A Technical Framework for Supply Chain Transformation using Blockchain

Final Year Project

Matt Dean

B.Sc. Computer Science with Year in Industry

School of Computer Science
University of Birmingham
2018

Preamble

Abstract

Improving Supply chain systems offers benefits to quality, cost, and regulatory compliance, but interoperability between organisations is an obstacle. This project addresses the issue of interoperability through the introduction of blockchain technology, implementing a complex supply chain model from literature on the Hyperledger platform.

The process of translating a supply chain network model to an implementation is described in detail; coupled with the the complexity of the network model chosen, this allows the system to be generalised well to a variety of supply chain use cases. The system includes infrastructure, application and integration layers, providing an end-to-end reference implementation for creation of supply chain systems in enterprise.

Software

The software for this project can be found in the Git repository located at:
https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/mxd434.

Acknowledgements

First and foremost I'd like to thank my supervisor, David Galindo, for his unwavering support throughout the lifespan of the project. Before the project had even started, David encouraged my ideas and enthusiasm for blockchain.

Through his insightful and challenging questions, David prompted me to deepen my understanding of blockchain and it's use cases, as well as offering invaluable advice on managing the project during our regular meetings.

I'd also like to thank Steve Vickers for offering constructive feedback at a key point in the project, providing the inspiration to pivot from a generic supply chain model to a specific one.
Finally, many thanks to Steve and Sandy Gould for their attentiveness and feedback at the demonstration.

Contents

Chapter 1

Introduction

Supply chain traceability is becoming increasingly important as regulators and governments concerned about public safety mandate ever more stringent rules, and consumers demand certified high quality products such as organic.

The weak link in current traceability systems is often the exchange of information between entities - from one proprietary system to another. Due to the strong economic and legal incentives, companies are scrambling to find an edge in this contested space.

Blockchain has been cited by many as a technology poised to revolutionise the supply chain ecosystem, promising to address the challenges of interoperability between organisations.
This paper describes the creation of a blockchain traceability system from a supply chain network model, showing how blockchain technology is ideally placed to address the shortcomings of existing traceability systems.

1.1 Report structure

The structure of this report is as follows:

1.1.1 Background

An overview of current literature covering supply chain traceability and blockchain.

First, some background on traceability, and exploration of the drivers behind improving traceability.
Second, detailing of common issues or deficiencies in traceability systems, and synthesis of the common attributes of successful traceability systems.

Lastly, a background on blockchain, and review of related work at the intersection of blockchain and supply chain research.

1.1.2 Specification

First a hypothesis is generated, proposing a blockchain-based supply chain system.
Second, the process of requirements capture is described, along with the specification of the desired properties of the proposed system.

1.1.3 Design

A more detailed design of the system, following the requirements set out in the previous section. Specifically, the choice of blockchain platform is discussed in at length, along with the choice of supply chain network model and software engineering methodology.

1.1.4 Implementation

Details of how the design is actualized.
First, creating the infrastructure and data models from the supply chain network model, then implementing the transactions between participants.
Second, implementing higher-level features such as access control and traceability.

1.1.5 Testing

This section discusses the testing strategy used, detailing unit and integration tests, and showing how traceability information can be extracted.

1.1.6 Discussion

Finally, the system is evaluated against the specification and related work.

Chapter 2

Background

2.1 Traceability

2.1.1 What is traceability?

There are many papers tackling the subject of traceability, particularly in the food industry. First it may be useful to define what exactly we are tracing.
Moe [43] draws a distinction between two types of traceability:

1. Product; it may relate materials, their origin, processing history, and their distribution and location after delivery.

2. Data; it relates calculations and data generated throughout the quality loop, sometimes back to the requirements for quality.

This paper focuses on the supply chain, which most closely relates to product traceability.

To provide some formal definitions, the ISO standard [32] defines product traceability as "the tracking of consumer products with respect to the origin of materials and parts; the processing history; the distribution and location of the product or service after delivery".
The European Union [53] describes traceability similarly as "an ability to track any food, feed, food-producing animal or substance that will be used for consumption, through all stages of production, processing and distribution".

Of course, traceability is not a topic restricted to the food industry, indeed, product traceability is used in diverse industries such as manufacturing [1], art [51], and pharmaceuticals [46].
For this reason then, the definition is necessarily more broad. Golan et al. [15] also argue for a loose definition of this term, stating "traceability is a tool for achieving a number of different objectives".

The term provenance, or a range of other terms like lineage or pedigree are used in place of traceability in some papers.[14] For simplicity and avoidance of confusion this paper will use the term traceability throughout.

Overall then, traceability may be loosely defined here as tracking a physical item through certain transformations or a chain of custody.

2.1.2 Example use case

An example use case for traceability can be seen in the food industry: in order to protect the name 'Parmigiano Reggiano' against imitations, a consortium of Italian providers implemented a traceability system to certify authentic cheese. A cow's milk is traced to a dairy where it is made into cheese; the cheese is then tracked throughout it's lifespan during ageing and distribution to retailers.

Information is recorded at multiple stages in the supply chain, for example at the dairy, attributes such as cooking temperature and the cow code are recorded. After this an RFID chip is attached to uniquely identify that 'batch' (a wheel) of cheese. After production, the cheese is aged in a warehouse, where attributes such as temperature and humidity are tracked. Before being sent to market, the cheese is cut into smaller portions which have an alphanumeric identification number printed on the packaging.

This all serves to give complete traceability of the cheese from cow to table, meaning customers can be certain the cheese they are eating is authentic Parmigiano Reggiano.[48]

2.1.3 Benefits of traceability in the supply chain

In the food industry there are many papers on the subject of traceability, due to the high economic incentives. A report by the United States Department of Agriculture (USDA) on traceability states that "traceability systems tend to be motivated by economic incentives, not government traceability regulation" and "When the cost of distributing unsafe food goes up, so too do the benefits of traceability systems".[15]

Economic benefits to the company may include: [15]

- Supply management - companies can improve the efficiency of logistics activities using the data collected in a traceability system
- Quality and safety - product recalls can be more targeted and therefore less expensive because companies can track contaminated goods more precisely. It may also be possible to determine liability using traceability information.
- Higher sales may be realised through product differentiation via credence attributes like certified fair trade or organic products.

In countries other than the U.S.A., varying degrees of food traceability may also be required by law so as to improve public health. For example, after a widespread salmonella outbreak in Sweden, the government mandated stringent import checks for fresh meat and eggs.[12]

The ability to determine provenance often becomes desirable in the case where a high-value object has had multiple owners, since it can be useful for proving ownership or authenticity. Examples of assets for which traceability would be desirable include wine [5], diamonds [49], or paintings [51].

Situations where errors are very costly may also have high suitability for a traceability system, for example in healthcare it is critical to determine provenance of blood and organ donations [44]. In fact, healthcare realises the same benefits as the food industry in improving supply efficiency, determining liability, and increasing quality and safety (through reduction of errors).
An additional benefit in this sector is anti-counterfeiting - "one of the best weapons against counterfeited drugs is a complete track and trace management systems able to authenticate drugs and their origins".[40] Again regulation has been put in place by the U.S. Food and Drug Administration (FDA) to mandate traceability.[13]

2.1.4 Issues with existing supply chain traceability systems

Golan et al. [15] argue that many companies have efficient internal proprietary traceability systems already in place. However, in many cases the difficulty in tracing a product from one end of the supply chain to the other is prohibitive.

In the United States spinach outbreak in 2006, no spinach was sold or eaten in the entire country for over two weeks.[55] During the Scandinavian salmonella outbreak in 2005, government response was inhibited by the complexity of tracing contaminated products, delaying intervention by weeks.[37]

The solution to this problem is interoperability between actors in the supply chain. [2] Bechini et al. [4] state "for example, the retailer should not be coerced to interact with a huge number of disparate systems just in order to accommodate traceability requirements ... it would be preferable to access a single system able to provide all the required information".

One way to implement such interoperability would be a centralised system, but Golan et al. [15] argues that may be difficult or even impossible, even with the introduction of new regulation.

> Government may also consider mandating traceability to increase food safety, but such a mandate may impose inefficiencies on already efficient private traceability systems. The already widespread voluntary use of traceability complicates the application of a centralized system because firms have developed so many different approaches and systems of tracking. If mandatory systems fail to allow for variations in traceability systems, they will likely end up forcing firms to make adjustments to already efficient systems or to create parallel systems.

2.1.5 Attributes of successful supply chain traceability systems

Here the attributes and structure of successful traceability systems are explored, in order to build a framework for creating a blockchain-based system later in the paper.

Golan et al. [15] state that the broad definition of traceability "makes it difficult to extract characteristics of good system ... the characteristics of good traceability systems vary and cannot be defined without reference to the system's objectives". On the other hand, Storøy et al. [50] argue that "There are some well established methods and principles that underlie efficient implementation of traceability in the food industry; many of them described in previous papers and in various guidelines".
Focusing on supply chain here allows the common structure of a typical traceability system to be discovered.

Beginning with a definition from McKean [42]:

> Traceability of a product requires a transparent chain of custody to achieve credibility and to complete the desired information transfer functions. Product traceability has two components, as follows:
>
> - unique animal or product identification systems
> - a credible and verifiable chain of custody or identity

Regattieri et al. [48] go further:

> A product traceability system, and particularly a food traceability system, is fundamentally based on 4 pillars: product identification, data to trace, product routing, and traceability tools.

This prompts some questions to consider when designing traceability systems:

- Product identification - how is the product uniquely identified?
- Data to trace - what data needs to be recorded? how should it be stored? who should have access to it?
- Product routing - at what stage in the supply chain is data recorded?
- Traceability tools - what extra tools are needed to integrate the traceability system with existing supply chain processes?

Regattieri's "data to trace" and "product routing" steps could be combined, into a step named something like "recording of traceability data". Now Regattieri's adapted definition looks very similar to McKean's i.e. a traceability system comprising two parts - unique product identification and some record of the transformations this product undergoes.

These two components may suffice for an internal traceability system, i.e. within one organisation. However, for effective end-to-end supply chain traceability these proprietary systems need to communicate with each other i.e. between organisations. Therefore, effective traceability systems must have a third component - inter-organisation communication.

To summarise, a traceability system consists of 3 conceptual components:

1. Unique identification of traceable units
2. Record of transformations of traceable units
3. Information exchange between supply chain actors

Discussing each in turn:
Unique identification of traceable units
Kim et al. [35] introduce the concept of a Traceable Resource Unit (TRU) as being a 'batch' of the asset being tracked, for example a crate of fish, or in the case of discrete entities e.g. paintings, the entity itself. Thakur and Donnelly [52] expand on this by stating that in practice, a TRU "refers to the smallest unit that is exchanged between two parties in the supply chain".
There are many approaches to the physical realisation of this - it could be a barcode, an RFID chip, or in the case of some high value assets, a microsopic engraving of a serial number.[49]

Record of transformation of traceable units
The data model for the system must be designed to capture all the information necessary for traceability at each stage in the supply chain, and this data must be stored reliably.

