (federated communication between instances) that is currently under development. Keeping this in mind, we are considering that the relevant data for our academic documents spans multiple Fenix instances and will use that when defining our network topology.

3.3 Public vs. Private Blockchain

Having defined our purpose, we must keep in mind its business requirements while developing the product, taking into consideration the possible costs and revenue streams.

The first major choice was between using an already established public blockchain or building a private network using a permissioned blockchain framework.

By choosing a public blockchain, the infrastructure would already be present and we would leverage the value added by the service for revenue. Our concept deals with documents where at least some personally identifying information is always present, so GDPR requirements would also be in play.

For public blockchains, there was one clear choice of platform, Ethereum. It is a stable network with the required features for addressing the problem.

Describing our high-level concept, consider that a digital document establishes some information (e.g., a student earned its degree on date X with final grade Y, having attended K out of N classes). Even if block size was not a limitation (Ethereum has an average block size of 30 kB [67]), we still could not place the document on the blockchain with the identifying information still present. Using encryption would also be too cumbersome with all the key management and distribution required and the impossibility of updating information if a key was compromised. A simpler solution is to hash the document and place that value in the blockchain. After the block is confirmed, we would have a timestamped proof that the document existed in that form at that point in time. We are eschewing signing the document on purpose as this would allow us to work around the legal requirements for who can sign in representation of an institution (see Section 2.5.1).

To validate the document, the user would calculate its hash and compare it with the value stored in the corresponding block. To compromise this system and fake a document (e.g., by replacing it with another), you would need to break preimage resistance and create a hash collision, something known to be difficult for most hash functions [25].

A solution could then be offered that makes the connection between the hash, the full document and the block where the hash is stored, allowing any third party to verify it easily. That party can also verify it directly on the blockchain using a tool such as Etherscan [68] and a known address associated with the transaction, be it either the destination or origin.

Although choosing Ethereum for its smart contract support, the procedure devised can be performed even without using one. By controlling two addresses on the Ethereum network, one could issue a value transfer of zero ETH between them and add the hash as a comment, a mechanism similar to a Proof of Burn, storing that in the transaction data. We can use a smart contract to add some extra logic, but since we are only interested in the block timestamp, it is not strictly required.

From a technical standpoint, this process is simple enough that it can be validated as safe (even the smart contract version) and performs that operation with minimal gas costs. But, albeit minimal, public blockchains are, in a way, "pay to play", meaning they always have some cost associated, usually in the form of transaction fees. Since this cost is directly tied to usage, it is not incurred upfront but funds or other resources must be available to operate on the network should the need arise. Gas costs could also rise to ensure that our transactions are picked within a certain time frame.

If we consider the volatility associated with cryptocurrencies, there is a clear and present probability of signing on to a deal (e.g., 1 € per academic document for the next 5 years) and incurring in loses due to rising cryptocurrency prices. Buying Ethereum, required to pay the gas costs, is an unregulated activity [69]. If you add to that the requirements for acquisition of goods and services required by public institutions, it becomes complicated to sell a product that can not be used (much like a car that can not be fueled) by the clients. We could take on that cost, and accompanying risk, but demand would have to be estimated beforehand which is hard to do for an untested service with no market parallels. We might purchase a large amount upfront and end up losing money due to market devaluation or the service not being used.

Another concern against using this model was that putting the document's hash on a public blockchain would not add anything that is not already provided by digital signatures using the LTV standard [58, 60]. Considering that, by design, a hash provides no information about the document that produced it, a user will require access the associated document for any meaningful transaction. In addition, LTV does not require the blockchain to be operating or event accessible to validate the content and timeline of the document produced (see Section 2.5.1).

Another regulatory issue on the public blockchain side is that there is no legally recognized way that an institution can claim a blockchain address as its own. The lack of legal support could be surpassed by building up trust somewhere else: an institution could request a certificate attesting that a certain address on the network is theirs and that they are responsible for its transactions henceforth. Notwithstanding, the strength of that document would be built upon digital signatures, begging the question of why not use digital signatures directly. Pairing that with a system that provides access to the signed PDF on demand, given some form of authentication, would be a much simpler proposition that does not hinge on unregulated financial activities.

In the end, we decided against using public blockchain for these three reasons and turned our attention to exploring the blockchains presented in Section 2.3.

3.4 Fabric vs. Corda

The two main candidates for our project were Fabric and R3's open source version of Corda. Fabric was the default candidate, as the *de facto* standard for permissioned blockchain applications. Corda caught our attention due to their data privacy model and, most importantly, being JVM-based.

In order to decide which one to use, both were compared across three major categories, with several components each. We followed the base framework outlined by Moezkarimi et al. [3] and added components where we felt necessary to contrast both options. Our choice of components is present in Table 3.1.

Category	Description	Components
Governance	What data is visible, who can access it and how	Network topology, Governance model
Support	How likely it is that updates will continue to be published and how easily can you receive assistance	Consortium Participants, Development Team, Documentation, Community
Architecture	Encompasses the design choices and architectural details of a system	Language support, Smart contracts, Consensus, Privacy, Transaction type, Hardware, Scalability, Interoperability, Storage, Deployment

Table 3.1: Categories for blockchain comparison [3].

3.4.1 Governance

Governance extends beyond data privacy: it dictates what information is available, to whom, and how can it be accessed according to an organization's policies.

Using a permissioned blockchain, we intend for each school to be represented by a network peer, including Reitoria that should be placed in an oversight position.

Both Fabric and Corda support that and allows us to place Reitoria in the critical path of certain operations (e.g., approving new degrees), be it as an endorser (Fabric) or required signer (Corda). Since the networks are permissioned, that node's identity will be clear and known to every party.

As for data access, again both frameworks allow us to limit access to information as we see fit.

Fabric offers channels to create compartmentalization of information to "need-to-know parties" [34] while Corda has that concept built-in: a node stores a separate ledger for each party it interacts with through transactions [53]. This results in one key difference: in Corda there is no shared world state. A node has its Vault, transaction storage, and its own copy of the ledger that may be different from other nodes with whom it has communicated. Interaction between these ledgers is possible if supported by a specific flow that makes the other party aware of the states being used in the transaction. There is also no notion of channel in the sense of subscribing to updates: peers must be added explicitly to be made aware of a transaction.

Each Fabric channel is backed by a separate blockchain, with the possibility of communicating between themselves if required, and the world state is shared amongst every participant of a channel.

Regarding smart contracts, both frameworks offer mechanisms to manage and update what versions are authorized to be used on the network.

Since governance requirements are met by both, there is no clear advantage in this category. We then looked at the ecosystem surrounding each framework.

3.4.2 Support

As stated in Section 2.3.3, Fabric is part of the Hyperledger project and is itself a re-branding of an IBM project, IBM OpenChain. It quickly rose to prominence as the most visible Hyperledger project due to strong backing and technical contributions by IBM.

The code is fully open-sourced and available on GitHub [70]. The community pulse is very positive, with 101 pull requests being merged from September 1st to December 31st [71]. Fabric uses Jira for issue tracking and had 28 open issues for the same period. Also available on GitHub, documentation is ample and very complete: Fabric not only has a full set of documentation across every version as there is Fabric related documentation by several providers such as Azure and AWS.

For gauging community engagement, we looked at the official forums, GitHub and StackOverflow metrics. The Fabric mailing lists is active, with questions being posted and answered daily. On GitHub, Fabric has 248 listed contributors, with a core team of 107 members under the Hyperledger organization [72]. As for StackOverflow, the *hyperledger-fabric* tag has 5,264 questions associated, with 44.5% answered[2] and a good number of users (> 30+) actively replying [73]. If we take into consideration the associated tags, Hyperledger has more than 12000 questions being asked and replied.

Corda is mainly backed by banking institutions [74] with some consulting firms like Everis or Accenture also participating. None of the Big 4[3] are present in that space though. Perusing the apps available [75], most of them belong to banks as expected, since Corda positions itself as being tailor fit for digital asset transfers.

The open source version is available on GitHub [76], along with associated projects such as the Deterministic Sandbox. The community pulse is strong, with 306 pull requests merged between September 1st and December 31st, but 142 issues, with the oldest being from 2017, still open. As Fabric, Corda offers documentation on GitHub, with separate repositories for the open source and the Enterprise version.

The Corda open source repository has 183 contributors, but the Corda organization has only 7 people [77].

The official mailing list has 330 topics since 2018, with questions following main Enterprise version releases. On StackOverflow, there are 2,085 questions with 33.5% unanswered [78]. Most of the questions are answered by the same 3 members of the development team that feature on Corda's official blog publications.

Overall, Fabric shows a clear advantage in this category, something expected since a community is a self-reinforcing mechanism: a strong and vibrant community, more than the product, drives its own growth. Despite the variety of Hyperledger projects, Fabric seems to "cut through the noise" and retain critical mass in terms of adoption by cloud providers and will continue to be developed in the future. Corda is also well supported but the community activity reflects its focus on the Enterprise product, where support is offered as a paid service.

3.4.3 Architecture

Language support. Language support considers what can we use to create smart contracts i.e., Chaincode for Fabric and CorDapps for Corda. Unlike Ethereum, we cannot implement operations on top of these blockchains without using smart contracts as there are no primitive operations involved.

Fabric is written in Go, but offers Software Development Kits for both Java and

[2]Note that on StackOverflow an unanswered question is one where the original poster did not accept a specific answer as the best, something very common with one-off questions.

[3]Deloitte, PwC, EY and KPMG

JavaScript (Node). All of them offer the same features but there are some advantages to each choice. Since Fabric development is Go focused, that SDK is the first to receive new features. Using Node allows you to have a common language for both front and back end, simplifying development. Java is supported but the baseline version is Java 8. There are also some issues with the serialization of complex objects for which the common advice is to use the Fabric Gateway SDK[4] and write the chaincode in Go or Node.

Corda is built with the JVM (or variations) in mind and supports any language compatible with it. Notwithstanding, its favored development language is Kotlin and the SDK is offered both in Kotlin and Java, complete with JavaDoc. From a feature perspective, they are equivalent.

Smart Contracts. Unlike Ethereum, where the language used by smart contracts was designed specifically for that purpose, both Corda and Fabric make use of general purpose languages to implement their version of a smart contract. While this drives adoption as more developers are familiar with the languages used, it cuts both ways as there is nothing to stop a developer from making mistakes.

When a blockchain follows an order-then-execute model, a smart contract should execute in a deterministic form, so that with the same input every node reaches the same end state and consensus is achievable. Non-deterministic execution breaks those guarantees and may thwart consensus.

Fabric uses a execute-order-validate model, whereby the chaincode is first executed, ordered and then committed after validation. This allows it to use non-determinism and, by extension, does not require it to limit the usage of language features. Since this support extends to three different languages, Fabric is especially affected by unsafe coding patterns that must be kept in check on a per-language basis. Analysis of tooling support [79] for the Go language, for example, shows that common Go tools (e.g., golint) are not enough to ensure code safety and more specialized tools must be used.

Corda requires determinism and tries to enforce it by recommending that their smart contracts (CorDapps) to be run using a deterministic JDK. That feature was still under development (the Deterministic Sandbox project on Github) and has now been replaced with a Java module that can be imported to replace non-deterministic classes.

Using JDK-based languages allows it access to linting tools for code quality but those are not aware of the deterministic requirement.

Also, there is no compiler help or linting associated with its annotations, making it easy for a developer to commit mistakes (for more on this, see Section 5.2).

Consensus. When using permissioned blockchains, consensus mechanisms can take advantage of every peer being well-known so more common algorithms can be used.

[4]https://hyperledger.github.io/fabric-gateway-java/

For consensus, Fabric refers to its ordering service, a modular component that allows the user to choose what better suits its needs. By default, it uses **`etcd`**, a library that uses Raft [33] as the underlying algorithm. Raft, in this case, is crash fault tolerant (CFT) and does not offer a Byzantine Fault Tolerance (BFT) version. If the user requires BFT, it can take advantage of the pluggable nature of Fabric and use an externally maintained orderer based on BFT-SMaRt [35] or Tendermind Core [44]. Fabric also allows the use of multiple ordering services, so some channels can use BFT and others CFT for greater throughput.

Corda applies consensus through the Notary, a service that determines if a state has not been used before (uniqueness) and if the transactions leading up to it are valid (validity). Corda also purports a pluggable consensus, by allowing for Notary clusters or implementing a custom Notary service. However, documentation states that any other use beyond a single notary are still experimental[5] as well as creating your own notary implementation. There is a repository detailing the use of notary clusters with other algorithms [80], but the results are also experimental. Production use seems to hinge on a single Notary being used, effectively centralizing the system and removing the need for consensus.

Privacy. Fabric supports the use of channels to keep transaction data private to the relevant parties. In addition to channels, since everything is hash-oriented, Fabric has visibility models that ensure a party can validate a transaction using only its hash, with no access to the private data being passed. Other options include encrypting the information with AES before placing it on the channel, therefore only allowing participants with the key to read it. They can, however, still endorse the transaction [81] despite not being aware of its content. All of the information is kept separate as for each channel there is a different blockchain backing the corresponding world state.

As for participant privacy, since Fabric uses a gossip protocol to communicate between peers [82], a node is aware of what members are present in a channel.