Information exchange between supply chain actors
Traceability systems are typically proprietary [15] so a standardised method of data exchange is needed. In today's global economy companies produce and sell products all over the world, so the data must be exchanged across borders. Some international standard data formats do exist, for example GS1 [33], but these may limit the expressibility of storing supply chain information due to their generic nature. Thakur and Donnelly [52] state that "To enable effective electronic information exchange, work needs to be carried out on a sector-specific level".

2.2 Blockchain

2.2.1 What is blockchain?

A somewhat nebulous term, blockchain could be described as a group of technologies sharing similar properties. Jansson and Peterson [33] provide a detailed summary of blockchain definitions from literature.

Consolidated from those definitions, the pertinent properties of blockchain technology are:

- Distributed / Peer-to-peer - network peers are supply chain participants.
- Timestamped shared ledger - a record of transactions for which every peer has a copy.
- Smart contracts - programmable logic associated with transactions.

Note that in a smart contract the logic is "tied to the transaction itself, contrasting with business logic associated with a standard database application, in which rules [(for example, access control)] are often set at the entire database level, or in the application, but not in the transaction".[18]
Many blockchain systems have some other useful properties in addition to, or as a result of, those listed above, for example making it easy detect fraudulent behaviour.[18]

2.2.2 How does it work?

At a high level, participants submit transactions to update the ledger. Participants agree which updates are actually included in the ledger history through some consensus protocol.

Public and Permissioned

Blockchains can be split into two general categories, public and private (also known as permissioned).

In a public blockchain network, anyone can participate. This means participants can be pseudonymous and there must be a rigorous consensus process in place to avoid malicious entities manipulating the state of the ledger. Well-known public blockchain networks include Bitcoin, Ethereum, and many other cryptocurrencies.[3]

In a private, or permissioned blockchain network, only selected, trusted participants can update the ledger. This means a less rigorous (less resource intensive) consensus algorithm can be employed.[18] Well-known private blockchain networks include Hyperledger Fabric, MultiChain, and Corda.[17]

Byzantine Fault Tolerance

When talking about consensus protocols, the concept of Byzantine Fault Tolerance (BFT) is important. The name comes from a paper titled "The Byzantine Generals Problem" [38], where the network participants are described as generals attempting to agree on a common battle plan:

> Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement.

If a network is Byzantine Fault Tolerant, it guarantees the transactions in the ledger are correct if "a qualified majority of the nodes are behaving correctly, even if the faulty nodes behave in arbitrary and adversarial ways".[7]

Consensus

In a public blockchain network it is usually desirable for the consensus protocol to be BFT, for example Bitcoin's Proof-of-Work consensus mechanism.[45] The problem with consensus mechanisms of this class is their high resource requirements and low transaction throughput.[47]
Permissioned blockchains, as a result of only allowing trusted actors to participate in the network, can use a much simpler voting-based consensus algorithm, which allows for higher transaction throughput and lower resource costs.[20]

2.2.3 Benefits of a blockchain based system for traceability

During the discussion of traceability, it was identified that the major obstacle for improving supply chain traceability is interoperability between participants. The use of standardised data formats and centralisation can make some headway, but may be difficult to implement across the supply chain due to low economic incentives.
Blockchain systems are designed to be distributed, with entities sharing transaction information through the ledger. As a result, blockchain could be a suitable technology framework for implementing a traceability system, because blockchain platforms have many of the technical requirements for a traceability system baked in.

2.2.4 Existing work on blockchain-based traceability systems

Here existing work on traceability systems using blockchain is examined.

Chen

Chen [8] implements a supply chain with a generic idea of a commodity and a participant using Hyperledger. See Figure 2.1 for a visual of Chen's network model.

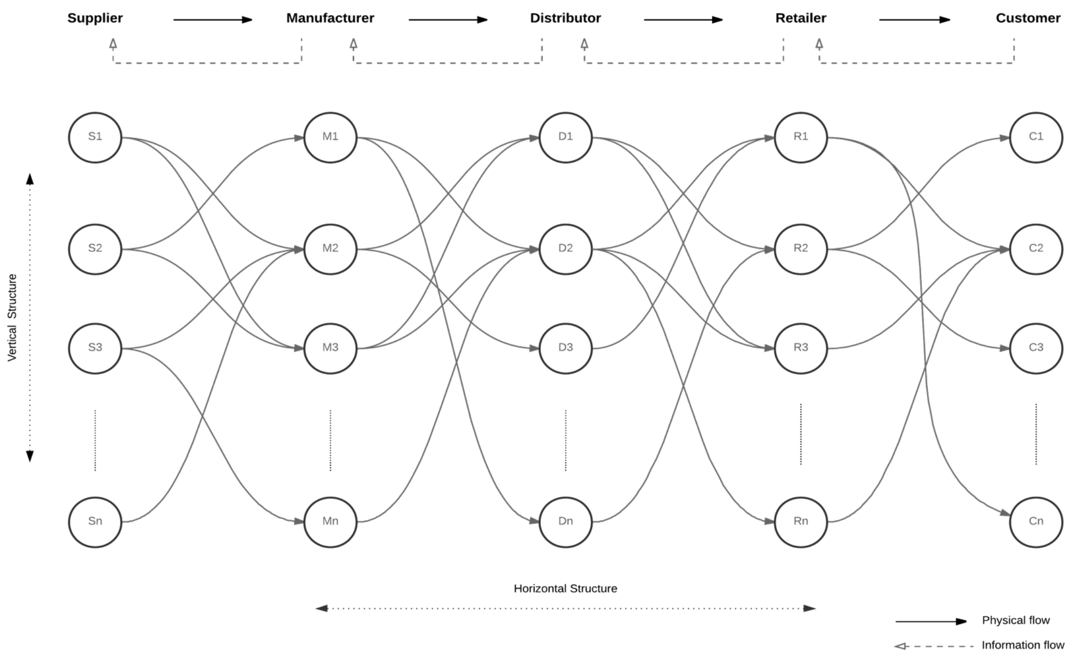


Figure 2.1: Chen's Supply Chain network model

Chen's implementation is less useful than it could be because it's too generic. This works fine as a proof-of-concept, but as shown in the requirements and design sections, much more needs to be built when implementing a more complex supply chain network model for enterprise.

Kim

Kim and Laskowski [36] use TOVE ontology to define model and implements a proof-of-concept on the Ethereum network. This is again a very generic or low level implementation, abstracting all asset types into a single TRU type. This provides a good basis for further work, but struggles with same issues as Chen, namely not generalising well to more complex supply chain models.

A second issue with the approach described is the choice of a public (non-permissioned) blockchain. While this has the advantage of being Byzantine Fault Tolerant, it does not scale well in a business setting due to the low data throughput and high computational requirements of the Ethereum (public) network - partly as a result of using a Byzantine Fault Tolerant consensus algorithm.

Biswas

Biswas et al. [5] use MultiChain to track wine through the supply chain but doesn't cover the technical implementation.
What that paper does cover well is the network model and data requirements at each stage in the supply chain. See Figure 2.2 for a visual of their network model.

Other related work

Liang et al. [39] and Kaku [34] separately implement data provenance systems on MultiChain. From their implementations, conclusions can be drawn about how data storage should be implemented as part of a supply chain traceability system.

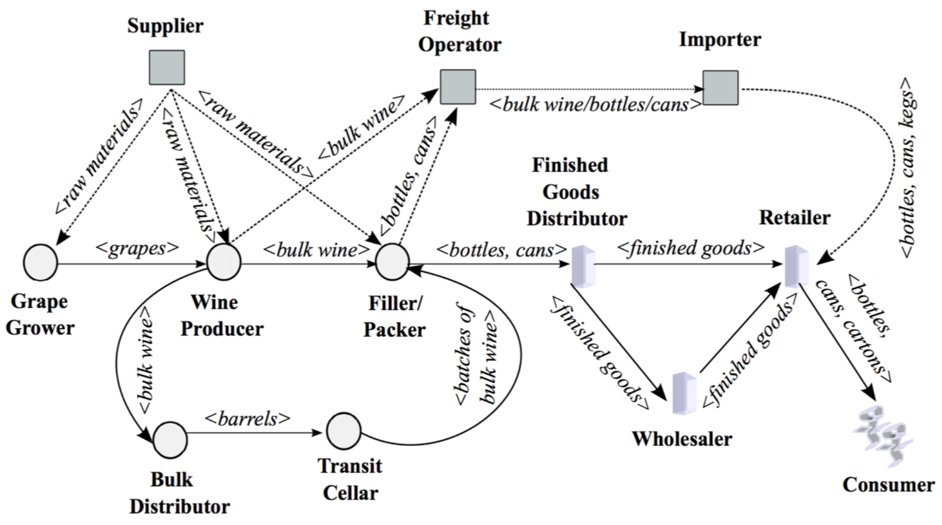


Figure 2.2: Biswas et al. Supply Chain network model

2.2.5 Evaluating if blockchain is a good fit for an organisation

While blockchain technology now seems applicable in a traceability context, it's also important to look at things from a higher level. As shown when discussing traceability, supply chains can vary widely across verticals and even within an industry. The application of blockchain should be evaluated at an organisational level, for example using the framework described in Jansson and Peterson's paper.[33]

2.2.6 Levels of blockchain adoption

Coordination of immediate blockchain adoption across the entire supply chain seems infeasible, but the benefits of blockchain technology can be realised on a smaller scale. Iansiti and Lakhani [31] give some examples of how blockchain adoption may be useful on a more localized scale:

- In a single organisation, a blockchain could be used to consolidate and replace multiple internal databases.
- Over a small number of organisations, it could be used to simplify complex transaction interactions such as tracking batches of commodities through steps in a supply chain where they may be mixed or blended.

Chapter 3

Specification

3.1 Hypothesis

Frameworks to analyse the business case for blockchain in the supply chain are already in place [33]. If evaluation using the frameworks shows a positive lean to implementing blockchain, how should one proceed?
Companies need a technical framework for creating blockchain supply chain systems, and a process to follow for integrating blockchain into their supply chain.
An implementation of a sufficiently complex supply chain model could serve as a proof of concept and a useful point of reference for developers creating similar systems within their organisation.

3.2 Process

The majority of my technical requirements have been taken from a collection of sources from literature. From there we get the high level user requirements from a successful traceability system, and extrapolate those to the features of a system implementing traceability capabilities.

3.3 Specification

3.3.1 Traceability systems

From section 2.1.5, a successful traceability system has three main components:

1. Unique identification of traceable units
2. Record of transformations of traceable units
3. Information exchange between supply chain actors

Unique identification of traceable units

The concept of a TRU will certainly factor into the data model, but the physical realisation of this, for example an RFID chip, is out of scope of a software project. Here I will assume each asset can be allocated a unique identifier, assuming it would be attached to the asset in the real world, for example via a label with a barcode.

Record of transformations of traceable units

From component two of a traceability system, the proposed system should record and store traceability data every time a TRU is transformed.

Req. 1. Record data every time a TRU is sold or converted into another TRU.

This record must be queryable so the traceability information can actually be used.

Req. 2. The record of traceability information must be queryable, allowing users to find the full history of an asset.

Since the proposed system implements the entire supply chain, it must also have business logic associated with these transformations, i.e. the transformations should be programmable.

Req. 3. The system must support programmable transformations.

Information exchange between participants

Each participant should have the ability to submit transactions and query traceability info, so the system must support some method of identity management.

Req. 4. The system must allow credentials for participation in the network to be issued and revoked.

It is also desirable to restrict access to some assets, for example so participants are only able to modify commodities they own.

Req. 5. The system must support expressive access control based on identity so operations on, and views of, data in the network can be restricted.

Participants must agree on the history of the supply chain - otherwise traceability information may be incorrect.

Req. 6. The system must include some consensus mechanism to ensure all participants agree on the state of the ledger.

3.3.2 Business context

Preconditions for adoption

Typically, multiple organisations take part in a supply chain system.[15] It is reasonable to assume different organisations have have differing technology stacks - perhaps using a different cloud infrastructure provider, for example.

Req. 7. The system should be able to run on multiple hardware platforms, ideally on common or easily accessible platforms such as GCP or AWS.

It is not feasible for all organisations in the supply chain to switch over to a completely new supply chain immediately.

Req. 8. Adoption of the system should be possible in an incremental manner, i.e. the system must present an interface enabling it to integrate with existing systems.

The specific implementation of business logic within a supply chain will vary widely between use cases and industries.

Req. 9. Developers within an organisation should find it straightforward to generalise the software implementation given in this paper when creating a system tailored to their use case.

Requirements during day to day operation

One of the three main drivers for creating a new supply chain system is to increase efficiency within the supply chain.

Req. 10. The system must have low running costs, and the potential for high transaction throughput.

The system should have longevity and be able to store the traceability history of old or long-lived items.

Req. 11. The system must have the ability to update software and business logic while the network is running, without losing the traceability history of assets.

Req. 12. The system must have the ability to add participants to the network while it is running.

Chapter 4

Design

4.1 Blockchain platform

4.1.1 Preexisting platform

An important part of the design phase for this project is choosing the software platform on which to build. As seen in the literature review, there are many options available - Hyperledger, Multichain, or even implementing a blockchain from scratch.
Use of an existing software platform was a better fit for the overall goals of the project. The project aims to provide a reference implementation for enterprise, so choosing an existing and well known platform over a custom implementation was important for fulfilling *Req. 8.*, to have confidence that the implementation would generalise to a variety of use cases without issues. Use of an existing platform also enables enterprise adoption due to it being more well trusted.

4.1.2 Public vs Private

Req. 10. The system must have low running costs, and the potential for high transaction throughput.
There are some public blockchains designed for high transaction throughput [47], but only private blockchains offer the combination of both low running costs and high throughput, as a result of their simpler consensus process.
With a simpler consensus mechanism, there is opportunity for malicious actors to corrupt the history of the ledger. However, in an enterprise consortium where companies collaborate on the same supply chain, this is not an issue, because the participants are limited and the consortium can verify the identity of participants before allowing them to join.

4.1.3 Feature set

Next, it is important to determine which platforms provide the necessary feature set described in the requirements. What follows is a summary of the requirements I have determined to be most important to platform selection for each platform. Of course it may be possible to bolt on features where they do not already exist for a platform, but unless there are other advantages, the platform that fulfills the most requirements out-of-the-box will be selected.
Below follows an assessment of the feature sets of the three most popular private blockchains - Hyperledger, MultiChain, and Corda.[17]

Requirement	**Hyperledger** [20]	**Corda** [19]	**MultiChain** [16]
Req. 2. Queryable distributed ledger	Yes (Pluggable)	Yes (SQL)	Yes (Bitcoin based)
Req. 5. Access control	Yes	Yes	Yes
Req. 3. Smart contracts	Yes (Chaincode)	Yes (CorDapps)	No
Req. 6. Consensus	Yes (Pluggable)	Yes (Pluggable)	Yes (Round robin)

4.1.4 Decision

Hyperledger and Corda both fulfil all the requirements, but looking at the overall aim of each platform, the better choice becomes apparent:

- **Hyperledger**: “Hyperledger’s strategy encourages the re-use of common building blocks via a modular architectural framework” [20]
- **Corda**: “a shared ledger fabric for financial services use-cases” [6]

Corda has since been adapted for non-financial use cases, but since Hyperledger was designed for modularity from the beginning it offers the most flexibility for the use case of supply chain.

4.2 Hyperledger

Having selected Hyperledger, next follows an exploration of the architectural concepts, showing how these relate to the requirements.

4.2.1 Background

“**Hyperledger** is an open source collaborative effort created to advance cross-industry blockchain technologies.” [30]. There are multiple projects under this umbrella, but here I will briefly describe two - Fabric and Composer.

Fabric is the core platform and infrastructure that the blockchain system will run on. The Hyperledger webpage states: “Hyperledger Fabric is a blockchain framework implementation ... intended as a foundation for developing blockchain applications” [29]

Composer is an abstraction layer for building business logic and integrations quickly. The Hyperledger webpage states: “Hyperledger Composer is a set of collaboration tools for building blockchain business networks that make it simple and fast ... to create smart contracts and blockchain applications [on Hyperledger Fabric]” [23]
Composer was introduced because, while it is possible to create a blockchain system directly on Fabric, there is a large amount of boilerplate code required. In addition, the interface presented by e.g. the Fabric SDK is very limited and does not fulfil *Req. 8.*. Therefore a need a higher level of abstraction is required.

Next follows a summary of the main architecture concepts not treated in the following sections.[26] See Figure 4.1 for a diagram.

- Organisation (Org) - A company or similar “legally separate entity”.
- Consortium - A group of organisations that participate in the same network.
- Peer - “A network entity that maintains a ledger and runs chaincode containers in order to perform read/write operations to the ledger. Peers are owned and maintained by organisations”.

4.2.2 Identity

Req. 4. The system must allow credentials for participation in the network to be issued and revoked.
This is achieved through the Fabric Certificate Authority (CA) - a component of the fabric architecture which has the capability to:[24]

- register identities, or connect to existing authentication systems
- issue certificates
- renew and revoke certificates

Each organisation has one CA - see Figure 4.1 for a visualisation.

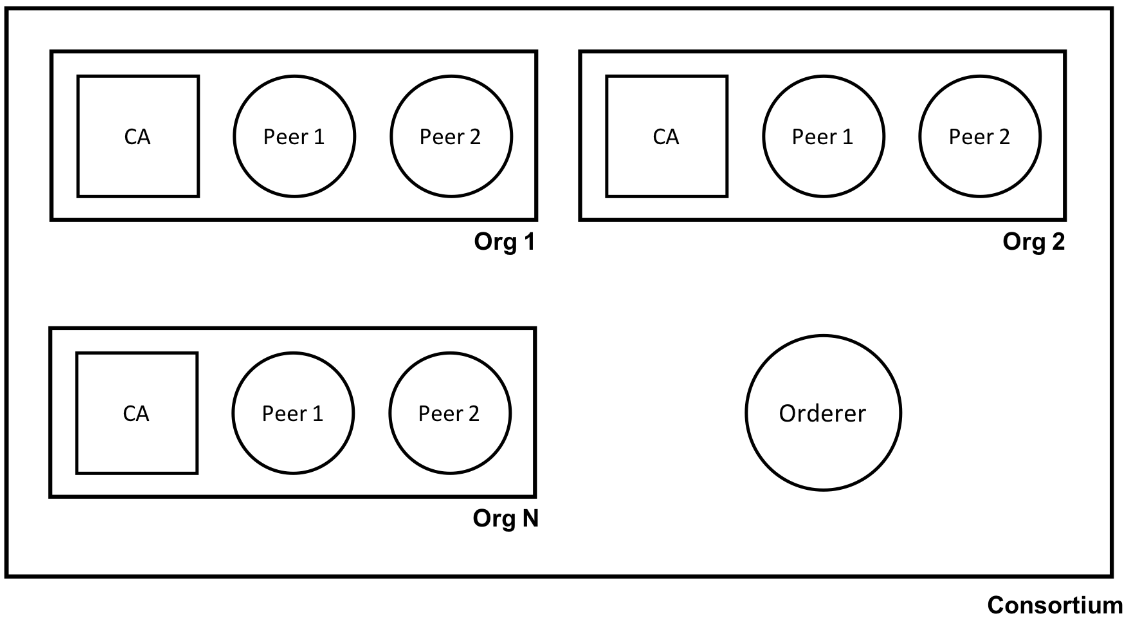


Figure 4.1: Fabric Architecture Diagram

4.2.3 Consensus

Req. 6. The system must include some consensus mechanism to ensure all participants agree on the state of the ledger.

Hyperledger Fabric supports multiple 'pluggable' consensus mechanisms for each phase of consensus:[20]

- Endorsement is driven by policy (eg m out of n signatures) upon which participants endorse a transaction.
- Ordering accepts the endorsed transactions and agrees on the order to be committed to the ledger.
- Validation takes a block of ordered transactions and validates the correctness of the results, including checking the endorsement policy.

4.2.4 Access control and Privacy

Req. 5. The system must support expressive access control based on identity so operations on, and views of, data in the network can be restricted.

Composer includes a domain-specific access control language (ACL) "that provides declarative access control over the elements of the domain model".[21]

Fabric supports encapsulation of private data between participants through the use of channels.

> Private **channels** are restricted messaging paths that can be used to provide transaction privacy and confidentiality for specific subsets of network members. All data, including transaction, member and channel information, on a channel are invisible and inaccessible to any network members not explicitly granted access to that channel.[25]
>
> To further obfuscate the data, values within chaincode can be **encrypted** (in part or in total) using common cryptographic algorithms such as AES before sending transactions to the ordering service and appending blocks to the ledger.[28]

4.2.5 Queries

Req. 2. The record of traceability information must be queryable, allowing users to find the full history of an asset.

Composer includes an SQL-like query language that "defines queries to run and return data from business networks".[22]

4.2.6 Integration

Req. 8. Adoption of the system should be possible in an incremental manner, i.e. the system must present an interface enabling it to integrate with existing systems.
Composer provides a tool called *composer-rest-server*, which generates a REST API from the business network. This API provides a standardised interface for client applications to interface with the network.

4.3 Supply chain network model

The supply chain network model used in this project was developed from a selection of models from literature. As stated in section 2.2.4, Chen's [8] implementation (see Figure 2.1) is too generic to be useful in the context of this project.

The wine supply chain described in Biswas et al. [5], however, is sufficiently complex and detailed to be an ideal candidate for implementation, following *Req. 8.*. See Figure 2.2 for a visualisation of the full network model used by Biswas.

The addition of more than two distribution channels to the network model does not add value to the work of this project, so the design decision was taken to implement a simplified version of the model described by Biswas et al., shown in Figure 4.2.