Corda's approach is similar. A party is aware of a state if it is included in the flow that creates it either as a participant or as an observer. Since there is no shared world state, each node only knows the information in its own Vault and transaction storage and with whom it has interacted during transactions. When more information is required for validating a transaction (e.g., by a Notary), it can be requested from proposing peers that have options to offer it in an anonymous way.

Since Corda uses point-to-point messaging, only the network map service knows every party and makes that information available on demand. Parties may not be aware of the presence of others on the network, nor do they need to as there is no shared ledger state.

[5]https://docs.corda.net/docs/corda-os/4.5/running-a-notary.html

Transaction type. Dealing with permissioned networks, transaction finality is derived from its consensus algorithm. One could, for example, use Nakamoto consensus [4] in a closed network but finality would still be probabilistic with regard to the length of the chain built upon a block. Since participants are well known, choice favours algorithms were finality has been demonstrated, meaning that committed transactions will not be reversed.

In Fabric, the world state is the best facsimile of a ledger: a key-value store. A transaction is simply a proposal to create or change the value associated with a specific key, so transaction validations focus on the transaction being authorized and asserting the order of those updates is consistent.

While most blockchains follow an order-then-execute transaction flow, deciding first what operations will be performed, Fabric uses an execute-order-validate approach [34]: a node locally executes the transaction then asks other nodes to endorse it. Once the endorsement requirements are fulfilled, that transaction is pushed to the ordering service for consensus. After that, the transaction is returned to the endorsing peers for further validation (e.g., that it has not been made invalid by some transaction committed while it was in flight) and finally committed to the ledger.

Corda is focused on asset transfers so its transactions follow a concept called unspent transaction output (UXTO), also used in Bitcoin, meaning that only unspent outputs can be used as inputs. In concrete terms, a transaction is a proposal that consumes zero or more states (e.g., creating a new asset consumes no states) and produces zero or more states (e.g., removing a participant consumes all of its states but produces none). An unspent state is one that represents the current status of that information.

Corda has no shared world state, so only parties involved in a transaction have to execute and validate it. Each node stores states in the Vault and transaction data, having a representation of its ledger that is consistent only in regard to other participants in the same set of transactions.

Hardware. Hardware requirements are important as they determine inherent running costs. Cloud resources are reasonably priced nowadays but if we could use hardware that we already have that is underutilized, even better.

Fabric's documentation does not specify any hardware requirements, so any machine that can run a Node application can be considered a baseline. Since a Raspberry Pi meets that threshold, its clear that it will run in modest hardware unless you need high throughput. With efficient digital signature mechanisms there is no need for dedicated hardware for cryptographic operations.

Corda, in a similar vein, will run on any machine that is able do run Java applications, with different levels of performance. This is familiar territory as **qubIT**'s solutions are all Java based.

Documentation for Corda Enterprise lists requirements [83] for estimated node sizes. From experience, the bottleneck on Java applications ends up being memory and associated garbage collection. Recommendation is 4GB so any 16GB machine should be able to handle it. Since we predict the nodes will be low volume, one option would be to deploy them in tandem with the Fenix instances and take advantage of unused resources.

Scalability. Performance with permissioned blockchain generally depends on consensus [84, 85]: adding more participants requires more messages being exchanged before consensus is reached, therefore affecting throughput.

Scalability is not a key concern for our project since network participation is well defined from the start. For ULisboa, we are considering a maximum of 20 nodes (one for each school and Reitoria, plus a node for offering external services) and 190 channels/ledgers, which is well bellow the values where performance drops dramatically. Combined with operations that do not need real time confirmation (we are still considering if we will perform validations on-chain or off-chain), we are confident that both Corda and Fabric can handle the network load with ease.

We do not predict much growth in the number of peers participating on the network, as we are considering that each consortium of schools under an University has its own deployed solution and they communicate through the external services.

Interoperability. Interoperability is important for future expansion. If schools can communicate within their institution in a federated manner, it would be advantageous to deploy this solution elsewhere and have it communicate seamlessly with already existing networks in a "federation of federations" of sorts. As companies start deploying blockchain solutions for different problems, this issue keeps gaining more visibility: Tendermind, mentioned in Section 2.3.1, created the IBC protocol [47] for such purpose, but still has little traction.

Fabric and Corda support information exchange but only within their own network: Fabric's applications can interact with multiple channels while on Corda this is simply a matter of adding references to the necessary states.

To create that kind of "blockchain of blockchains", some external solution would have to be used. At the time of writing, Hyperledger has released Hyperledger Cactus [86], a project focused on blockchain integration, allowing for communication between blockchains like Fabric, Corda, Hyperledger Besu (an Ethereum client [87]) or Quorum [88], a strong choice for multi-chain or hybrid blockchain projects.

Storage. Fabric supports two types of storage: on-chain and off-chain. On-chain is data associated with the key-value pairs, essentially the world state. This is stored using either LevelDB [89] or CouchDB [90], with the later using JSON and allowing for richer queries. To stop the blockchain from growing too quickly, off-chain storage is used when transactional data is too large or must be able to be updated. This data is then stored elsewhere, referenced in the transaction and nodes that chose to store a copy can still interact with it. One advantage of the shared world state is that nodes can be brought up to speed on a channel by requesting information from other peers, which can be used to recover a node that is in an inconsistent state.

Corda offers the same model as Fabric but most storage is all "on-chain" as if to say in-node, with each node having its own Vault and transaction storage. Off-chain storage is mentioned in the documentation, but there are no examples as how to configure it. Corda takes advantage of only having to store its view of the ledger to keep size down, but that has to be paired up with small contract sizes (added as attachments to transactions). This, however, does hamper its reliability as database corruption can become an irrecoverable event if no backups are available.

Deployment. Fabric is built around Docker for node deployment, with solutions using Kubernetes or Docker Swarm being logic extensions for management of extensive networks. If we want to deploy Fabric applications, we must use a Docker environment, which conflicts with our idea of using it alongside Fenix without incurring in some overhead.

Corda uses Gradle [91] as its build system and bundles tasks that can produce Docker images or traditionally packaged Java code in JAR format. It runs on any machine with JVM, either by using its internal Jetty server or another applicational. It can be deployed as any other web-app by placing the Java archives in the correct directory.

3.5 Conclusion

In the end, despite Fabric being a better and more mature framework overall, Corda was chosen as the implementation platform. The decision came down to the strong synergy with the software stack: Corda is built for Java and the JVM, which **qubIT** knows very well (the learning curve is just understanding Corda itself), requiring minimal changes to the existing build system, all leading to faster proof of concept development.

It was also clear that this was not a final choice: with the server infrastructure moving slowly but surely towards using only containers, Fabric would overcome this integration hurdle so this choice could be revisited in the future.

Chapter 4

Academic CorDapp

In this chapter, we start by describing the DevOps work that was done to setup the development environment for Corda. We then detail the design and implementation of the proof of concept, the Academic CorDapp.

4.1 Environment Setup

Before any code was written, we spent some time preparing the development environment. If the proof of concept showed promise, therefore bringing in more developers, it was important to minimize the time spent setting up the Integrated Development Environment (IDE) and installing necessary tools (e.g., build systems). The chosen approach was to mirror what was already being provided to development teams and create a preloaded Docker image. Distribution was done through private a Docker repository, where users could pull the image and start using it immediately.

We chose Visual Studio Code (VSCode) for the default IDE as it has a R3 Corda plugin [92] that allows for quick access to common operations such as building and interacting with nodes. We also added IntelliJ IDEA, used in Corda's training materials, as an alternative. Its Gradle integration is better as a Java/Kotlin specialized IDE instead of a general purpose IDE like VSCode.

Also packed in was Ganache [93], an Ethereum development tool that is now compatible with Corda. It can be paired with either IDE to deploy and interact with a Corda network. Finally, a template project was written for a message sending CorDapp. It covers the main aspects of using Corda, from Vault queries to using external services through Oracles, and provides examples of the coding style used in CorDapps.

With the environment created, work began on the Academic CorDapp. Development took place between December and March, using version 4.3 of open source Corda. We followed a weekly sprint model, with a Kanban board for issue tracking and a meeting every Monday with the rest of the R&D team to present and discuss progress.

4.2 Network Topology

The desired network topology mirrors the current school organization of ULisboa. Each school is represented by a node on the network, with Reitoria in an oversight position and with a single Notary cluster being shared across them.

Figure 4.1 shows our test case consisting of two schools, FCUL and FLUL, Reitoria and a single Notary.

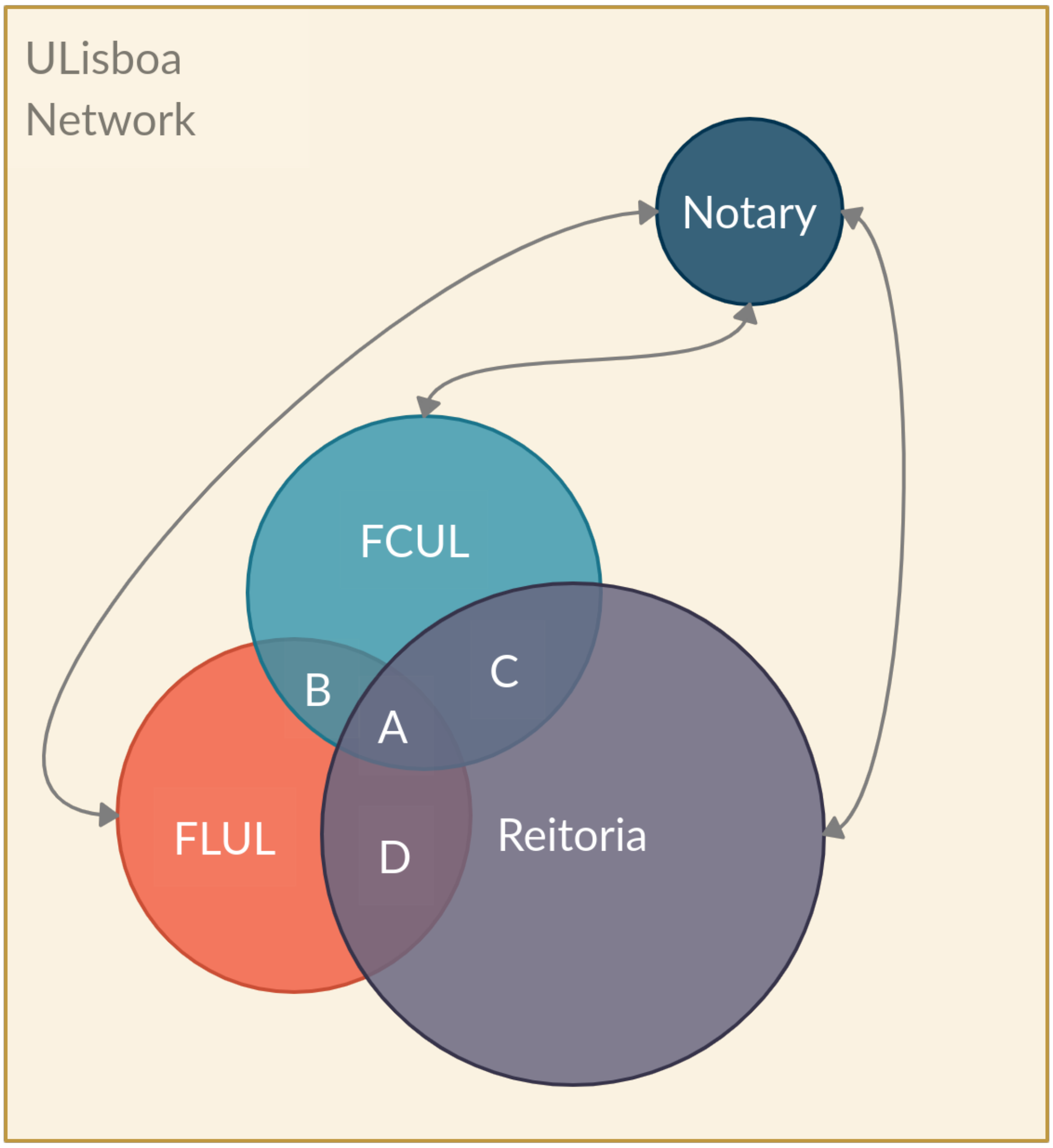

Figure 4.1: Test Network Topology for Academic CorDapp.

Recall that, in Corda, data on the ledger is only visible to the participants in the specific transaction (see Section 3.4.1). In Figure 4.1, each of the letters **A**, **B**, **C** and **D** represents a set of facts shared between those nodes through the ledger.

4.3 Domain

qubIT's solutions are built in Java using domain-driven design (DDD), where objects contain data as well as business logic. Our application uses the concepts present in the academic scope of Fenix's domain. However, it was unfeasible to use that domain directly in a proof of concept as it is too complex, holding more than a thousand different entities. We opted to create a smaller, bespoke domain model, mirroring the most relevant Fenix entities, presented in Figure 4.2.