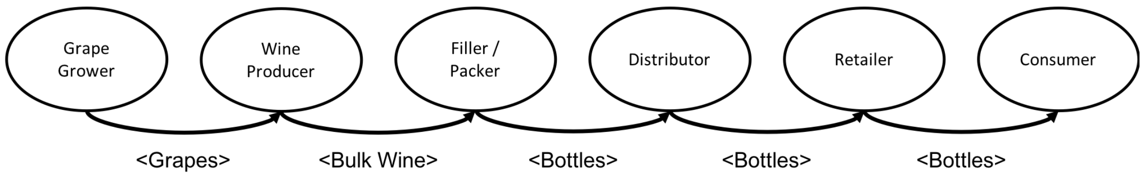


Figure 4.2: The Supply Chain Network Model used in this project

4.4 Methodology

For this project, a mix of agile and incremental methods of software design and development have been used, because of the authors unfamiliarity with the Hyperledger software platform. A Minimum Viable Product (MVP) was created, that included a subsection of the following supply chain network model with limited features. This allowed the author to gain familiarity and confidence with the platform. Afterwards, the completeness of both the network model and the feature set of the system was incrementally increased.

Since Hyperledger Fabric only supports object-based databases [27], designing a relational data model would not be suitable, so Entity Relationship Diagrams [9] have been created to describe the system.

4.5 Identification of MVP

For a Minimum Viable Product the supply chain network for the first two participants in the chain - "Grape Grower" and "Wine Producer" - was implemented. Two is naturally the smallest number necessary to perform transactions between multiple types of participant.

The MVP should implement basic data models for these participants, including the transactions between the two, and should run on Hyperledger Fabric.

See Figure 4.3 for the subsection of the network model implemented for MVP.

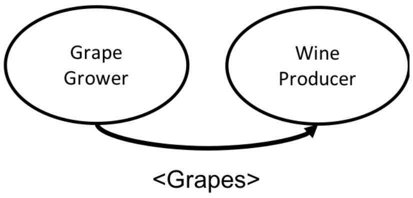


Figure 4.3: MVP Supply Chain Network Model

Biswas et al. [5] identify potential data to be stored at each step in the supply chain:

Grape Growers should store information on "location, altitude, types of vines, origin, irrigation, treatments, and pruning or purging date".
Wine Producers need to keep "suppliers' details, received date, description of the received products, variety of grapes, chemical contents, distributor records, and additives (if any)".

See Figure 4.4 for an Entity Relationship Diagram (ERD) of this subset of the system.

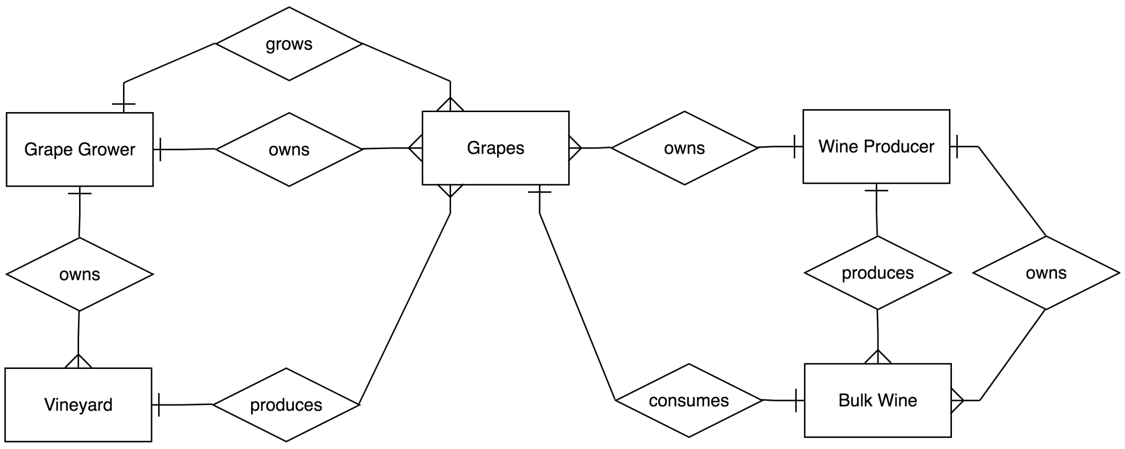


Figure 4.4: Entity Relationship Diagram for MVP

4.6 Creating the remainder of the system

After completing the development of the MVP, the remainder of the system was developed as follows:

4.6.1 Network model

The complete system will require the addition of the following, from the complete network model in Figure 4.2.

- The remainder of the participants
- The transactions between them

Adding new participants will also mean new entities need to be added in Fabric.

4.6.2 Data model

As described above for MVP, the potential data to be stored at each step is identified by Biswas et al. [5].
However, in the full system, many entities share attributes:

- Participants (Grape Grower, Retailer, etc.) are all *Actors*, having the same data attributes like Company Name.
- Assets like Grapes or Wine Bottles (*Batches*) share similar attributes like Quantity and Owner.
- Transactions always need to record information about the Batches they transform for traceability purposes.

An efficient data model will abstract these attributes into a superclass, taking inspiration from the ontology driven design described by Kim and Laskowski [36], shown in Figure 4.5.

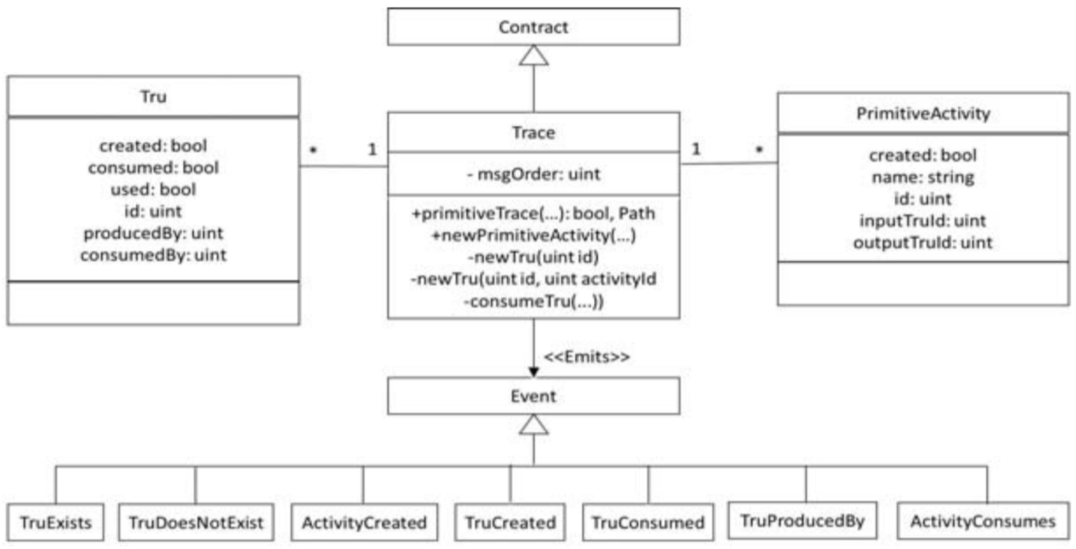


Figure 4.5: Generic data model from Kim and Laskowski

Chapter 5

Implementation

5.1 Mapping design to components

Now the designed components must be mapped into actual software.

- Network models map to nodes in the network, which are Docker containers in Fabric.
- ERDs map to data models in Composer.

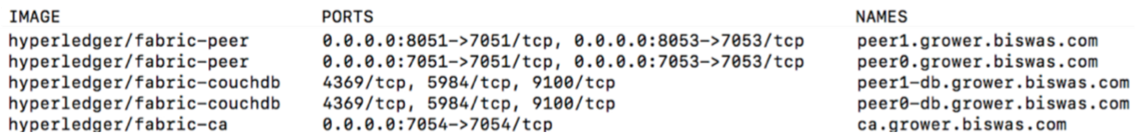
5.2 Translating the network model into a Fabric network

5.2.1 Docker

Req. 7. The system should be able to run on multiple hardware platforms, ideally on common or easily accessible platforms such as GCP or AWS.
Fabric provides Docker container images for each type of node in the network.

> A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Docker containers are based on open standards and run on all major Linux distributions, Microsoft Windows, and on any infrastructure including virtual machines, bare-metal, and in the cloud.[54]

See Figure 5.1 for an example of the docker containers created for a single organisation.



```
IMAGE                         PORTS                                            NAMES
hyperledger/fabric-peer       0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp   peer1.grower.biswas.com
hyperledger/fabric-peer       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp   peer0.grower.biswas.com
hyperledger/fabric-couchdb    4369/tcp, 5984/tcp, 9100/tcp                     peer1-db.grower.biswas.com
hyperledger/fabric-couchdb    4369/tcp, 5984/tcp, 9100/tcp                     peer0-db.grower.biswas.com
hyperledger/fabric-ca         0.0.0.0:7054->7054/tcp                           ca.grower.biswas.com
```

Figure 5.1: Docker containers for a single participant running on a single machine

5.2.2 Modelling

Each participant in the supply chain network (see Figure 4.2) was modelled as a single organisation having two peer nodes, backed by persistent databases. Each organisation also has a CA. See Figure 5.1 for the container listing for a single organisation.

The containers were run on a single machine during development, but could be deployed to any cloud hosting provider. On a single machine the container ports are remapped to simulate different web addresses, whereas in the real world they would be referred to by a URL or IPV4 address.

The setup and configuration of these containers is specified in a *Docker Compose* file, similar to the partial example in Figure 5.2.2. The Docker Compose files are located in the `network/fabric/docker` directory.

To enable peer-to-peer communication, the participants also need to know each other's web addresses, which is specified in the connection profiles. Part of a connection profile is shown in Figure 5.2.2. The connection profiles are located in the `network/fabric/connection-profiles` directory.

5.2.3 Adding more participants

Req. 12. The system must have the ability to add participants to the network while it is running.
In a real world use case there would be multiple organisations of each participant type (i.e. multiple Grape Growers) taking part in the system - from the organisations perspective this would simply require creating some containers and installing the necessary system. A network administrator would then add the new organisation to the channel. This process is described in the Fabric Documentation.

5.3 Translating the ERD into Composer data models

Participants in the supply chain network model have been modelled as channel members in Fabric, with sufficient infrastructure to participate in the network. Now the same participants must be modelled as entities in the business logic in Composer, along with the assets shown in the ERD in Figure 4.4.

5.3.1 Assets

While the ERD shown in Figure 4.4 is a subset of the full system, models were created for the following assets:

- Grapes
- Bulk Wine
- Bottled Wine (many bottles of wine before labelling)
- Wine Bottle (a single bottle)

See Figure 5.3.1 for an example data model for the asset Grapes. For the remainder of the asset models see the `network/composer/models` directory.

5.3.2 Code reuse through abstraction

Section 4.6.2 described the use of a generic data model to abstract some shared attributes of these assets into a superclass.
In Figure 5.3.2, the assets Grapes *extends* an abstract model called Batch, shown in Figure 5.3.2.
Participant models extend the *Actor* abstract model.
An abstract model cannot be instantiated, but allows other models to inherit it's properties, avoiding the need to specify shared properties more than once.

5.3.3 Transaction models

Transactions within the network also require models to be created, specifying the data needed to submit the transaction. Here again abstraction was used to avoid code duplication. Figure 5.3.3 shows generic transaction models for

- transforming a batch (for example, producing Bulk Wine from Grapes)
- transferring ownership of a batch

5.3.4 Modelling language

These model are in a domain-specific language (DSL), allowing data models to be specified very quickly. This is one of the advantages of using Composer over developing directly on Fabric.