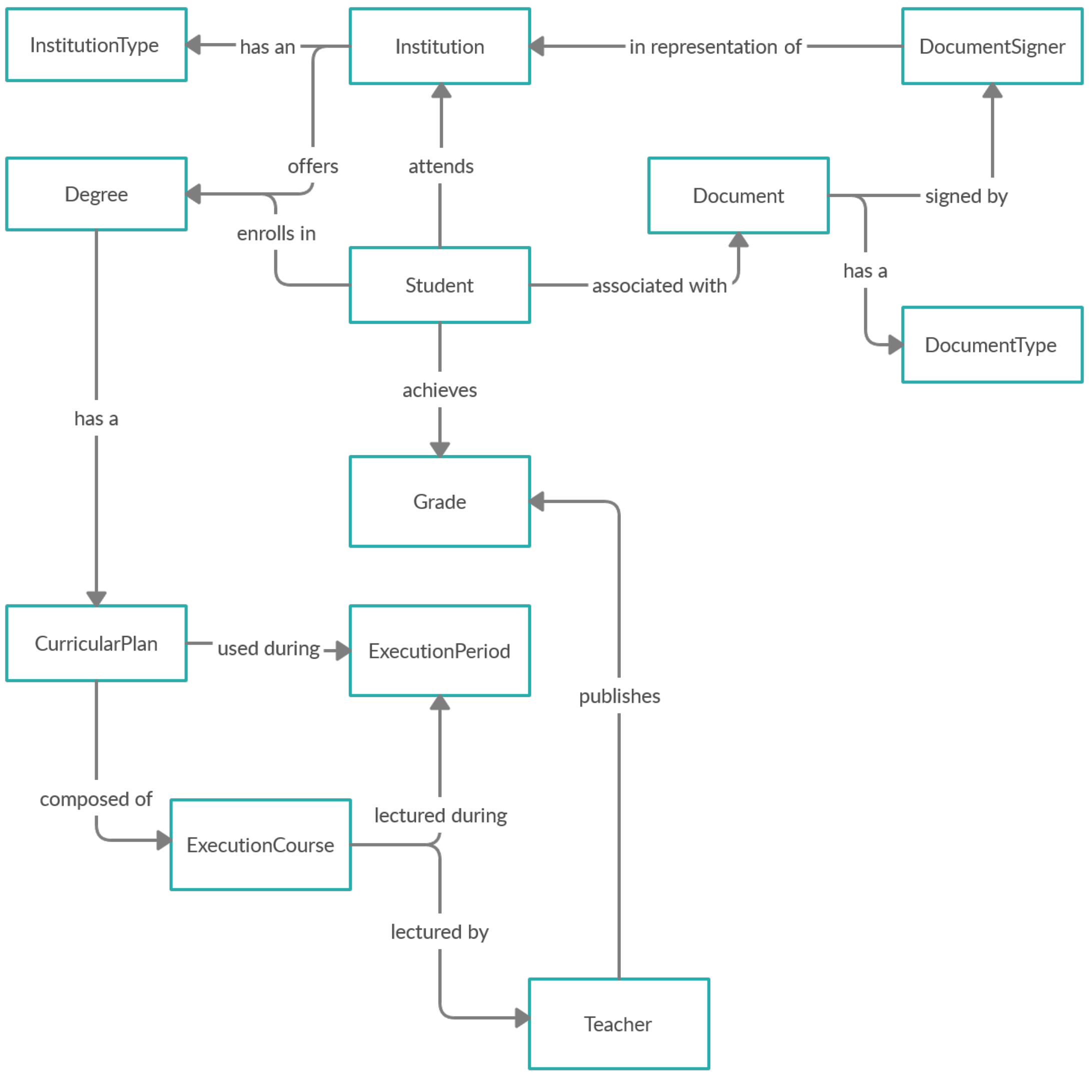

Figure 4.2: Domain model of Academic CorDapp.

We decided against adding attendance as schools have different rules regarding it: attendance records can be used for grading, statistics or simply ignored.

This domain was only used to model the use cases on the Fenix side, as Corda's programming paradigm is not suited for DDD in the style of other existing solutions.

4.4 Use Cases

Within Fenix, our users will be whomever is working in the academic services, handling the student's administrative processes, and teachers, through grade publication. In both cases, interaction should remain identical to present form: we mirrored the necessary classes in our domain model for easier conversion and all extension points are managed behind the scenes through a bridging module the connects Fenix to the Corda network. It is important that adding this functionality does not disrupt any existing workflows. Figure 4.3 presents the use case diagram for the application.

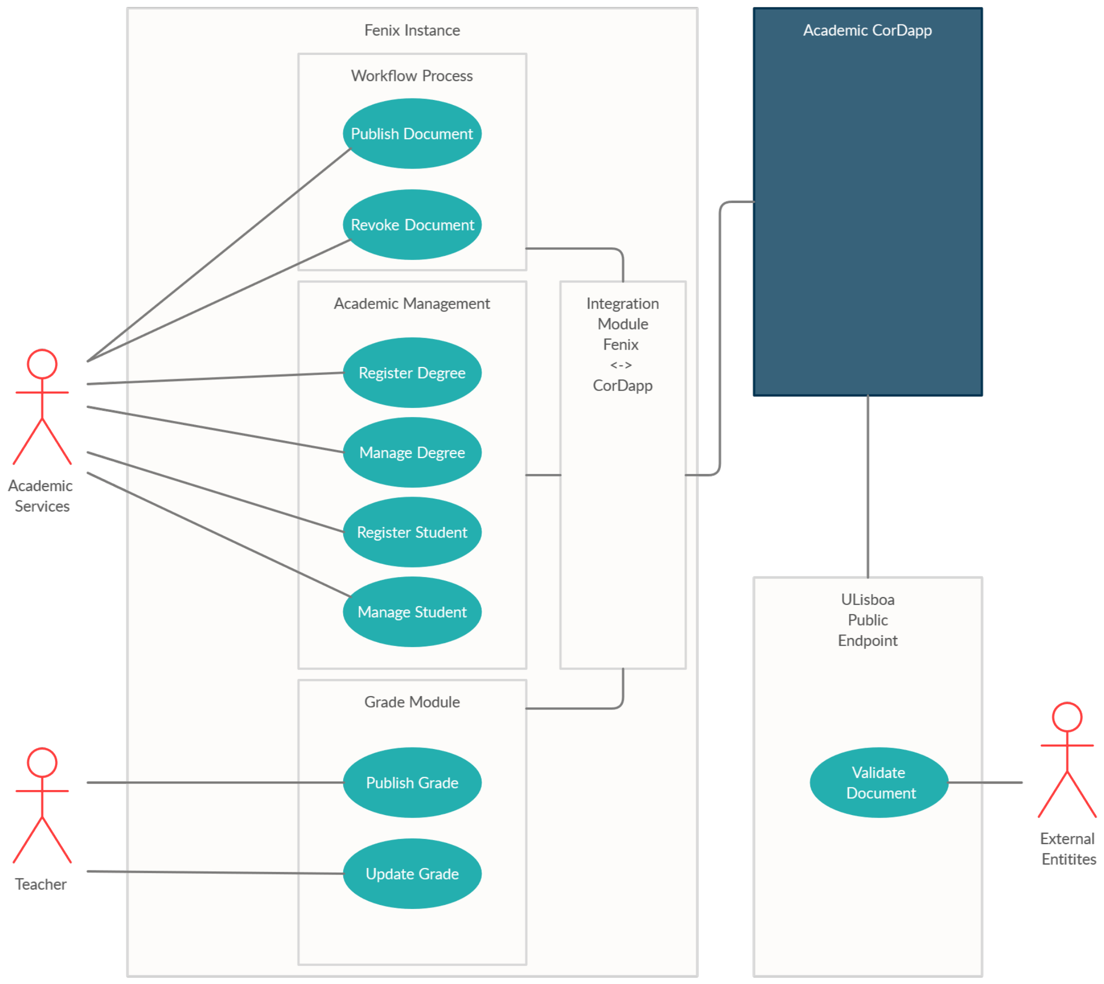

Figure 4.3: Use Case Diagram of the Academic CorDapp.

The implementation of the public endpoint is yet to be determined. It could be created as an autonomous system or integrated, through a Fenix module, into Reitoria's Fenix instance.

With the domain and use cases established, it was necessary to convert them into states and flows used by Corda.

4.5 Entities

Recalling Chapter 3, Corda's transactions follow the UXTO model: every state is immutable and business logic is implemented through flows, effectively separating data from code. This makes directly using the DDD approach unfeasible as Corda resembles more a service-oriented architecture.

Corda's documentation asserts that states can contain arbitrary data, so they could, in theory, contain other states and arbitrary objects, like internal classes stored in a serialized form. Nonetheless, trying to avoid serialization and its associated issues pushed us to use simpler types, since a state reference may be outdated (or consumed) when materialized from its serialized form. We can still map information from the Fenix system to Corda states without adding serialization complexity through object identifiers. Domain relationships are established between the states using the input, output and reference states of each flow.

From that domain model in Figure 4.2, we extracted four key entities: Student, Grade, Degree and Certificate (our chosen document type), shown in Figure 4.4. These states will represent the information stored on the ledger and were chosen due to being the elements where more update operations are expected to happen. The rest of the information will be provided by Fenix when necessary.

Using a smaller number of states allows us to keep flows simple and straightforward and reduce the size of each transactions in terms of associated data (see Section 4.6).

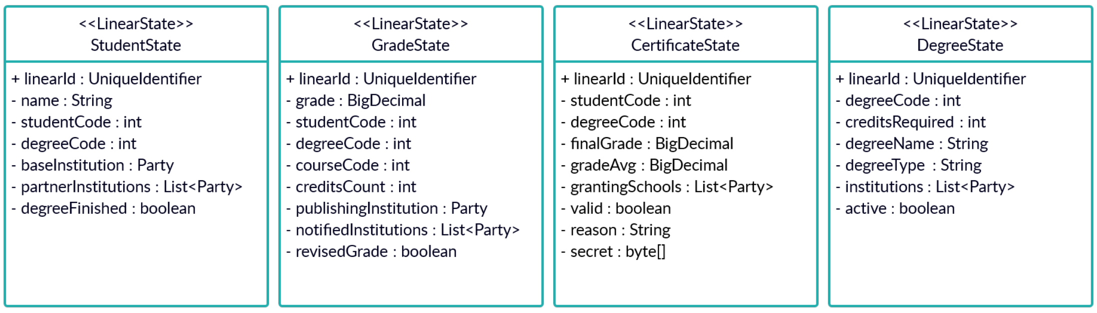

Figure 4.4: States corresponding to the key entities with respective fields.

Each school is represented by a node on the network, hence having an associated **`Party`** (the code representation of a network address) to represent them. For example, a student registered to FCUL would have the **`baseInstitution`** field with FCUL's node identity.

Since we want keep a continuous track of activities, each state is created with an unique ID that is maintained across transactions and propagated to states that originate from it. Querying with that ID allows us to find every related state and transaction up to the root.

Our convention is that every integer variable with the *-Code* suffix in its name stores the external id of the corresponding object in the Fenix instance, pairing it with the Fenix domain. This allows for the bridging module to look up domain objects and materialize them when synchronizing information.

4.6 Contracts

For a transaction to be accepted, its inputs and outputs must comply with the contract that regulates them. Each state we created is annotated with **`@BelongsToContract`**, pointing to its contract class. Contracts in Corda perform invariant checking and run in a sandbox to ensure that no outside information sources are used. This ensures that a contract is consistent: a call to its **`verify`** method should always fail or succeed for a given transaction, regardless of the world state.

Since this validation can be called upon at any time, contracts are bundled with the transaction they validate as additional data. We are encouraged to keep them lightweight or the ledger would grow in size too quickly.

A common mistake is to use them to validate business logic, e.g., checking if a **`CertificateState`** meets a required number of ECTS, with that value being obtained from a dynamic source. That kind of validation should occur in the flow as a change in that value would break past transactions.

If for some reason requirement changes in the future, Corda offers an internal mechanism to upgrade contracts, the **`ContractUpgradeFlow`**, re-bundling the “old” states with the new version of the contract. This is only possible if the new contract is still compatible.

To speed up invariant verification, Corda offers a Domain Specific Language (DSL) [94] to model the required assertions, with accompanying error messages that are captured and shown to the user. Having access to this, we decided that, for now, data checks would be performed on Corda instead of using Fenix. An example of DSL usage is presented in Figure 4.5, where a **`CertificateState`** is validated.

```
@Override
public void validate(CertificateState state) {
    requireThat(require -> {
        require.using(i18n("Student code must be a positive
         ↪  integer."), state.getStudentCode() > 0);
        require.using(i18n("Degree code must be a positive
         ↪  integer."), state.getDegreeCode() > 0);
        require.using(i18n("Final grade must be greater or
         ↪  equal to 10."),
         ↪  state.getFinalGrade().compareTo(PASSING_GRADE)
         ↪  >= 0);
        require.using(i18n("Grade average must be greater
         ↪  or equal to 9.5."), state.getGradeAvg()
        .compareTo(PASSING_GRADE_AVG) >= 0);
        require.using(i18n("There must be at least one
         ↪  Degree granting Institution."),
         ↪  state.getGrantingSchools().size() >= 1);
        return null;
        });
    }
```

Figure 4.5: **`CertificateContract`** code sample.

The validations applied were the same as those currently in use for those Fenix processes (e.g., validate grade values according to grade scales) as well as some sanity checks.

4.7 Flows

Flows are how we interact with states and, without a delete operation, we reduced operations to a pattern that can be described as create, update and consume.