5.4 Implementing business logic through transaction functions

Now that the infrastructure for the network to run on has been implemented, and the data models created for all the assets and participants, the business logic of the system must be implemented, i.e. the code tied to each transaction. These are smart contracts, or as they are known in the Fabric ecosystem, *chaincode*.
Fabric chaincode can be written in GoLang or Node.js, however Composer only supports JavaScript due to the interaction with the transaction models defined above.[23]
Figure 5.7 shows an example transaction function for transforming batches (e.g. producing Bulk Wine from Grapes). Some variable assignments have been omitted for concision. Transaction functions are located in the `network/composer/lib` directory.

5.4.1 Input and Decorators

The *transformBatch* function takes an argument which is the transaction data specified in the transaction model in Figure 5.3.3. The decorator information in the comment above the function is necessary and specified which transaction model the function is for.

5.4.2 Input validation

The implementation of more powerful access control is described later in the paper, but lines 9 - 12 show a basic level of input validation, checking whether the participant submitting the transaction actually owns the batch they are attempting to transform.

5.4.3 Functionality

This transaction produces a batch of a new asset type, consuming the previous one in the process.

- Adding a new batch is performed on lines 18 - 33.
- The old batch is consumed on lines 38 - 41.

5.4.4 Events

On lines 43 - 58 an event is emitted. Participants can subscribe to these events, for example to be notified when a certain bottle of wine is sold. A log of events is also kept in the system, allowing them to be queried for traceability purposes.

5.4.5 Atomic

Transaction functions in Hyperledger are atomic, meaning that all changes made in the transaction will complete, or none of them. If an error occurs after only part of the transaction logic has been executed, all changes made will be rolled back.

5.5 Access control

As described in the Design section 4.2.4, Composer uses another DSL to describe access control rules for the business network. These are declarative and allow for granular control over visibility of assets and execution of transactions. See Figure 5.5 for an example of some access control rules in the system. See the file `network/composer/permissions.acl` for the full listing.
As shown on line 14 of Figure 5.5, access control rules can be conditional upon a JavaScript expression, this particular rule mandating that participants can update batches if and only if they own them.

5.6 Queries

As described in section 4.2.5, Composer implements another DSL providing SQL-like query functionality against the ledger. These queries can be used by clients of the system for traceability purposes or as part of transaction functions. Figure 5.6 shows some example queries. See `network/composer/queries.qry` for more examples.

5.7 Consensus

As described in section 4.2.3, Fabric provides a pluggable consensus mechanism, which is needed so participants can agree on the state of the ledger, as specified in *Req. 6.*.

5.7.1 Endorsement

The first stage, endorsement, is specified as the endorsement policy in Composer. This specifies which participants' signatures are required for a valid transaction to be submitted. See Figure 5.7.1 for the endorsement policy for a transaction between a distributor and a retailer. This policy mandates the transaction must be signed by a peer from each participant to be considered valid.

5.7.2 Ordering

The second stage, ordering, is performed by the ordering node(s) that are owned by the consortium. Fabric v1.1 offers two types of orderer setup:

- Solo - a single orderer node for development purposes
- Kafka - a multi-node setup utilising Apache Kafka to share information between orderers.

In this project the Kafka setup has been used, specifying two orderer nodes. Although the Kafka-based consensus algorithm is not Byzantine Fault Tolerant, adding more nodes would increase the system's tolerance to crash faults or network errors.[20]

5.7.3 Validation

After endorsement and validation, transactions are automatically validated by peers by checking the state of the ledger. This ensures the relevant items in the ledger have not been updated since endorsement, avoiding the issue of double spending.[20]

5.8 Interface

Once the system is running, participants want to interface with the network to submit transactions and then to query traceability information.

5.8.1 API

Composer provides a tool to generate a REST API from the data models defined above.
Authentication was added to the API using GitHub OAuth - this allows a user to login to the API, then import their network certificate (issued by the CA), and interact with the network.
See Figure 5.11 for a screenshot of the API explorer, showing the endpoints created for each asset and transaction. A client can then interact with this API using regular HTTP requests, for example from the command line or a web application.

5.8.2 Traceability

Lastly, a small web application in React.js has been built to demonstrate consuming the REST API for traceability purposes, located in the `frontend` directory. See Figure 5.12 for a screen capture of this application.
A consumer inputs the bottle identifier (found on the bottle label) into the search box, then the application makes some HTTP requests against the network (including the queries described in section 5.6) and displays information about the origins of the bottle.

```yaml
services:
  ca:
    container_name: ca.grower.biswas.com
    extends:
      file: base.yaml
      service: ca-base
    environment:
      - FABRIC_CA_SERVER_CA_NAME=ca-grower
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile ...' (truncated)
    volumes:
      - ../artifacts/certs/peerOrganizations/grower.biswas.com/ca:/etc/hyperledger

  peer1-db:
    container_name: peer1-db.grower.biswas.com
    image: hyperledger/fabric-couchdb
    environment:
      - COUCHDB_USER=growerpeer1admin
      - COUCHDB_PASSWORD=pass

  peer1:
    container_name: peer1.grower.biswas.com
    extends:
      file: base.yaml
      service: peer-base
    environment:
      - CORE_PEER_ID=peer1.grower.biswas.com
      - CORE_PEER_ADDRESS=peer1.grower.biswas.com:7051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.grower.biswas.com:7051
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.grower.biswas.com:7051
      - CORE_PEER_LOCALMSPID=GrowerMSP
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=peer1-db.grower.biswas.com:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=growerpeer1admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=pass
    volumes:
        - /var/run/:/host/var/run/
        - ../artifacts/certs/peerOrganizations/... (truncated)
    ports:
      - 8051:7051
      - 8053:7053
```

Figure 5.2: A partial example of a Docker Compose file

```
"organizations": {
  "Grower": {
    "mspid": "GrowerMSP",
    "peers": ["peer0.grower.biswas.com", "peer1.grower.biswas.com"],
    "certificateAuthorities": ["ca.grower.biswas.com"]
  },
  "Producer": {
    "mspid": "ProducerMSP",
    "peers": ["peer0.producer.biswas.com", "peer1.producer.biswas.com"],
    "certificateAuthorities": ["ca.producer.biswas.com"]
  },
  "Filler": {
    "mspid": "FillerMSP",
    "peers": ["peer0.filler.biswas.com", "peer1.filler.biswas.com"],
    "certificateAuthorities": ["ca.filler.biswas.com"]
  },
  "Distributor": {
    "mspid": "DistributorMSP",
    "peers": ["peer0.distributor.biswas.com", "peer1.distributor.biswas.com"],
    "certificateAuthorities": ["ca.distributor.biswas.com"]
  },
  "Retailer": {
    "mspid": "RetailerMSP",
    "peers": ["peer0.retailer.biswas.com", "peer1.retailer.biswas.com"],
    "certificateAuthorities": ["ca.retailer.biswas.com"]
  }
}
```

Figure 5.3: Part of a connection profile

```
asset Grapes extends Batch {
  o String species
  o DateTime harvestDate
  --> Vineyard vineyard
  --> GrapeGrower grapeGrower
}

asset Vineyard identified by vineyardId {
  o String vineyardId
  o String region
  o Location location
  o Integer altitude  // metres above sea level
}
concept Location {
  o Double latitude
  o Double longitude
}
```

Figure 5.4: A partial example of a Composer data model

```
abstract participant Actor identified by name {
  o String name
  o String email
}
abstract asset Batch identified by batchId {
  o String batchId
  o Integer quantity
  --> Actor owner
}
```

Figure 5.5: Abstract composer data models

```
transaction sellBatch {
  o Integer quantity
  --> Batch batch
  --> Actor buyer
}

transaction transformBatch {
  --> Batch batch
}
```

Figure 5.6: Composer transaction models

Figure 5.7: An example transaction function forming a smart contract

```
/**
 * Transform a batch
 * @param {biswas.base.transformBatch} tx
 * @transaction
 */
function transformBatch(tx) {
  //variable assignments omitted

  var submitter = getCurrentParticipant();
  if (oldBatch.owner.$identifier !== submitter.$identifier) {
    throw new Error('You do not own that batch');
  }

  return getAssetRegistry(newFQBatchName)
    .then(function(newBatchRegistry) {
      // create a new batch for the new owner
      var id = newBatchDetails.name + '_' + Date.now();
      newBatch = factory.newResource(
        newBatchDetails.namespace,
        newBatchDetails.name,
        id
      );

      newBatch.quantity = parseInt(
        oldBatch.quantity * newBatchDetails.scaleFactor
      );
      newBatch.owner = factory.newRelationship(
        submitter.$namespace,
        submitter.$type,
        submitter.$identifier
      );
      assignBatchProperties(oldBatch, newBatch, factory);
      return newBatchRegistry.add(newBatch);
    })
    .then(function() {
      return getAssetRegistry(oldFQBatchName);
    })
    .then(function(oldBatchRegistry) {
      // Consume the original batch
      oldBatch.quantity = 0;
      return oldBatchRegistry.update(oldBatch);
    })
    .then(function() {
      // emit an event
      var event = factory.newEvent('biswas.base', 'BatchTransformed');
      event.batchTypeConsumed = oldBatchName;
      event.oldBatch = factory.newRelationship(
        oldBatchNamespace,
        oldBatchName,
        oldBatch.$identifier
      );
      event.batchTypeCreated = newBatchDetails.name;
      event.newBatch = factory.newRelationship(
        newBatchDetails.namespace,
        newBatchDetails.name,
        newBatch.$identifier
      );
      emit(event);
    });
}
```

```
rule FillerCreate {
  description: "Allow fillers to create assets in their own namespace"
  participant: "biswas.filler.Filler"
  operation: CREATE
  resource: "biswas.filler.*"
  action: ALLOW
}

rule UpdateOwnBatches {
  description: "Allow participants to update assets they own"
  participant(p): "org.hyperledger.composer.system.Participant"
  operation: UPDATE
  resource(r): "biswas.base.Batch"
  condition: (p.getIdentifier() == r.owner.getIdentifier())
  action: ALLOW
}
```

Figure 5.8: Access control rules

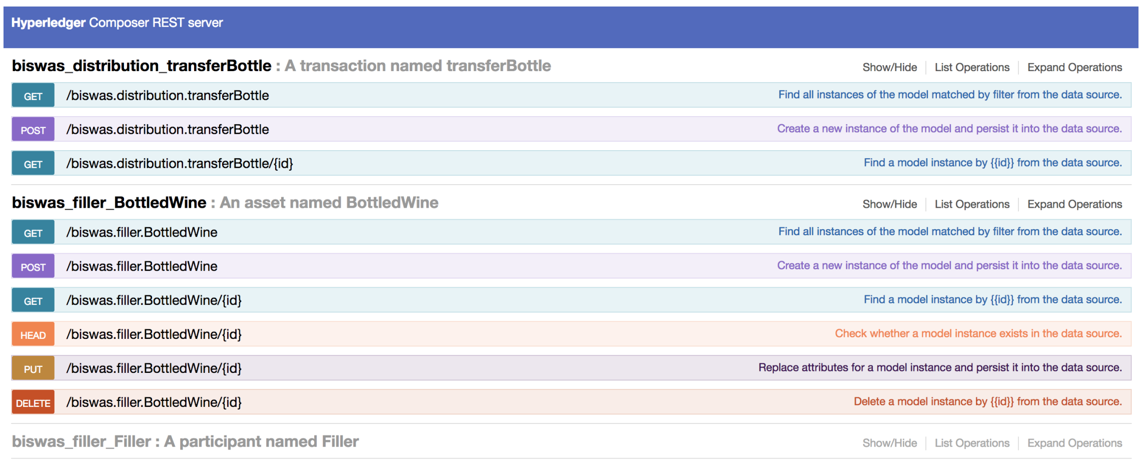
```
query previousOwners{
  description: "Find all the previous owners of a wine bottle"
  statement:
    SELECT org.hyperledger.composer.system.HistorianRecord
      WHERE (transactionType == 'biswas.distribution.transferBottle')
}

query getBatches{
  description: "Get all the batches owned by a participant"
  statement:
    SELECT biswas.base.Batch
      WHERE (owner == _$participantID)
}
```