Create and update are self-explanatory. Data added to the ledger cannot be deleted so it is important that it is thoroughly checked and can be updated if necessary. We define consume as a terminal operation: an academic process is finished when all of its states are consumed. This "locks in" the information and makes it so they can only be used as reference states (e.g., a student re-registers for a new degree and we can reference the previous one).

Table 4.1 describes the flows associated with these operations. They are semantically named for code readability but functionally identical.

Entity	Create	Update	Consume
Degree	RegisterDegreeFlow	OpenDegreeFlow CloseDegreeFlow	-
Certificate	IssueCertificateFlow	-	RevokeCertificateFlow
Grade	PublishGradeFlow	UpdateGradeFlow	-
Student	RegisterStudentFlow	UpdateStudentFlow	ConcludeStudentFlow

Table 4.1: Implemented flows.

Aside from our flows, there is also a group of core flows, provided by Corda [95], that perform common tasks such as collecting signatures, signing, receiving or sending transactions. The most important ones are **`FinalityFlow`** and **`ReceiveFinalityFlow`**, which send and receive transactions to the Notary.

As mentioned in Section 4.5, the domain relations between entities are established through what are the inputs, outputs and reference states of the flows. Table 4.2 presents those relations with their associated cardinality.

Flow	Inputs	Outputs	References
RegisterDegree	-	DegreeState (1)	-
OpenDegree	DegreeState (1)	DegreeState (1)	-
CloseDegree	DegreeState (1)	DegreeState (1)	-
RegisterStudent	-	StudentState (1)	DegreeState (1)
UpdateStudent	StudentState (1)	StudentState (1)	-
ConcludeStudent	StudentState (1)	-	CertificateState (1)
PublishGrade	-	GradeState (1)	DegreeState (1), StudentState (1)
UpdateGrade	GradeState (1)	GradeState (1)	-
IssueCertificate	GradeState (*)	CertificateState (1)	DegreeState (1), StudentState (1)
RevokeCertificate	CertificateState (1)	CertificateState (1)	-

Table 4.2: Flow inputs, outputs and references (from Figure 4.4).

To determine who needs to sign a transaction for it to be valid, Corda implements commands, indicating what action is to be taken and who are the required signers. Using commands would allow us to reduce those ten flows into four flows that supported different logic according to the command present but we eschewed this idea as having a one-to-one matching between flows and commands allowed for less branching of flow logic (no need to check the command at multiple points to dictate behavior). Also, Corda's access control for RPC is flow based, not command based (e.g., you can limit what flows a RPC user can initiate). If we wanted to restrict the use of a certain command by a party, we would have to code that restriction, probably having to access an external service to validate if the restriction was still valid, instead of getting from the node configuration that can be changed by a quick reboot.

The decision to consume or output a state is regulated by business logic. When we want to lock information, we opt to consume the states. The update operation, like in the **`UpdateGradeFlow`**, consumes and produces a state of the same type, enforcing the business logic that a grade can only be updated once by having the flow check the **`revisedGrade`** property instead of checking if the state is consumed.

To better explain flows and their effect on the system, we will explore the "happy path" sequence considering a student attending and graduating a degree shared between our two test schools.

First, one of the schools registers the degree using the **`RegisterDegreeFlow`**, identifying the other participating schools (see Figure 4.4 for field references). Figure 4.6 illustrates the flow steps.

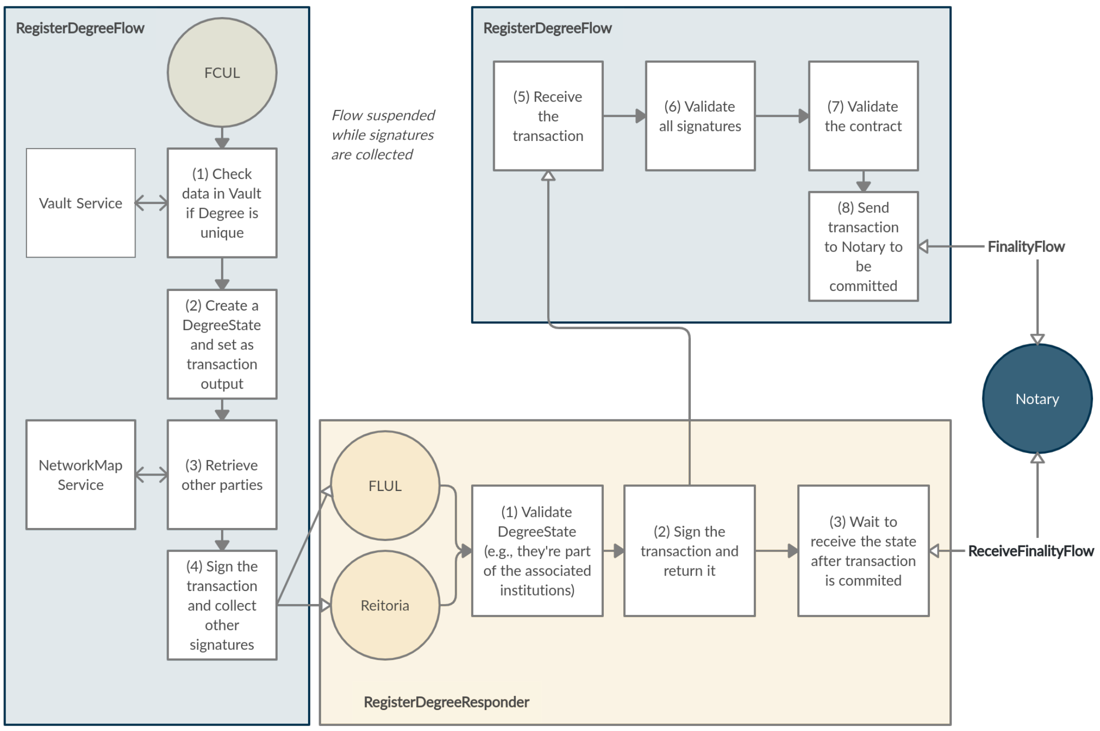
RegisterDegreeFlow
FCUL
Vault Service
(1) Check data in Vault if Degree is unique
(2) Create a DegreeState and set as transaction output
NetworkMap Service
(3) Retrieve other parties
(4) Sign the transaction and collect other signatures
Flow suspended while signatures are collected
RegisterDegreeFlow
(5) Receive the transaction
(6) Validate all signatures
(7) Validate the contract
(8) Send transaction to Notary to be committed
FinalityFlow
Notary
ReceiveFinalityFlow
FLUL
Reitoria
(1) Validate DegreeState (e.g., they're part of the associated institutions)
(2) Sign the transaction and return it
(3) Wait to receive the state after transaction is commited
RegisterDegreeResponder

Figure 4.6: **`RegisterDegreeFlow`** step-by-step diagram.

After that flow is concluded and the transaction is notarized, each participant (FCUL, FLUL and Reitoria) will have a **`DegreeState`** stored in their Vault as well as the transaction that created it in their transaction storage.

A student then chooses a base institution and is registered as attending that degree through **`RegisterStudentFlow`**. This flow follows the same structure as the one outlined in Figure 4.6, with the difference being that it validates that the student is unique and the degree exists, and produces the same type of outcome in the participants' vaults: each receives the new **`StudentState`** as the transaction output.

As she/he attends classes, grades are published by teachers in Fenix and pushed to the ledger using the **`PublishGradeFlow`**. This flow works in a different manner from the previous ones. Publishing a grade is an autonomous act, performed by a teacher and requiring no extra authorization. After the publishing party has validated the necessary information, no other signature is necessary and the transaction is only sent to the other parties for storage. Figure 4.7 shows the steps taken during this operation.

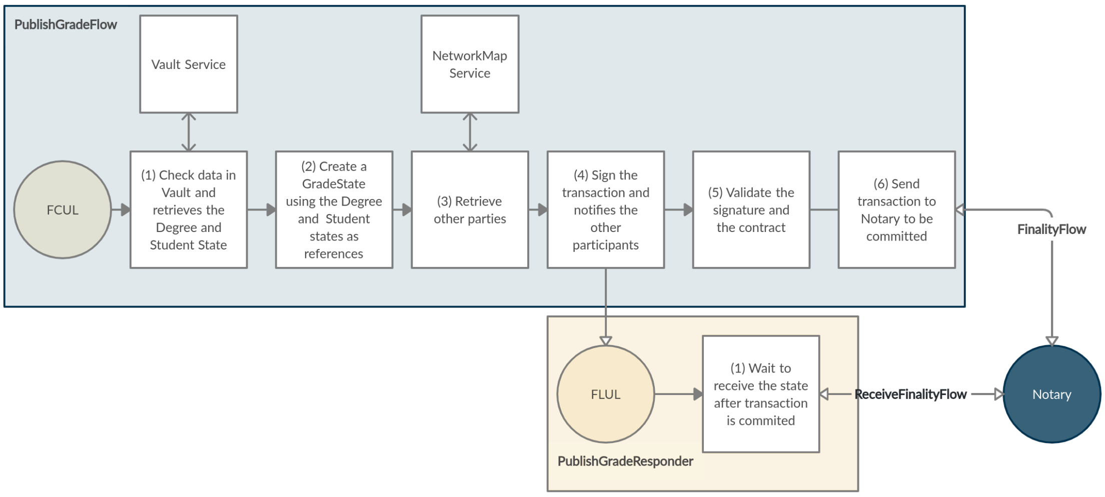
PublishGradeFlow
Vault Service
NetworkMap Service
FCUL
(1) Check data in Vault and retrieves the Degree and Student State
(2) Create a GradeState using the Degree and Student states as references
(3) Retrieve other parties
(4) Sign the transaction and notifies the other participants
(5) Validate the signature and the contract
(6) Send transaction to Notary to be committed
FinalityFlow
FLUL
(1) Wait to receive the state after transaction is commited
ReceiveFinalityFlow
Notary
PublishGradeResponder

Figure 4.7: **`PublishGradeFlow`** step-by-step diagram.

In this flow, reference states are used to add information about the student and the degree at that point to the transaction without being consumed. This allows us, in a way, to store state references in a state, something we excluded in Section 4.5 in favor of simple types, as reference states are stored with the transaction in the ledger. These reference states are used to capture the information present on the **`StudentState`** (e.g., base institution) and **`DegreeState`** when the grade is published.

Finally, when the degree requirements are met, a certificate is issued through the **`IssueCertificateFlow`** and the student concludes his academic path in the school with the **`ConcludeStudentFlow`**. These flows are interconnected, with the first output being a required reference for the second.

Figure 4.8 presents the steps in the **`IssueCertificateFlow`**. Reitoria is hard-coded into the critical path of this flow (there is a specific call to a party with that specific name) and this is our only use of an Oracle. Recalling Section 3.4.3, Corda requires determinism in execution. An Oracle is how Corda describes a service that will only sign a transaction if an associated fact is true [96] and can validate that fact using a non-deterministic external source. In the test case, we check a REST endpoint as a mock authorization to decide if the certificate creation is authorized. Following the Fenix integration, this will be an user action through a workflow process.

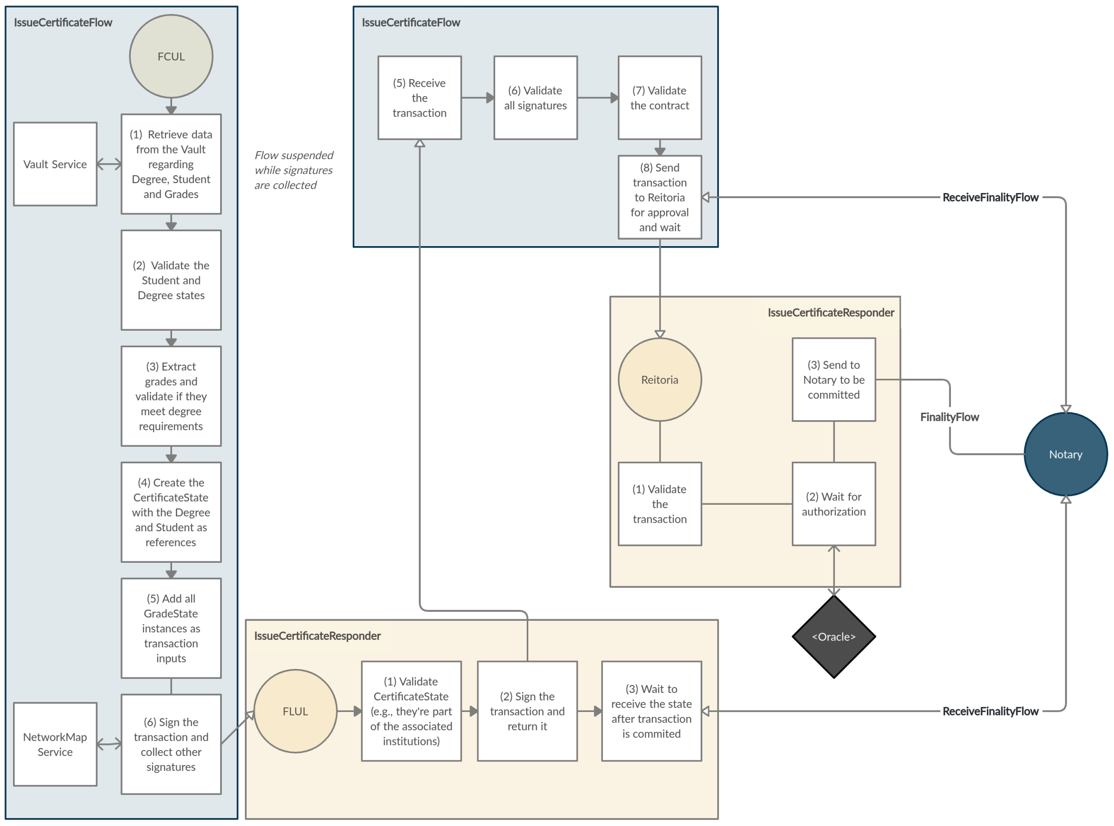
IssueCertificateFlow
FCUL
Vault Service
(1) Retrieve data from the Vault regarding Degree, Student and Grades
(2) Validate the Student and Degree states
(3) Extract grades and validate if they meet degree requirements
(4) Create the CertificateState with the Degree and Student as references
(5) Add all GradeState instances as transaction inputs
NetworkMap Service
(6) Sign the transaction and collect other signatures
Flow suspended while signatures are collected
IssueCertificateFlow
(5) Receive the transaction
(6) Validate all signatures
(7) Validate the contract
(8) Send transaction to Reitoria for approval and wait
ReceiveFinalityFlow
IssueCertificateResponder
Reitoria
(3) Send to Notary to be committed
FinalityFlow
Notary
(1) Validate the transaction
(2) Wait for authorization
<Oracle>
IssueCertificateResponder
FLUL
(1) Validate CertificateState (e.g., they're part of the associated institutions)
(2) Sign the transaction and return it
(3) Wait to receive the state after transaction is commited
ReceiveFinalityFlow

Figure 4.8: **`IssueCertificateFlow`** step-by-step diagram.

After the certificate is emitted, it is used as a reference to the **ConcludeStudent-Flow**, which takes the current student, associates the **CertificateState** and outputs a new **StudentState** with **degreeFinished** set as true.

The document validation process is built from this final state and corresponding transaction chain. That chain will have a reference to every grade that student earned (and associated transaction history), the transaction history of the degree he was attending and his academic status throughout the process. This information can be placed onto a timeline, detailing his progress through the institution and establishing transparency regarding his grades and attendance.

The ability to revoke a certificate was a requirement, as the document validation has little value if we cannot rescind it. As stated above, emitting a certificate consumes every grade and this operation is terminal. When revoking a certificate, we cannot bring back those grades, they must be reissued. Since this is a critical operation, we secured it in two ways. First, the **RevokeCertificateFlow** can only be invoked by Reitoria, being locked through access control and through code validations. Second, one of the attributes of the **CertificateState** is a secret, provided during the flow invocation through the command line (after integration, it will be generated by Fenix) and that is visible in the Vault (in this case, we are using a SHA-256 hash). To revoke a certificate, a reason must be stated (e.g., academic fraud) and a value must be provided that, when hashed, matches the stored secret, authenticating the request in a mechanism akin to Leslie Lamport's S/KEY [25].

4.8 Participant Management

As this application would be offered as a product, it was unlikely that every school would be using it from the start. An issue that quickly came to the forefront was how would we deal with the addition and removal of participants.

Adding a new Party to the network is simple enough: create the necessary key material, copy the folder structure from an existing template node (since they would use the same versions), deploy it on a machine and the node is online. Using containers, this process is even simpler: we mount the key material folder on a prepared Docker image. A new peer can be added in mere minutes but it starts in a *tabula rasa* state.

As shown in Figure 4.6, Figure 4.7 and Figure 4.8, only the flow participants receive the state in their Vaults. Our expected scenario is the following: **A** and **B** are parties that already have some state shared between them. Now, party **C** joins the network at a later date and needs to have access to the state stored by **A** and **B**. With its own Vault empty, it cannot start a flow using those states even if somehow aware of their existence in the counterparts' Vaults.

Corda validates transaction inputs by a process they call "walking the chain" [97].

This means that every input state's transaction history is validated recursively from the present moment to its creation. Whenever a node does not have that information in its Vault (e.g., it has no previous transaction involving that state), that missing information can be requested from the transaction proposer [98].

One way to solve this is already partially implemented by the chosen topology. Reitoria occupies an oversight position and is privy to every state that is supposed to have external visibility (e.g., grades are excluded). Instead of adding Reitoria explicitly to every flow call, we can implement the "Observable State" pattern. This makes the calls to **`FinalityFlow`** and **`ReceiveFinalityFlow`** store all visible states instead of only the relevant ones (default behaviour where you receive the states that you are a participant in). Code changes would be minimal, as exemplified in Figure 4.9.

```
@Suspendable
public SignedTransaction call() throws FlowException {
    //return subFlow(new ReceiveFinalityFlow(flowSession));
    return subFlow(new ReceiveFinalityFlow(flowSession,null,
    StatesToRecord.ALL_VISIBLE));
}
```

Figure 4.9: "Observable State" pattern implementation.

While this party could then offer a service to request states in bulk, we did not want to possibly overload a single node with this, especially since the flow state machine is single threaded (see Section 5.2) and requesting a large number of states would make it unavailable for other purposes.

It made more sense to create a flow that would query other Vaults and, if authorized, request relevant states to add to its own Vault. This operation would have to be in bulk, as the new participant may not be aware of all the states that it can query for.

So we created a "catch-up" flow, named **`UpdatePartyFlow`**, that is ran between existing parties, with privacy parameters to filter states, and recreate them with the new party as a participant. This flow can only be started by Reitoria (enforced through RPC access controls), which also decides what kind of states should be shared and between whom. Each party has a **`ProcessUpdatePartyFlow`** that performs no further verification so the transactions runs as quickly as possible. This rewrites that portion of the ledger and brings the new party up to date, with new versions of the unconsumed states in its Vault.

For the opposite case, when schools, who have some administrative independence, decide to leave. Our only concern if that school holds the only copy of an unconsumed state. For this purpose, we made **`UpdatePartyFlow`** receive a parameter that marks if you want to add or remove a Party from the participant list, effectively "passing on" the responsibility over those states and retaining no usable data. Figure 4.10 shows the steps in this flow.

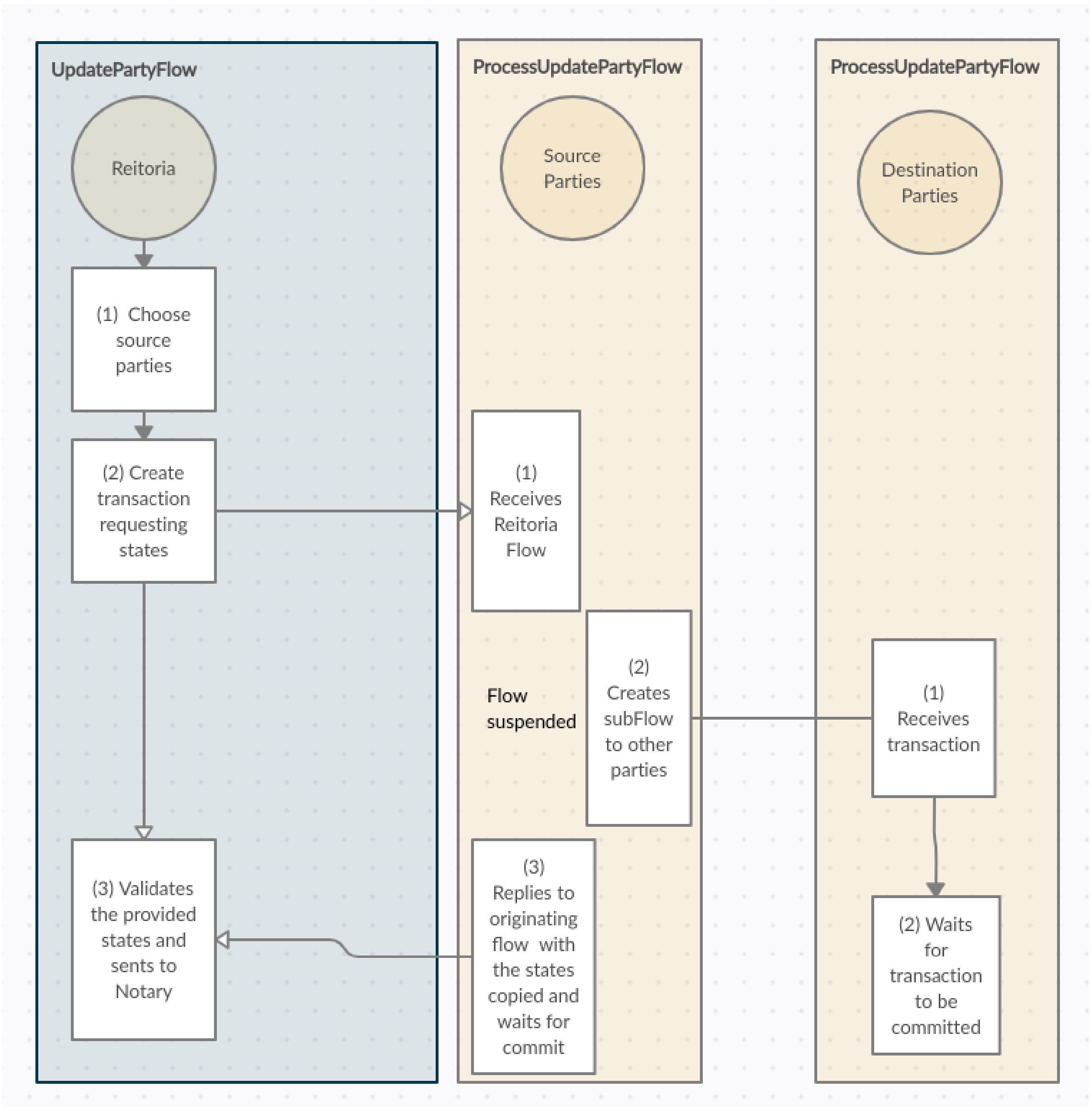
UpdatePartyFlow
ProcessUpdatePartyFlow
ProcessUpdatePartyFlow
Reitoria
Source Parties
Destination Parties
(1) Choose source parties
(2) Create transaction requesting states
(1) Receives Reitoria Flow
Flow suspended
(2) Creates subFlow to other parties
(1) Receives transaction
(3) Validates the provided states and sents to Notary
(3) Replies to originating flow with the states copied and waits for commit
(2) Waits for transaction to be committed

Figure 4.10: **`UpdatePartyFlow`** diagram.

When the transaction commits, every participant will have an updated state in its Vault with the new participants.

4.9 Notary

Since the Notary has the final word in validating a transaction, we explored implementing our own notary service. The idea was that, if the pricing model was per operation, we could have the service check for authorization before signing a transaction. The Notary would then become a licensing point within the network.

We tried to create our own notary service following the tutorial in the documentation [99], but were unsuccessful. Implementing a custom Notary requires extending an abstract class, **`NotaryService`**, which is not available in the libraries or documentation [100], making this task impossible.

Since we could not implement our own and had no use for multiple notaries as all States must be under the compatibility zone of a single Notary before transactions can proceed (essentially under its control), we used a single Notary node in the default configuration, as shown in Figure 4.1.

Corda supports using notary clusters to provide higher availability and lower latency [101] but, since our transactions must pass through Reitoria, we saw no advantage in having more than one node.

4.10 Integration

As shown in Figure 4.3, integration with Fenix would be provided through a bridging module. This module is planed but not yet developed during the proof of concept.

Flows can be annotated with **`StartableByRPC`** and **`StartableByService`**, with the later meaning it can be initiated by an internal service such as an Oracle. We annotated most flows with both, intending to use the configuration access controls to limit which schools can access which flows.

This module would interact with the **`CordaRPCClient`** and perform conversion from states to domain objects. One thing we would gain by bundling the Corda node into the same machine as the Fenix instance is that the RPC call could be done locally.

4.11 Test Deployment

With the CorDapp built, we deployed a small network for testing. Configuring a Corda node is a matter of replicating the directory structure and putting the correct contents in the corresponding folder. Corda provides several Gradle plugins that take care of that and produce the directory structure ready to be copied.

Gradle, at first, looked like a possible issue as the current stack's build system of choice is Maven and we make extensive use of custom maven plugins. Not wanting to add another build system to the continuous integration cycle, we evaluated the cost of converting the Gradle task that generates the nodes into a Maven plugin. That was quickly abandoned due to time constraints as all the Groovy code had to be converted, as well as the dependency and exclusion system. This was solved by creating a simple Maven task that calls a Gradle wrapper bundled with the project.

Since we want to possibly "piggyback" the Corda nodes on the Fenix servers, we did not use the Docker version of the task. In the future, when we move to a fully containerized architecture, it would be a matter of adding a new service and corresponding Docker image in orchestration to replicate the same behavior.

We successfully deployed a local version of the application running with a validating Notary, two schools and Reitoria as parties and, after some debugging, we were able to complete a full cycle of student's registration, grade publishing and certificate emission. Interaction was done by command line or using the Corda plugin in VS Code, as we do not have an user interface (that will provided through Fenix). One thing that was noticed during debugging is that if the debug flag is set, method calls are not sent over to the other local nodes: they run locally and then communicate. This is an important detail when diagnosing if a flow is broken on a receiving end.