Figure 5.9: Composer Traceability Queries

```
{
    "identities": [
        {
            "role": {
                "name": "member",
                "mspId": "DistributorMSP"
            }
        },
        {
            "role": {
                "name": "member",
                "mspId": "RetailerMSP"
            }
        }
    ],
    "policy": {
        "2-of": [
            {
                "signed-by": 0
            },
            {
                "signed-by": 1
            }
        ]
    }
}
```

Figure 5.10: Endorsement Policy



Hyperledger Composer REST server

biswas_distribution_transferBottle : A transaction named transferBottle — Show/Hide | List Operations | Expand Operations

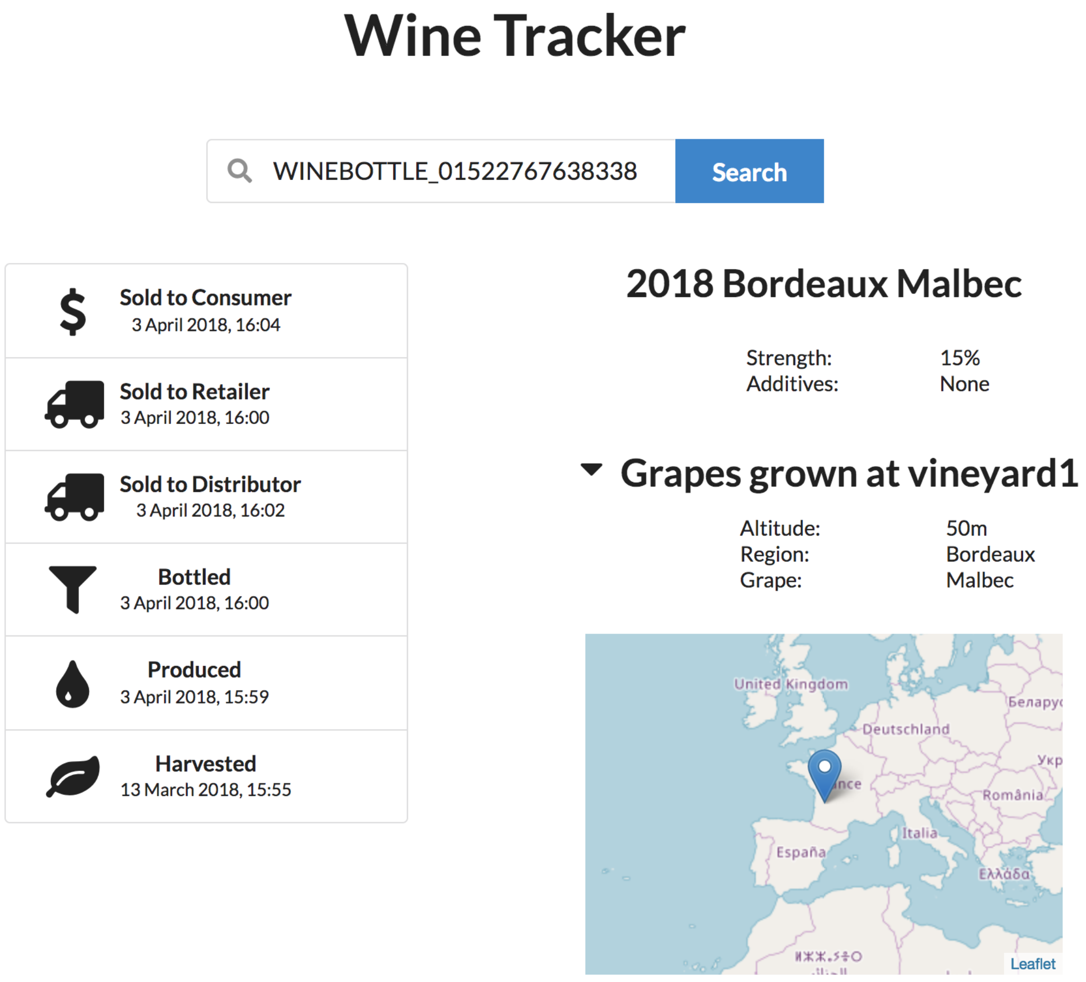
GET	/biswas.distribution.transferBottle	Find all instances of the model matched by filter from the data source.
POST	/biswas.distribution.transferBottle	Create a new instance of the model and persist it into the data source.
GET	/biswas.distribution.transferBottle/{id}	Find a model instance by {{id}} from the data source.

biswas_filler_BottledWine : An asset named BottledWine — Show/Hide | List Operations | Expand Operations

GET	/biswas.filler.BottledWine	Find all instances of the model matched by filter from the data source.
POST	/biswas.filler.BottledWine	Create a new instance of the model and persist it into the data source.
GET	/biswas.filler.BottledWine/{id}	Find a model instance by {{id}} from the data source.
HEAD	/biswas.filler.BottledWine/{id}	Check whether a model instance exists in the data source.
PUT	/biswas.filler.BottledWine/{id}	Replace attributes for a model instance and persist it into the data source.
DELETE	/biswas.filler.BottledWine/{id}	Delete a model instance by {{id}} from the data source.

biswas_filler_Filler : A participant named Filler — Show/Hide | List Operations | Expand Operations

Figure 5.11: API Explorer



Wine Tracker

WINEBOTTLE_01522767638338

Search

Sold to Consumer
3 April 2018, 16:04

Sold to Retailer
3 April 2018, 16:00

Sold to Distributor
3 April 2018, 16:02

Bottled
3 April 2018, 16:00

Produced
3 April 2018, 15:59

Harvested
13 March 2018, 15:55

2018 Bordeaux Malbec

Strength:	15%
Additives:	None

Grapes grown at vineyard1

Altitude:	50m
Region:	Bordeaux
Grape:	Malbec

Figure 5.12: Web application displaying traceability information

Chapter 6

Testing

Martin states in his book on Agile software development “comprehensive tests have a profoundly positive effect upon the structure of the software”.[41] The following section describes how the system has been tested.

6.1 Unit tests

6.1.1 TDD

Writing tests before functionality has been shown to increase programmer productivity and software quality.[11] Therefore, whilst developing the business logic in Composer, a test driven development (TDD) methodology was followed.

6.1.2 Tests as documentation

Unit testing is important not only to verify the behaviour of the system but also to document it.

Req. 9. Developers within an organisation should find it straightforward to generalise the software implementation given in this paper when creating a system tailored to their use case.

Martin [41] also states that “The tests act as a suite of examples that help other programmers figure out how to work with the code ... if you want to know how to call a function or create an object, there is a test that shows you”.
Using the testing framework Mocha along with the Chai assertion library allowed the expected test results to be specified using natural, English-like vocabulary, shown in Figure 6.1.2. This fulfils *Req. 9.*.

```
describe('labelBottles()', () => {
  it('should create the correct number of bottles', async () => {
    // ... omitted ...
    wineBottles.length.should.equal(constants.bottledWineQuantity);
  });
  it('should specify the correct owner for the new bottles', async () => {
    // ... omitted ...
    wineBottles[0].owner.$identifier.should.equal(constants.fillerName);
  });
  it('should consume the BottledWine', async () => {
    // ... omitted ...
    bottledWine.quantity.should.equal(0);
  });
});
```

Figure 6.1: Natural language in tests using Mocha and Chai

6.1.3 Running tests

Martin states “The simpler it is to run a suite of tests, the more often those tests will be run” [41], therefore gaining as much of the benefits of testing as possible.

Tests for this system are run using one simple command - `npm t`.
A git pre-commit hook was also added, ensuring code could not be committed to git if the unit tests did not pass, which further enhanced code quality.

6.1.4 Unit tests

Composer includes libraries for Node which simulate the Fabric infrastructure, allowing tests to run quickly for fast, iterative development.
Unit tests enabled testing the behaviour of:

- Transaction functions
- Access control rules
- Queries

All the transaction functions were tested thoroughly, making sure to include tests for the expected exception paths. The adding and updating of assets was also tested using different identities to ensure the access control rules behaved as expected.
An example of an exception path would be expecting an exception to occur if a participant attempts to update an asset they do not own.
Finally, the traceability capabilities of the system were tested through testing the queries.

6.2 Integration tests

Unit tests ensure the business logic behaves as expected, but integration testing was needed to have confidence the system works when deployed to an actual instance of Fabric.

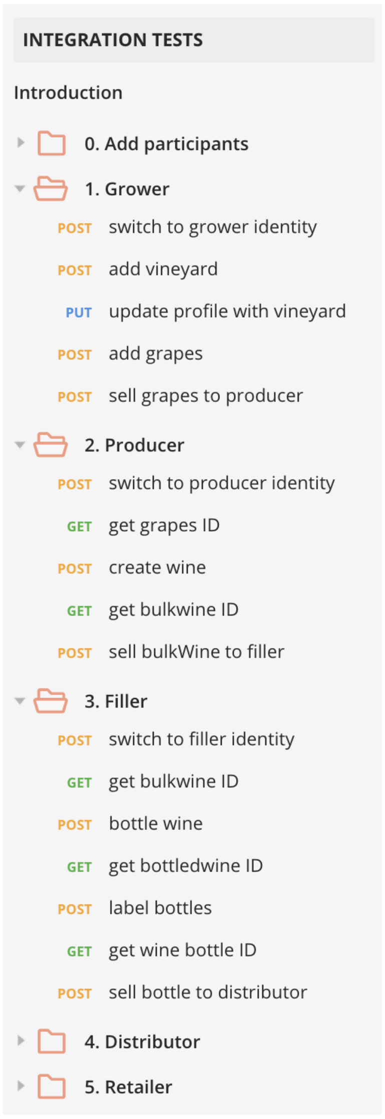
6.2.1 Scripts

To streamline the process of redeploying a Fabric instance, shell scripts were created that, for example, removed all the running docker containers and recreated them.
The full deploy script performed the following actions:

- Spin up a fresh instance of Fabric
- Create and instantiate the Fabric channel
- Join participants to the channel
- Compile the Composer business network
- Install and instantiate the business network

6.2.2 Interacting with the network

To interact with the network, the system uses the *composer-rest-server* tool to generate an API and *Postman* to submit a series of API requests.
Postman enabled persistence groups of requests, ensuring a consistent test run each time.
Integration tests as shown in Figure 6.2 and also located online simulated the entire supply chain, from creating Grapes as a Grape Grower to selling a bottle of wine to the Consumer as a Retailer.
The multi-user mode of the REST server described in section 5.8.1 allowed the use of multiple blockchain identities, ensuring *Req. 4.* was fulfilled.


INTEGRATION TESTS

Introduction

0. Add participants

1. Grower
- POST switch to grower identity
- POST add vineyard
- PUT update profile with vineyard
- POST add grapes
- POST sell grapes to producer

2. Producer
- POST switch to producer identity
- GET get grapes ID
- POST create wine
- GET get bulkwine ID
- POST sell bulkWine to filler

3. Filler
- POST switch to filler identity
- GET get bulkwine ID
- POST bottle wine
- GET get bottledwine ID
- POST label bottles
- GET get wine bottle ID
- POST sell bottle to distributor

4. Distributor

5. Retailer

Figure 6.2: Integration test run in Postman

6.3 Traceability

Unit tests ensured the business logic was correct, and the end-to-end integration tests ensured the system works as expected when deployed. The web application described in section 5.8.2 also enabled testing of the traceability capabilities of the system.
Unit testing the queries ensured the logic behaved as expected, but the frontend allowed simulation of a real use case for the fulfilment of requirement *Req. 2.*.
After adding sample data using Postman, the user is able to input a Wine Bottle identifier into the frontend and trace the asset all the way back to the Grapes it was created from, including information about the vineyard it was grown in.

Chapter 7

Discussion

7.1 Software evaluation

In section 3.1, following review of literature the aim of the project was defined as follows:

> Companies need a technical framework for creating blockchain supply chain systems, and a process to follow for integrating blockchain into their supply chain.
>
> An implementation of a sufficiently complex supply chain model could serve as a proof of concept and a useful point of reference for developers creating similar systems within their organisation.

7.1.1 Complexity

Hyperledger Composer provide some sample networks on GitHub [10]. These provide code samples, for example for unit tests, however again are quite simplistic, with a low number of participants. In addition, these sample networks are missing access control rules, and do not come with the accompanying Fabric network definition so these networks could not be deployed in a production environment.
The software system created as part of this project achieves the aim of implementing a sufficiently complex model as detailed below. In comparison to Chen's [8] generic implementation, and the basic examples provided by Composer [10], this project's implementation has a higher complexity across a number of factors as shown in the table below. The 'perishable' Composer sample network is shown as it is most similar to the supply chain use case implemented in this paper.

Network	Participants	Assets	Access control rules	Traceability queries
This paper's system	6	5	12	Yes
Chen	5	1 (generic)	15	No
perishable-network	3	1 (generic)	No	No

7.1.2 Generalisation

Req. 9. Developers within an organisation should find it straightforward to generalise the software implementation given in this paper when creating a system tailored to their use case. I believe a more complex system is closer to a real world supply chain implementation, and could be generalised more easily.
Another factor affecting the ability to use the system as a base for implementation is the level of documentation. One way in which my system achieves this is the comprehensive unit testing suite written in natural language as described in section 6.1.2.
Another advantage of this paper's system over Chen and the Composer sample networks is the comprehensive of the development process here, detailing how the supply chain network model was translated into Fabric infrastructure, and the ERD translated into Composer data models.
Finally, to the authors knowledge, the system developed in this project is the only one to create both the business logic in Composer and the network architecture in Fabric, delivering an end-to-end production ready system.

7.1.3 Integration

The final part of the hypothesis to be evaluated relates to *Req. 8.* Adoption of the system should be possible in an incremental manner, i.e. the system must present an interface enabling it to integrate with existing systems.. Neither Chen nor the Composer sample networks detail how their systems interface with the real world, or the process of adoption.
The system described in this paper implements a standardised REST interface, with multi-user authentication through the combination of OAuth and the certificates issued by a Fabric CA, also fulfilling *Req. 4.* The system must allow credentials for participation in the network to be issued and revoked..

The integration test mentioned in section 6.2.2 proves this paper's system supports a sufficient interface for all participants in the supply chain to interact with the network.
The web application described in section 5.8.2 shows that it is possible, using this API, to integrate traceability into consumer applications as well as existing traceability systems.

7.2 Traceability

In the literature review, studies on traceability were also looked at, and the key attributes of an effective traceability system discovered. Here this paper's system is evaluated against those criteria.

The key traceability requirements are *Req. 1.* Record data every time a TRU is sold or converted into another TRU. and *Req. 2.* The record of traceability information must be queryable, allowing users to find the full history of an asset..
The system stores data every time a transaction is submitted. Inside the transaction functions events are emitted containing more information about the transaction, as described in section 5.4.4. This storage of data fulfils *Req. 1.*, and combining this stored data with the querying functionality described in section 5.6 fulfils *Req. 2.*.
Finally, as described in the literature review, the system must enable interoperability between supply chain participants, which is specified in *Req. 8.* and fulfilled by the API described in section 5.8.1.

7.3 Project evaluation

7.3.1 Literature review

At the beginning of the project the initial scope was the creation of a generic traceability system similar to Chen's [8], but after reviewing similar work the author realised that use case had been well covered by previous papers. It became apparent that implementing a more specific, more complex supply chain would be more useful, so the project aim pivoted to the current system.
Another issue with the literature review was the breadth of reading. A number of papers on data provenance were studied, unnecessarily, since data provenance techniques and the issues with data provenance are actually quite different from those within a physical supply chain.
On the other hand, it was very useful to read some traceability literature rather than purely reviewing blockchain supply chain systems. Knowledge of general traceability enabled synthesis of the attributes of a successful traceability system from first principles, which has resulted in a system that can be effectively generalised to a number of different use cases.

7.3.2 Requirements Capture

During the requirements capture phase, this wider traceability reading enabled more business-focused requirements to be captured, prompting considerations about the system from a users perspective rather than focusing purely on the technology.
However, it would have increased the confidence in these requirements if some subject matter experts within the traceability field had been interviewed. The author attempted to reach out to some procurement employees at Eli Lilly but unfortunately they did not respond.

7.3.3 Design

Choice of technology

The choice of technology platform was a key part of the design section, with the final decision being Hyperledger. There were some unforeseen difficulties with using this platform as a result of it being designated as alpha release software at the beginning of the project.
The incompleteness of the platform meant the implementation of some system components were harder than necessary, especially where the documentation was incorrect or incomplete. This pushed back the timeline for completion of the system, meaning only one supply chain network model was implemented.
On the other hand, using Composer was a good choice as it allowed the author to use a familiar development ecosystem (JavaScript), and enabled fast, iterative test-driven development with the provided testing frameworks. The higher level of abstraction provided meant less boilerplate code, for example when creating new participants when implementing the full system.
Being required to use multiple domain-specific languages did increase the learning curve for Composer, but this disadvantage may be counteracted by implementing the MVP first, and outweighed by the speed of development once familiarity with the technology was gained.

Choice of network model

The other key decision point while designing the system was the choice of network model to implement. Using a model from literature [5], rather than creating one from scratch increases confidence in the applicability of the system in the real world, with sufficient complexity to generalise to multiple supply chain use cases.

7.3.4 Implementation

Methodology

During development an agile methodology was followed, first implementing a Minimum Viable Product, then incrementally adding features and functionality.
One disadvantage of this approach was that some of the MVP code was not used in the final system, or heavily refactored, for example when abstracting common asset attributes into a superclass as described in section 4.6.2. This resulted in wasted development time.
It could be argued that this MVP approach is more suitable for a business focused setting, where fast feedback from the client is important. An incremental development approach could have been used for the entirety of the project, which may have avoided writing as much redundant code, and therefore as much development time wasted. However, in the authors opinion, gaining familiarity with the Hyperledger ecosystem during the creation of the MVP outweighs this disadvantage, especially in the light of the immaturity of the software platform.
The MVP approach worked well for creating the application layer of the system in Composer, but there were some difficulties with deploying it to Fabric. If the project were to be completed again, perhaps a better choice would be to implement a smaller MVP or proof-of-concept using Fabric first, with only a single participant instead of the two implemented in this project.

Hyperledger

As described above, the immaturity of the Hyperledger platform implied a number of disadvantages. Fortunately, partway through the project, Fabric v1.1 and Composer v0.16 were released. The system was upgraded to take advantage of the numerous bugfixes and features implemented in these releases, which increased the reliability of the system.

7.3.5 Testing

As described above, the Composer testing ecosystem was very helpful for catching bugs in transaction logic, but the lack of testing options for Fabric infrastructure was a hindrance. Integration tests were only possible once the system was fully deployed, which meant they were only useful later in the project, after the MVP was completed.

Integrating the system with an existing traceability system would have a been a valuable addition to the testing section, to prove it works in the real world. However, the REST API successfully demonstrates this integration, as shown by consuming the API through the web application described in section 5.8.2.

7.3.6 Further work

As described above, difficulties with the technology led to project timelines being pushed back. With more time another supply chain model could be implemented using the same technology stack. Implementing multiple network models would improve the generalisation of the system to a wider variety of supply chain use cases.

Another feature which could be added to the project would be a deployment process onto a common infrastructure platform such as Amazon Web Services (AWS) or Microsoft Azure. This would allow developers to run the software with limited local hardware, and again provide a more complete, enterprise-ready system.

Finally, the same supply chain network could be implemented on a different technology stack such as MultiChain, allowing the two platforms to be compared, deriving some knowledge of the advantages of each platform for certain use cases.

Chapter 8

Conclusion

At the beginning of the project the elements of a successful traceability system were identified, along with a sufficiently complex supply chain network model from literature that could be generalised well. Then, a selection of permissioned blockchain platforms were evaluated, identifying the platforms that provide the necessary features to support such a supply chain system.

The identified supply chain model was then implemented on Hyperledger using an agile development methodology. Through this implementation, it was shown that a blockchain system can successfully improve traceability within the supply chain.

The project delivers a useful reference implementation of a supply chain on Hyperledger, which could be used as a basis for implementing such a system in enterprise.

Bibliography

[1] Abeyratne, S. A. and Monfared, R. P. [2016], 'Blockchain ready manufacturing supply chain using distributed ledger', *IJRET: International Journal of Research in Engineering and Technology* .

[2] Aung, M. M. and Chang, Y. S. [2014], 'Traceability in a food supply chain: Safety and quality perspectives', *Food Control* **39**, 172–184.

[3] Bauerle, N. [2017], 'What Are the Applications and Use Cases of Blockchains?', https://www.coindesk.com/information/applications-use-cases-blockchains/.

[4] Bechini, A., Cimino, M. G., Marcelloni, F. and Tomasi, A. [2008], 'Patterns and technologies for enabling supply chain traceability through collaborative e-business', *Information and Software Technology* **50**(4), 342–359.

[5] Biswas, K., Muthukkumarasamy, V. and Lum, W. [2017], *Blockchain Based Wine Supply Chain Traceability System.*

[6] Brown, R., Carlyle, J., Grigg, I. and Hearn, M. [2016], 'Corda: An Introduction'.