4.12 Final Remarks

After 12 sprints and 248 Kanban entries, the CorDapp was finished and ready to be evaluated by the Business Solutions team. Our proof of concept is unabashedly a simple application, nonetheless "simple is best" when exploring a new concept. In addition to the Kanban issues' description and comments, a document was produced (which later was refined into Chapter 5) detailing the issues found while developing with Corda.

Chapter 5

Analysis

This chapter describes the main challenges faced during the development of the Academic CorDapp, followed by the conclusions reached after discussions with the Business Solutions team.

At the time of writing, Corda has published version 4.6 (both Enterprise and open source), with several fixes and improvements.

Some of limitations presented here pertain specifically to the open source version of Corda and are not present in the Enterprise version. Even though a trial was available, we decided not to request it as to not tinge our viability analysis with the premium features.

5.1 Database Management

5.1.1 Production Databases

When a node starts, it runs an H2 embedded database [102] as its Vault storage. This is fine for development and testing as H2 is lightweight and requires no configuration from the user (the database schema is inferred). Even so, this database offers no commercial support [103] so we were against using it in a production system.

Corda supports PostgreSQL and SQL Server but only through community additions. In an official capacity, it does not support anything other than H2. Production systems currently use MariaDB [104], a robust and well tested database. To share a database with Corda, we would have to create a Java Database Connectivity (JBDC) connector that could work with it and to retain the "out-of-the-box" experience of using H2 with no additional configuration, we would also have to deploy with the database schema fully configured.

Developing and supporting a connector was out of the question so we kept this issue open while moving forward with the H2 database. It was marked as "Critical" and solving it was a *sine qua non* condition for deployment. We opted to wait and see if the next Corda open source version would add better database support.

5.1.2 Data Integrity

In Chapter 3 we established that Corda is not a classical blockchain as there is no shared world state. Since each node stores data in its Vault, data corruption is an important issue to consider.

Recalling Section 4.8, Corda validates transaction inputs by a process they call "walking the chain" [97]. This means that every input state's transaction history is validated recursively from the present moment to its creation. Whenever a node does not have that information in its Vault (e.g., it has no previous transaction involving that state), that missing information can be requested from the transaction proposer [98]. If a proposer has the only complete record of transactions for a group of assets and that information is corrupted, those assets are rendered useless.

As there is no global state from where to reconstruct a corrupted Vault, this situation must be handled through database backups, which ties in with the concerns in Section 5.1.1. We have robust mechanisms to ensure backups and restores are done properly, but all tailored for MariaDB. Lacking backups, it is not clear how a Corda node can recover from Vault corruption, either total or partial.

Recently released Corda 4.5 Enterprise now offers collaborative recovery to partially address these concerns. Nodes can query other peers for missing ledger data and retrieve it, allowing a node to recover from partial corruption provided another node holds a valid version of the corrupted state in its Vault. This mechanism is a more robust version of the flow we implemented to update new parties on the network (see Section 4.8).

5.1.3 Queries

Performing a database query is a common procedure for software applications. In Corda, queries are placed to a node's Vault using the Vault Query API and are limited to the columns of Vault related tables (e.g., **`VAULT_STATES`**), containing state's meta-information.

Only having access to the Vault related tables means that, in practical terms, we can query for all instances of **`StudentState`** that are unconsumed (a Vault attribute) but not for instances where the name starts with a certain letter.

To perform arbitrary queries, states need to implement the **`QueryableState`** interface and provide another Java class strictly for Java Persistence API (JPA) mapping. In Corda, it is not possible to use annotations directly on the state, which leads to duplicated code.

Since the pattern of retrieving every instance and filtering later is a common pattern in Fenix (all data is already loaded into memory), we chose not to create the additional mapping classes.

5.2 Code Validation

As mentioned in Chapter 3, using general purpose languages instead of a DSL opens up room for errors.

The open source version of Corda handles flows using a single threaded state machine (Enterprise version has a multi-threaded implementation), shown in Figure 5.1. Each flow starts in the "Running" state.

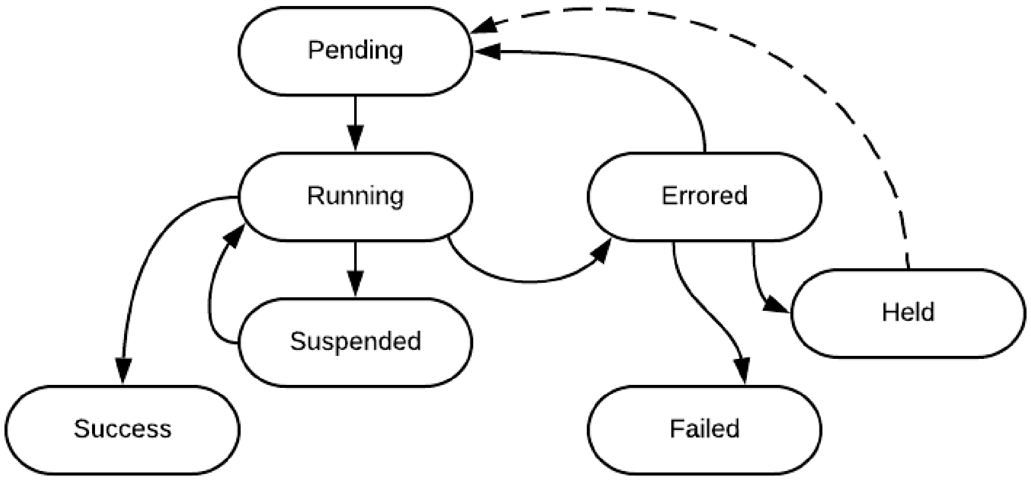

Figure 5.1: Diagram of internal flow handling state machine [2].

It is important to be aware of its single threaded nature to avoid creating a long running flow that blocks transaction progress. Also important is to correctly track the steps within a flow since an incorrect transition may place the flow handling in a state from where it cannot recover.

When calling flows between parties, like nodes in a graph, every transition is directed. A flow determines its origin and destination through annotations, **`@InitiatingFlow`** for the initial call and **`@InitiatedBy`** for the matching flow on the recipient side. That flow is tasked with receiving and processing the message, triggering the next state transition until "Success" is reached. Inside each call, other flows can be called, using **`subFlow()`**. That places the outer flow in the "Suspended" state, awaiting the conclusion of the inner flow call. And herein lies the issue.

Without tools for code validation, it was easy to make a mistake and call a receiver flow (annotated with **`@InitiatedBy`**) out of order or create a recursive flow call that placed the flow state machine in an irrecoverable state. We referred to this as "sequence breaking". In the case of the receiving flow, that flow thread would be suspended, waiting to reply with an unrequested message and the node will not recover by itself. To aggravate this, Corda did not offer a timeout mechanism that could be used to force flows to fail, as transactions should not be lost if nodes leave and rejoin the network due to temporary failures.

Since each annotation indicates the class where the flow code must be, it would be good if Corda added annotations for flow validation, be it through a specific module or Gradle task, that checks if each initiating flow has a corresponding flow that can receive that message and reply to it. Another annotation used in flows, **`@Suspendable`** - used to annotate methods so the state machine can be placed in the "Suspended" state, achieves this by throwing an error whenever non-annotated methods are called, something easily detected when testing code. We considered implementing this by adapting one of our custom Maven plugins, but the effort of converting it to Gradle was considered too high.

We also explored creating our own annotations to help with other parts of the code validation process but, in the end, it was still up to the user to annotate correctly.

5.3 Performance

The overall throughput of the test network was not evaluated as it was running in an optimal environment with local networking and ample resources. We did, however, profile the code while debugging issues with the flows, to determine if it was running slowly or had crashed.

One performance finding is related to Section 5.1.3. When creating a large number of students and corresponding grades for each, there was a small but progressively noticeable slowdown as the **`RegisterStudentFlow`** needed to query every unconsumed Vault state to ascertain that the student information is unique and valid.

This is a common code pattern in Fenix (due to how the FenixFramework [105] ensures consistency) but we could not apply the usual mitigation strategy of using caches. It was not clear how we could ensure that the cache data was still valid (e.g., the conclusion of a suspended flow could consume or create new states) without incurring in the costly operation that we were trying to avoid. In the end, the proposed solution was to shift these checks to Fenix, where caching is available, and remove the more costly operations from the final code.

We did not run into any issues but we could foresee that another possible bottleneck was **`@Suspendable`**. This annotation is used in methods called during flows to indicate that the Quasar library [106] should instrument their byte code so execution can be suspended and resumed. When suspending, all information is serialized to disk for persistence, with the associated I/O cost. This is negligible for smaller flows but each **`subFlow`** called increases that amount of information that must be written from the call stack.

Finally, the lack of asynchronous calls, coupled with the single threaded state machine (shown in Figure 5.1), meant that overall flow performance depended on network latency more than anything.

5.4 Fenix Integration

While contracts are meant to be deterministic, flows can access outside data sources. Flows expect the network to be the only source of delay, mirroring a smart contract in executing automatically, without any user intervention. But since external queries have to be performed sometimes and user intervention is required to proceed with certain actions, we end up blocking a node for calls that should be asynchronous.

An example is the creation of the degree certificate: the flow can check if every academic requirement is met, but only a user, acting in representation of the school, can authorize the emission of the degree certificate document. If a mistake is made and it triggers a flow call, we can recover it in Fenix but will be locked in waiting on the Corda side, a situation similar to the one described in Section 5.2.

Corda 4.4 introduced asynchronous flow calls but it is still an experimental feature. A StackOverflow answer by one of the developers states that Vault operations may still be synchronous, which could block asynchronous calls that depend on it.[1]

Regarding integration by sharing the same machine, Corda's deterministic JVM implementation was in beta before being dropped completely with the release of version 4.5 in favor of a JDK module.

That is an important difference, allowing a single JDK to be used in the production server instead of having to worry about supporting multiple execution environments, each with its own configuration. However, that module is built for Java 8, which we have already migrated from to Java 11. This choice of an end-of-life version makes it clear that the Corda development team is favoring Kotlin much like Fabric prefers Go.

5.5 Business Analysis

After the Academic CorDapp was finalized, a meeting was held with the Business Solutions team to ascertain its viability.

At this point, the academic transparency aspect for external institutions was still under appreciation due to GDPR concerns, so it would only be used within ULisboa. There was also doubt if we could get the institution on board for further development (this blockchain project was out of the scope of the current agreement). The secondary objective was then brought to the forefront as they intended to compare Corda with the other solutions already in consideration for data sharing between instances (see Section 3.2) and decide if it met the criteria for further investment, effectively re-purposing the project. This decision hinged on three questions:

- *Does it perform better than the other options?*

- *Does it provide more value for the costumer than what we currently have available?*

[1]https://stackoverflow.com/questions/49085980/

- *Does it allow us to meet requirements in an easier way?*

Everyone had full access to the CorDapp, including the source code, and documentation detailing the findings present in the previous sections.

5.5.1 Performance Review

To answer the first question, the team reviewed the findings in Section 5.3 and considered them an acceptable trade-off, as the cause of the issues is intrinsically built into Corda and cannot be separated from its use.

Our use case has a low volume of concurrent transactions as most academic documents are produced in a staggered fashion. Grades are rarely published in large batches and only a subset of them would be placed on the ledger. Even using a single Notary node, this would not be an issue. According to Corda's own benchmarks [107], the Enterprise edition can achieve over 255 flows per second in its default configuration.[2]

Since the predicted throughput requirements are low, the performance is acceptable as none of the operations need to be responsive and data can be stored in each Fenix instance and synchronized to the ledger later. In spite of this, the lack of support for more robust database offerings was a sticking point, for which there was no good answer, even when using the Enterprise edition.

5.5.2 Value Proposition

In Section 3.3, we describe why we did not pursue Ethereum. In this meeting, we returned to an argument made during that choice: if a value placed on a blockchain does not have legal standing, that trust could be built up somewhere else and complemented by the information stored in the public blockchain. For the sake of such argument, we excluded the gas payment for transactions. The idea in discussion was that if it was even possible to get a certificate, signed by a trusted authority, that ULisboa controlled an Ethereum address and leverage that in the document validation process.

Even if academic documents could be validated that way, they would, by law, need to digitally signed, where the LTV standard is used. Since we can append additional data to the LTV signature, currently used to ascertain who signed the document and in what capacity, we could place the same data as in the blockchain. We could not, however, replicate other conditions in the blockchain (e.g., identify who placed the hash) due to address being associated with the institution as a whole. Pairing it with the strengths noted in Section 2.5.1, introducing a new process would be against **qubIT**'s interests as modules are already available that provide digital signature options through Fenix.

[2]A previous version of these benchmarks placed that figure at 2500 Transactions per Second (TPS) but has since been removed.

Regarding pricing models, it was clear that using the Enterprise version would be required, if not exclusively for the added database support. This would be factored into our pricing model, but the question remained if **qubIT** would take on the license or present it as a requirement to the end customer. This last option had one unknown factor: R3 does not use a standard pricing model, only custom pricing. This raised concerns about how different the prices would be for both the scenarios on the table.

Without a standard pricing table, and not knowing how would the network scale, estimating the costs was impossible. Pushing the negotiation to the customer side was not favoured as it would require the client to negotiate with R3 without technical knowledge. Taking on the cost would require the creation of a flexible pricing model as some costs would be negotiated up front and others diluted across the contract duration. In **qubIT**'s experience, clients prefer turnkey solutions and that is an important part of the value proposition as a solutions builder. After discussing it, we did not reach a conclusion on the pricing issues.

5.5.3 Ease of Use

Considering the number of participants and the desired topology (see Section 4.10), managing the network through some interface was desirable, extending beyond the monitoring and logging already planned. Building our own network manager was considered but not pursued due to the likelihood of our code breaking due to Corda updates and lack of support from R3. It was also a matter of "reinventing the wheel" as a tool like that is already provided when using the Enterprise version.

At this moment, having stalled in the previous two questions, it was becoming clear that Corda would not meet expectations so the discussion shifted to the Hyperledger project and if we should just rebuild the application using either Fabric or Sawtooth. Fabric had recently launched its 2.0.1 version, a big step ahead of the 1.4 version we used in the Chapter 3 comparison.

The software stack was an issue. Fabric has Java support but it is not favored over Go due to serialization issues (see Section 3.4.3). It will also only run on Docker, which we could accommodate, but being able to share our current servers was important to keep infrastructure costs down.

Reducing the information sharing problem to its basic elements, it is all a matter of putting data somewhere and retrieving it. The other options in consideration were Redis [108] (a in-memory key-value store) and Apache Artemis [109] (a message queue). Since information would not be stored in the communication medium (each Fenix instance would duplicate the data), simply transferred or consulted during a certain time window, both options are viable. As these integrate with Java with a complete API, it is simpler to integrate and use than having to create flows and associated states, propagate information to the ledger and then convert it back to domain objects.

5.6 Final Remarks

On a technical level, we were in agreement that Corda's shortcomings, especially regarding database support and multi-threading, were detrimental but still manageable through the use of the Enterprise edition. However, using Corda added permanent overhead costs that placed it at a disadvantage when compared to using another approach such as a key-value store (see Section 5.5.3).

With regards to being chosen as the data sharing medium, a key issue was Corda's state and flow oriented design. All of **qubIT**'s solutions follow the Domain-driven Design (DDD) paradigm, so adding more functionalities to Corda would require a conversion effort similar to the one presented in Section 4.5 for integrating with existing Fenix features. This could be aggravated if a completely new feature was being added as the extended domain would need to be crafted in a way that state extraction and flow logic would not be burdensome.

Another overhead source was that testing would need to be much more extensive, leading to slower release cycles, as an error in a flow running in a production system could have catastrophic consequences since data cannot be deleted and unconsumed states cannot be restored without rolling back every participant.

We also came to the conclusion that the document validation aspects represented a poor value proposition for possible clients. Since **qubIT** already offers digital signature options, it would be difficult to convince a client to buy the blockchain solution, with associated infrastructure costs, when it does not clearly surpass what is provided by the LTV signature standard.

Coupling that with GDPR concerns (e.g., users not wanting their data to be validated or rescinding consent) that would make it difficult to expand the validation beyond ULisboa, the Business Solutions team decided that this project would not be pursued any further, scrapping the Fenix bridging module and associated developments.

Chapter 6

Conclusion

With the goal of potentially opening up new business opportunities, this project set out as an exploration of using blockchain technology within **qubIT**'s core business. It fulfilled that purpose, albeit with a different outcome than initially expected. Blockchain proved to be a powerful tool but could not sustain the business proposition by itself. It was not a total loss as it provided a lot of know-how about the blockchain ecosystem and will help guide future projects.

Despite our comparison favoring Fabric, Corda was chosen as it required minimal tweaking of the current software stack, speeding up the proof of concept development. In the end, that choice was pivotal to our blockchain aspirations as Corda could fulfill the requirements, but its drawbacks and difficulty in adding value over existing digital signature offerings made it unfeasible to use in a production environment.

Nevertheless, the technological shortcomings continue to be addressed in recent releases but the design aspects (e.g., no shared state amongst all peers) place Corda in an odd position within the blockchain ecosystem.

From my perspective as the developer, even if Fabric had been chosen, we would arrive at a similar conclusion. It was a business-centric decision, not a technological one.

A core tenet of blockchain, decentralization, complicates the use of common revenue models such as a subscription. Being a middleman is not a strong position in a decentralized system that is designed by principle to operate without depending specifically on any of its participants. This begged an obvious question: how long could we generate revenue before the users simply cut us out? The safest position is offering the service strictly as a provider (e.g., the AWS model or "Blockchain-as-a-Service") and/or managing it for others. Hence the consistent push, even from platform builders, to license their technology for managed applications.

If you can assume that middleman position comfortably and without risking loss of income, decentralization is probably not needed to begin with. A distributed system using Redis [108], Kafka [110] or even ZooKeeper [111] can be used to offer the service and reap the benefits without the added complexity of blockchain, something that we are currently exploring in lieu of this solution.

We also found that the need for BFT and/or blockchain technology must be paired with a baseline level of distrust in the system itself and its participants to justify its overhead. If every participant implicitly trusts each other, either by rule of law or lack of reward for breaking deals, BFT can still be used but you only need to worry about actor takeover as no participant is expected to act maliciously. Crashes or non-malicious Byzantine faults can be handled by other mechanisms such as active replication.

Blockchain, in a trusted environment, loses some of its information security “shine” and becomes a way to provide an auditable system. But, with participants being trusted, this again creates high overhead and complexity, discouraging its use in favor of simpler solutions such as extensive logging. With the ELK stack [112], logging is almost hassle-free and eminently searchable. One can eschew **`logrotate`**, offload logs to a server that cannot be accessed by normal (non-admin) users and your logging is as secure as the off-site location.

These technological challenges are not insurmountable and can, given enough resources, be solved. The regulatory challenges, due to their nature and the quickly evolving technology, are a much tougher problem.

Corda, and other blockchain solutions, fall into a regulatory grey zone. We have digital signature laws and regulatory agencies that oversee those implementations (Agência para a Modernização Administrativa[1]), but there are no regulations regarding the value of a token transaction or the probatory value of a block timestamp. GDPR, with all of its complexity, makes it difficult for any meaningful data to be stored in an append-only format.

Can every operation be performed using an hash function for obfuscation? Yes, but we still need some way to connect data with that hash and add semantic meaning to it. This requires an external system to link hashes to documents or transactions. Which, again, begs the question: if we have that system, why not use it directly? We were not able to reach a good enough answer for this question, despite multiple software solutions offering this model.

6.1 Future Work

Considering the results obtained, **qubIT** will not pursue further blockchain projects until the legal gray area has been cleared up. There are still big regulatory hurdles that must be settled, probably at an European Union level. A better understanding of the technology is also necessary from the legislative bodies as a block being mined does not have an easy real world analog as does a physical signature versus a digital signature.

[1]https://www.ama.gov.pt/

As a secondary requirement, we will also look for a situation where a blockchain is not only viable but clearly a better choice regarding the value proposition for our customers.

Considering the decision flowcharts published by NIST [1] and CompTI [113], two big regulatory organizations, there are still very strong alternatives and few proven use cases for a hard bet on blockchain development. Most of the companies that are thriving in the blockchain space have other ventures, something **qubIT**, by sheer size alone, cannot do.

My personal take regarding blockchain is that I have no doubts that it will shape the future of many services, but the use cases still need more time to mature. We are still a few years away from government agencies adopting the use of blockchain for services such as land registries, especially considering that digital signatures have only relatively recently caught on and become more prevalent.

Acronyms

AWS Amazon Web Services. 3, 53

BAT Basic Attention Token. 2

BFT Byzantine Fault Tolerance. 9, 10, 12, 25, 54

CFT Crash Fault Tolerance. 25

DAG Directed Acyclic Graph. 8

DDD Domain-driven Design. 52

DLT Distributed Ledger Technology. 7, 8, 16

DoS Denial-of-Service. 1, 12

DSL Domain Specific Language. 34, 47

GDPR General Data Protection Regulation. 14–16, 19, 49, 52, 54

IDE Integrated Development Environment. 29

JBDC Java Database Connectivity. 45

JPA Java Persistence API. 46

JVM Java Virtual Machine. 21

LTV Long Term Validation. 15, 20, 50, 52

OCSP Online Certificate Status Protocol. 15

P2P Peer-to-Peer. 7

PoS Proof-of-Stake. 10

PoW Proof-of-Work. 10, 11

R&D Research and Development. 4

RPC Remote Procedure Call. 36, 41, 43

TPS Transactions per Second. 50

ULisboa University of Lisbon. 4, 15–18, 27, 30, 49, 50, 52

Glossary

fintech A portmanteau of 'financial technology', pertaining to companies in the technological and financial crossover space. 2

fork bomb Denial-of-service attack where a process replicates itself until it consumes all of the system's resources. 12

Kanban A lean method of sofware development where issues are placed on a board and moved along different states until completed. 29, 44

Bibliography

[1] D. J. Yaga, P. M. Mell, N. Roby, and K. Scarfone, “Blockchain Technology Overview,” NIST, Tech. Rep., Oct. 2018, accessed on Nov 2019. [Online]. Available: https://www.nist.gov/publications/blockchain-technology-overview

[2] R3. Overview of the corda state machine. Accessed on Feb 2020. [Online]. Available: https://www.corda.net/blog/overview-of-the-corda-state-machine/

[3] Z. Moezkarimi, F. Abdollahei, and A. Arabsorkhi, “Proposing a Framework for Evaluating the Blockchain Platform,” in *2019 5th International Conference on Web Research (ICWR)*, 2019, pp. 152–160.

[4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Bitcoin.org*, 2009, accessed on Nov 2019. [Online]. Available: https://bitcoin.org/en/bitcoin-paper

[5] D. Treat, “Powered by blockchain: Realizing AI’s full potential,” *Accenture*, Oct. 2018, accessed on Dec. 2019. [Online]. Available: https://www.accenture.com/us-en/insights/blockchain/ai-plus-blockchain

[6] J. Liu, A. Kadiyala, and P. Cannistraci, “Value Driven Supply Chain powered by Blockchain and IoT,” *Deloitte*, Jan. 2019, accessed on Dec. 2019. [Online]. Available: https://www2.deloitte.com/global/en/pages/technology/articles/deloitte-oracle-value-driven-supply-chain-by-blockchain-iot.html

[7] M. del Castillo and M. Schifrin, “Blockchain 50,” *Forbes*, Feb. 2020, accessed on Apr. 2020. [Online]. Available: https://www.forbes.com/sites/michaeldelcastillo/2020/02/19/blockchain-50/

[8] Google. ’blockchain’ - Explore - Google Trends. Accessed on Feb. 2020. [Online]. Available: https://trends.google.com/trends/explore?date=all&q=blockchain

[9] Brilliant.org. Merkle tree. Accessed on Feb. 2020. [Online]. Available: https://brilliant.org/wiki/merkle-tree/

[10] A. Back. (2002, Aug.) Hashcash - A Denial of Service Counter-Measure. Accessed on Feb. 2020. [Online]. Available: http://hashcash.org/papers/hashcash.pdf

[11] Coindesk. Bitcoin (BTC) Price Index. Accessed on Mar. 2020. [Online]. Available: https://www.coindesk.com/price/bitcoin

[12] Coinbase. Coinbase - Token List. Accessed on Jun. 2020. [Online]. Available: https://www.coinbase.com/price

[13] The Register. Single-line software bug causes fledgling YAM cryptocurrency to implode just two days after launch. Accessed on Aug. 2020. [Online]. Available: https://www.theregister.com/2020/08/13/yam_cryptocurrency_bug_governance/

[14] France24. Maduro bids to revive Venezuela's 'petro' cryptocurrency. Accessed on Aug. 2020. [Online]. Available: https://www.france24.com/en/20200114-maduro-bids-to-revive-venezuela-s-petro-cryptocurrency

[15] ITU-T Focus Group on Application of Distributed Ledger Technology, "Distributed ledger technology use cases," International Telecommunication Union, Tech. Rep., 2019, accessed on Jan. 2020. [Online]. Available: https://www.itu.int/en/ITU-T/focusgroups/dlt/Documents/d21.pdf

[16] M. M. Eljazzar, M. A. Amr, S. S. Kassem, and M. Ezzat, "Merging supply chain and blockchain technologies," *CoRR*, vol. abs/1804.04149, 2018. [Online]. Available: http://arxiv.org/abs/1804.04149

[17] J. Vos. (2017) Blockchain-based land registry : Panacea illusion or something in between? Accessed on Jan. 2020. [Online]. Available: https://www.elra.eu/wp-content/uploads/2017/02/10.-Jacques-Vos-Blockchain-based-Land-Registry.pdf

[18] F. Vogelsteller and V. Buterin. (2019) EIP-20: ERC-20 Token Standard. Accessed on Dec. 2019. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[19] Brave Browser. About Brave Rewards. Accessed on Dec. 2019. [Online]. Available: https://brave.com/brave-rewards/

[20] Hyperledger Fabric. Building Your First Network. Accessed on Feb. 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/build_network.html

[21] K. Zhang and H. Jacobsen, "Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1337–1346.