[7] Cachin, C., Schubert, S. and Vukolić, M. [2016], 'Non-determinism in Byzantine Fault-Tolerant Replication', *arXiv:1603.07351 [cs]* .

[8] Chen, E. [2016], 'An Approach for Improving Transparency and Traceability of In- dustrial Supply Chain with Blockchain Technology'.

[9] Chen, P. P.-S. [1988], The entity-relationship model—toward a unified view of data, *in* 'Readings in Artificial Intelligence and Databases', Elsevier, pp. 98–111.

[10] *Composer-Sample-Networks: Sample Business Network Definitions for Composer* [2018].

[11] Erdogmus, H., Morisio, M. and Torchiano, M. [2005], 'On the effectiveness of the test-first approach to programming', *IEEE Transactions on Software Engineering* **31**(3), 226–237.

[12] European Commission [2005], 'Commission Regulation (EC) No 1688/2005'.

[13] FDA [2006], 'Guidance for Industry - Prescription Drug Marketing Act (PDMA) Requirements'.

[14] Glavic, B. and Dittrich, K. R. [2007], Data Provenance: A Categorization of Existing Approaches., Vol. 7, pp. 227–241.

[15] Golan, E., Krissoff, B., Kuchler, F., Calvin, L., Nelson, K. and Price, G. [2004], 'Traceability in the US food supply: Economic theory and industry studies', *Agricultural economic report* **830**(3), 1–48.

[16] Greenspan, G. [2015], 'MultiChain Private Blockchain — White Paper', https://www.multichain.com/download/MultiChain-White-Paper.pdf.

[17] Greenspan, G. [2017], 'Where Have All the Private Blockchains Gone?', https://www.coindesk.com/private-blockchains-gone/.

[18] Hancock, M. and Vaizey, E. [2016], 'Distributed Ledger Technology: Beyond block chain', https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

[19] Hearn, M. [2016], 'Corda: A distributed ledger', *Corda Technical White Paper* .

[20] *Hyperledger Architecture, Volume 1* [2017], https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

[21] *Hyperledger Composer Documentation - Access Control Language* [2018], https://hyperledger.github.io/composer/latest/reference/acl_language.html.

[22] *Hyperledger Composer Documentation - Reference* [2018], https://hyperledger.github.io/composer/latest/reference/reference-index.

[23] *Hyperledger Composer Home Page* [2018], https://www.hyperledger.org/projects/composer.

[24] *Hyperledger Fabric Documentation - Certificate Authorities* [2018], http://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html.

[25] *Hyperledger Fabric Documentation - Functionalities* [2018], https://hyperledger-fabric.readthedocs.io/en/release-1.1/functionalities.html#privacy-and-confidentiality.

[26] *Hyperledger Fabric Documentation - Glossary* [2018], https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html.

[27] *Hyperledger Fabric Documentation - Ledger* [2018], https://hyperledger-fabric.readthedocs.io/en/release-1.1/ledger.html#state-database.

[28] *Hyperledger Fabric Documentation - Model* [2018], https://hyperledger-fabric.readthedocs.io/en/release-1.1/fabric_model.html#privacy-through-channels.

[29] *Hyperledger Fabric Home Page* [2018], https://www.hyperledger.org/projects/fabric.

[30] *Hyperledger Home Page* [2018], https://www.hyperledger.org.

[31] Iansiti, M. and Lakhani, K. R. [2017], 'The Truth About Blockchain', https://hbr.org/2017/01/the-truth-about-blockchain.

[32] *ISO 9000:2015(En), Quality Management Systems — Fundamentals and Vocabulary* [2015], https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en:term:3.6.13.

[33] Jansson, F. and Peterson, O. [2017], 'Blockchain Technology in Supply Chain Traceability Systems Developing a Framework for Evaluating the Applicability', http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8918347&fileOId=8919918.

[34] Kaku [2017], 'Using Blockchain to Support Provenance in the Internet of Things', https://ecommons.usask.ca/bitstream/handle/10388/8099/KAKU-THESIS-2017.pdf?sequence=1&isAllowed=y.

[35] Kim, H. M., Fox, M. S. and Grüninger, M. [1999], 'An ontology for quality management—enabling quality problem identification and tracing', *BT Technology Journal* **17**(4), 131–140.

[36] Kim, H. M. and Laskowski, M. [2016], 'Towards an ontology-driven blockchain design for supply chain provenance'.

[37] Kivi, M., Hofhuis, A., Notermans, D. W., Wannet, W. J. B., HECK, M. E. O. C., Van De Geissen, A. W., Van Duynhoven, Y. T. H. P., Stenvers, O. F. J., Bosman, A. and Van Pelt, W. [2007], 'A beef-associated outbreak of Salmonella Typhimurium DT104 in The Netherlands with implications for national and international policy', *Epidemiology and Infection* **135**(6), 890–899.

[38] Lamport, L., Shostak, R. and Pease, M. [1982], 'The Byzantine generals problem', *ACM Transactions on Programming Languages and Systems (TOPLAS)* **4**(3), 382–401.

[39] Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K. and Njilla, L. [2017], ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability, IEEE, pp. 468–477.

[40] Lovis, C. [2008], ‘Traceability in healthcare: Crossing boundaries’, *IMIA Yearbook* pp. 105–113.

[41] Martin, R. C. [2002], *Agile Software Development: Principles, Patterns, and Practices.*

[42] McKean, J. [2001], ‘The importance of traceability for public health and consumer protection’.

[43] Moe, T. [1998], ‘Perspectives on traceability in food manufacture’, *Trends in Food Science & Technology* **9**(5), 211–214.

[44] Murphy, W. G., Katz, L. M. and Flanagan, P. [2013], Regulatory Aspects of Blood Transfusion, *in* M. F. Murphy, D. H. Pamphilon and N. M. Heddle, eds, ‘Practical Transfusion Medicine’, John Wiley & Sons, Ltd, Oxford, UK, pp. 169–180.

[45] Nakamoto, S. [2008], ‘Bitcoin: A Peer-to-Peer Electronic Cash System’, p. 9.

[46] Orcutt, M. [2017], ‘Who Will Build the Health-Care Blockchain?’, *MIT Technology Review* .

[47] Poon, J. and Dryja, T. [2016], ‘The Bitcoin Lightning Network:’, p. 59.

[48] Regattieri, A., Gamberi, M. and Manzini, R. [2007], ‘Traceability of food products: General framework and experimental evidence’, *Journal of Food Engineering* **81**(2), 347–356.

[49] Roberts, J. [2017], ‘The Diamond Industry Is Obsessed With the Blockchain’, http://fortune.com/2017/09/12/diamond-blockchain-everledger/.

[50] Storøy, J., Thakur, M. and Olsen, P. [2013], ‘The TraceFood Framework – Principles and guidelines for implementing traceability in food value chains’, *Journal of Food Engineering* **115**(1), 41–48.

[51] Tan, W. C. [2004], ‘Research problems in data provenance.’, *IEEE Data Eng. Bull.* **27**(4), 45–52.

[52] Thakur, M. and Donnelly, K. A.-M. [2010], ‘Modeling traceability information in soybean value chains’, *Journal of Food Engineering* **99**(1), 98–105.

[53] Thakur, M., Sørensen, C.-F., Bjørnson, F. O., Forås, E. and Hurburgh, C. R. [2011], ‘Managing food traceability information using EPCIS framework’, *Journal of Food Engineering* **103**(4), 417–433.

[54] *What Is a Container* [2017], https://www.docker.com/what-container.

[55] Yiannis, F. [2017], ‘Genius of Things: Blockchain and Food Safety with IBM and Walmart’.

Appendix A

Submission

All software for this project can be found in the Git repository located at:
https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/mxd434.

The submission consists of a `.zip` file with the following folder structure:

- `binaries` - Hyperledger Fabric binary files
- `frontend` - The web application used to consume the API
- `network` - The supply chain network system
  - `composer`
    - `lib` - Transaction functions
    - `models` - Data models
    - `src` - Reusable functions used in tests
    - `test` - Unit tests
    - `endorsement-policy.json` - The transaction endorsement policy
    - `permissions.acl` - Access control rules
    - `queries.qry` - Traceability queries
  - `fabric`
    - `artifacts` - Crytography materials, for example ssh keys
    - `config` - Fabric network configuration
    - `connection-profiles` - Peer connection profiles
    - `docker` - Docker configuration files
    - `id-cards` - More cryptographic materials, for example AES keypairs
    - `*.sh` - Various scripts for managing the Fabric instance
  - `*.sh` - Various other scripts for managing the full system deployment

Appendix B

Running the software

B.1 Prequisites

- Docker
- Postman
- Node.js v9+ / npm / yarn
- a GitHub account

B.2 Install

B.2.1 Download Fabric Docker images

```
  curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0-rc1
```

Fabric documentation

B.2.2 Install Composer command line tools

```
npm i -g composer-cli@0.16.3 composer-rest-server@0.16.3 passport-github
```

B.2.3 Import sample participant identities

```
cd network
./importIdCards.sh
```

B.2.4 Enable authentication for API server

Create a new OAuth application on GitHub

- Application name: `composer-rest-server`
- Homepage URL: `http://localhost:3000/`
- Application description: `OAuth application for the multi-user mode of composer-rest-server`
- Authorization callback URL: `http://localhost:3000/auth/github/callback`

B.3 Run

B.3.1 Start the network

```
# in 'network' directory
export COMPOSE_PROJECT_NAME=biswas
./deployNetwork.sh
```

B.3.2 Start the API server

```
export COMPOSER_PROVIDERS='{
  "github": {
    "provider": "github",
    "module": "passport-github",
    "clientID": "REPLACE_WITH_CLIENT_ID",
    "clientSecret": "REPLACE_WITH_CLIENT_SECRET",
    "authPath": "/auth/github",
    "callbackURL": "/auth/github/callback",
    "successRedirect": "/",
    "failureRedirect": "/"
  }
}'

composer-rest-server -c grower-network-admin@biswas -a true -m true
```

The API explorer is now located http://localhost:3000/explorer.

B.4 Interacting with the network

B.4.1 Import identities into API server

- Navigate to http://localhost:3000/auth/github and authenticate with GitHub
- Find the `/wallet/import` endpoint at the bottom of the list in the API explorer
- Import the admin identity card from `./network/fabric/id-cards/grower-network-admin/card` using the name `admin`
- Import the identity cards for the sample network participants from `./network/fabric/id-cards/users` using the same endpoint. The names should be the same as the name of the card, i.e. the file `distributor.card` should be imported as `distributor`

B.4.2 Add sample data using Postman

- Navigate to the end-to-end test run documentation and click 'Run in Postman'
- Select the imported `sample-env` environment
- Display your OAuth access token in the explorer by clicking 'show'. Copy this into your Postman environment as the value for the key `accessToken`
- Run the requests using the collection runner

B.4.3 View traceability information

- Copy your OAuth access token into line 9 of `./frontend/src/controllers/api.js`
- Start the web application by running `yarn start` inside `./frontend`
- Copy the value of `bottleID` from the Postman environment
- Navigate to the frontend at http://localhost:3001/
- Search for the copied bottleID.

B.5 Development

B.5.1 Unit tests

```
# in ./network/composer
npm t
```