[22] A. F. Anta, K. Konwar, C. Georgiou, and N. Nicolaou, "Formalizing and implementing distributed ledger objects," *SIGACT News*, vol. 49, no. 2, p. 5876, Jun. 2018. [Online]. Available: https://doi.org/10.1145/3232679.3232691

[23] IOTA. Home | IOTA. Accessed on Feb. 2020. [Online]. Available: https://www.iota.org/

[24] B. Cao, Z. Zhang, D. Feng, S. Zhang, L. Zhang, M. Peng, and Y. Li, "Performance analysis and comparison of pow, pos and dag based blockchains," *Digital Communications and Networks*, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352864819301476

[25] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*. Self-published, 2020. [Online]. Available: http://toc.cryptobook.us/

[26] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security. FC 2014. Lecture Notes in Computer Science*, S.-N. R. Christin N., Ed., vol. 8437. Springer, Berlin, Heidelberg, 2014.

[27] M. Saad, J. Spaulding, L. Njilla, C. A. Kamhoua, D. Nyang, and A. Mohaisen, *Overview of Attack Surfaces in Blockchain*. John Wiley & Sons, Ltd, 2019, ch. 3, pp. 51–66. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119519621.ch3

[28] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, "On the Instability of Bitcoin Without the Block Reward," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 154167. [Online]. Available: https://doi.org/10.1145/2976749.2978408

[29] V. Buterin. (2013) A Next-Generation Smart Contract and Decentralized Application Platform. Accessed on October 2019. [Online]. Available: https://ethereum.org/en/whitepaper/

[30] S. Popov. (2018) The Tangle. Accessed on Mar. 2020. [Online]. Available: http://www.descryptions.com/Iota.pdf

[31] Kadena | Blockchain | Hybrid Platform. Accessed on Oct. 2019. [Online]. Available: https://www.kadena.io/

[32] M. Castro, B. Liskov *et al.*, "Practical Byzantine fault tolerance," in *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, 1999.

[33] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 305–319.

[Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[34] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[35] J. Sousa, A. Bessani, and M. Vukolic, "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*. IEEE Computer Society, 2018, pp. 51–58. [Online]. Available: https://doi.org/10.1109/DSN.2018.00018

[36] PoET 1.0 Specification. Accessed on Mar. 2020. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html

[37] Digiconomist. Bitcoin Energy Consumption Index. Accessed on Mar. 2020. [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption/

[38] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," in -, 2012, accessed on Mar. 2020. [Online]. Available: https://decred.org/research/king2012.pdf

[39] Iain Stewart. (2012) Proof of Burn. Accessed on Apr. 2020. [Online]. Available: https://en.bitcoin.it/wiki/Proof_of_burn

[40] M. Paquet-Clouston, B. Haslhofer, and B. Dupont, "Ransomware payments in the Bitcoin ecosystem," *Journal of Cybersecurity*, vol. 5, no. 1, 05 2019, tyz003. [Online]. Available: https://doi.org/10.1093/cybsec/tyz003

[41] Banco de Portugal. (2020) Consulta pública do Banco de Portugal n. 5/2020 Projeto regulamentar relativo ao registo de entidades que exercem atividades com ativos virtuais. [Online]. Available: http://bit.ly/bdpConsulta

[42] PayPal. (2020) Paypal launches new service enabling users to buy, hold and sell cryptocurrency. [Online]. Available: http://bit.ly/paypalPress

[43] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[44] Tendermint Core. Accessed on Nov. 2019. [Online]. Available: https://tendermint.com/core/

[45] Hyperledger Burrow. Accessed on Nov. 2019. [Online]. Available: https://www.hyperledger.org/use/hyperledger-burrow/

[46] Cosmos SDK. Accessed on Nov. 2019. [Online]. Available: https://tendermint.com/sdk/

[47] IBC Protocol. Accessed on Nov. 2019. [Online]. Available: https://tendermint.com/ibc/

[48] Ripple, Inc. (2020, Jun.) Open Source from Ripple, Inc. Accessed on Jun. 2020. [Online]. Available: https://github.com/ripple

[49] B. Chase and E. MacBrough, "Analysis of the XRP ledger consensus protocol," *CoRR*, vol. abs/1802.07242, 2018. [Online]. Available: http://arxiv.org/abs/1802.07242

[50] Apache Foundation. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/

[51] Amazon Managed Blockchain. Accessed on Mar. 2020. [Online]. Available: https://aws.amazon.com/managed-blockchain/

[52] IBM Blockchain. Accessed on Mar. 2020. [Online]. Available: https://www.ibm.com/blockchain

[53] M. Hearn and R. G. Brown. (2019) Corda: A distributed ledger. Accessed on Nov 2019. [Online]. Available: https://www.r3.com/reports/corda-technical-whitepaper/

[54] GraalVM Community. Accessed on Nov. 2019. [Online]. Available: https://www.graalvm.org/

[55] Azure Blockchain Service. Accessed on Mar. 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/blockchain/service/overview

[56] Oracle Blockchain. Accessed on Mar. 2020. [Online]. Available: https://www.oracle.com/blockchain/

[57] Amazon Quantum Ledger Database (QLDB). Accessed on Mar. 2020. [Online]. Available: https://aws.amazon.com/qldb/

[58] ANACOM. Aprova o regime jurídico dos documentos eletrónicos e da assinatura digital. Accessed on Dec. 2019. [Online]. Available: https://www.anacom.pt/render.jsp?categoryId=331655

[59] IETF. (2001, Aug.) Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). [Online]. Available: https://tools.ietf.org/html/rfc3161

[60] European Commission. (2020) eSignature standards. Accessed on Nov 2019. [Online]. Available: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eSignature+standards

[61] G. F. . R. K. Johannes Sedlmeir, Hans Ulrich Buhl, "The Energy Consumption of Blockchain Technology: Beyond Myth," *Business & Information Systems Engineering*, 2020, https://doi.org/10.1007/s12599-020-00656-x.

[62] European Commission. General Data Protection Regulation. Accessed on October 2019. [Online]. Available: https://gdpr-info.eu/

[63] ——. (2020) Art. 7 - General Data Protection Regulation. [Online]. Available: https://gdpr-info.eu/art-7-gdpr/

[64] Erasmus Without Paper. Accessed on Oct. 2019. [Online]. Available: https://www.erasmuswithoutpaper.eu/

[65] A. Kamilaris, A. Fonts, and F. X. Prenafeta-Bold, "The rise of blockchain technology in agriculture and food supply chains," *Trends in Food Science & Technology*, vol. 91, p. 640652, 2019.

[66] T. Sund, C. Lööf, S. Nadjm-Tehrani, and M. Asplund, "Blockchain-based event processing in supply chainsA case study at IKEA," *Robotics and Computer-Integrated Manufacturing*, vol. 65, p. 101971, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584519301905

[67] Etherscan. Ethereum Average Block Size Chart. Accessed on May 2020. [Online]. Available: https://etherscan.io/chart/blocksize

[68] ——. The Ethereum Blockchain Explorer. Accessed on May 2020. [Online]. Available: https://etherscan.io/

[69] Banco de Portugal. Moedas virtuais. Accessed on Nov. 2019. [Online]. Available: https://www.bportugal.pt/page/moedas-virtuais

[70] Hyperledger Fabric - GitHub. Accessed on Oct. 2019. [Online]. Available: https://github.com/hyperledger/fabric

[71] GitHub | Hyperledger Fabric. Pull requests - hyperledger/fabric. Accessed on Mar. 2020. [Online]. Available: https://github.com/hyperledger/fabric/pulls?q=is%3Apr+merged%3A2019-09-01..2019-12-31

[72] GitHub - Hyperledger. Hyperledger - Members. Accessed on Mar. 2020. [Online]. Available: https://github.com/orgs/hyperledger/people

[73] StackOverflow. 'hyperledger-fabric' Top Users. Accessed in Mar. 2020. [Online]. Available: https://stackoverflow.com/tags/hyperledger-fabric/topusers

[74] R3. R3 Marketplace | Directory. Accessed on Nov. 2019. [Online]. Available: https://marketplace.r3.com/directory?referrer=featured-partners

[75] ——. R3 Marketplace. Accessed on Nov. 2019. [Online]. Available: https://marketplace.r3.com/solutions?referrer=featured-solutions

[76] Corda - GitHub. Accessed on Oct. 2019. [Online]. Available: https://github.com/corda/corda

[77] GitHub - Corda. Corda - Members. Accessed on Mar. 2020. [Online]. Available: https://github.com/orgs/corda/people

[78] StackOverflow. 'corda' Top Users. Accessed in Mar. 2020. [Online]. Available: https://stackoverflow.com/tags/corda/topusers

[79] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential Risks of Hyperledger Fabric Smart Contracts," in *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2019, pp. 1–10.

[80] R3. Notary demo. Accessed on May 2020. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.5/samples/notary-demo

[81] Hyperledger Fabric - Documentation. Hyperledger FAQ - Security & Access Control. Accessed on Set. 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/Fabric-FAQ.html#security-access-control

[82] ——. Gossip data dissemination protocol. Accessed on Set. 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/gossip.html

[83] R3. Sizing and performance. Accessed on Feb. 2020. [Online]. Available: https://docs.corda.net/docs/corda-enterprise/4.3/sizing-and-performance.html

[84] R. Han, G. Shapiro, V. Gramoli, and X. Xu, "On the performance of distributed ledgers for internet of things," *Internet of Things*, vol. 10, 2020. [Online]. Available: https://doi.org/10.1016/j.iot.2019.100087

[85] S. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, “Performance evaluation of blockchain systems: A systematic survey,” *IEEE Access*, vol. PP, pp. 1–1, 06 2020.

[86] Hyperledger Cactus. Accessed on Set. 2020. [Online]. Available: https://www.hyperledger.org/use/cactus

[87] Hyperledger Besu. Acccessed on May 2020. [Online]. Available: https://www.hyperledger.org/use/besu

[88] ConsenSys Quorum. Accessed on Dec. 2019. [Online]. Available: https://consensys.net/quorum/

[89] Google’s LevelDB. Accessed on Mar. 2020. [Online]. Available: https://github.com/google/leveldb

[90] Apache CouchDB. Accessed on Mar. 2020. [Online]. Available: https://couchdb.apache.org/

[91] Gradle Build System. Accessed on Dec. 2019. [Online]. Available: https://gradle.org/

[92] VSCode-Corda. Accessed on Jan. 2020. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=R3.vscode-corda

[93] Truffle Suite. Accessed on Feb. 2020. [Online]. Available: https://www.trufflesuite.com/

[94] R3. ContractsDSL. Accessed on Oct. 2020. [Online]. Available: https://api.corda.net/api/corda-os/4.6/html/api/javadoc/index.html

[95] ——. API: Flows | Corda OS 4.6 | Corda Documentation. Accessed on Oct. 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.6/api-flows.html

[96] ——. Oracles | Corda OS 4.6 | Corda Documentation. Accessed on Oct. 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.6/key-concepts-oracles.html

[97] ——. Consensus | corda os 4.6 | corda documentation. Accessed on Oct. 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.6/key-concepts-consensus.html

[98] ——. Validity Consensus | Corda OS 4.5 | Corda Documentation. Accessed on May 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.5/key-concepts-consensus.html#validity-consensus

[99] ——. Writing a custom notary service (experimental) | Corda OS 4.6 | Corda Documentation. Accessed on Oct. 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.6/tutorial-custom-notary.html

[100] ——. Corda API Documentation - All Classes. Accessed on Oct. 2020. [Online]. Available: https://api.corda.net/api/corda-os/4.6/html/api/javadoc/allclasses-noframe.html

[101] ——. Notaries | Corda OS 4.6 | Corda Documentation. Accessed on Oct. 2020. [Online]. Available: https://docs.corda.net/docs/corda-os/4.6/key-concepts-notaries.html

[102] H2. H2 database engine. Accessed on Jan. 2020. [Online]. Available: https://www.h2database.com/html/main.html

[103] ——. H2 frequently asked questions. Accessed on Jan. 2020. [Online]. Available: http://www.h2database.com/html/faq.html

[104] MariaDB Enterprise Open Source Database & SkySQL MariaDB Cloud | MariaDB. Accessed on Jan. 2020. [Online]. Available: https://mariadb.com/

[105] S. M. Fernandes and J. a. Cachopo, "Lock-free and scalable multi-version software transactional memory," in *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 179188. [Online]. Available: https://doi.org/10.1145/1941553.1941579

[106] Parallel Universe. Accessed on Jan. 2020. [Online]. Available: https://docs.paralleluniverse.co/quasar/

[107] R3. Performance benchmarking results | corda enterprise 4.6 | corda documentation. Accessed on Oct. 2020. [Online]. Available: http://bit.ly/cordaBenchmarks

[108] Redis. Accessed on May 2020. [Online]. Available: https://redis.io/

[109] ActiveMQ Artemis. Accessed on May 2020. [Online]. Available: https://activemq.apache.org/components/artemis/

[110] Apache Kafka. Accessed on May 2020. [Online]. Available: https://kafka.apache.org/

[111] Apache ZooKeeper. Accessed on May 2020. [Online]. Available: https://zookeeper.apache.org

[112] What is the ELK Stack? Accessed on Dec. 2019. [Online]. Available: https://www.elastic.co/what-is/elk-stack

[113] CompTIA. (2019, Nov.) Blockchain decision tree. Accessed on Jan. 2020. [Online]. Available: https://www.comptia.org/content/infographic/blockchain-decision-tree