NTNU

Norwegian University of
Science and Technology

Blockchain as an activity tracker to prevent greenwashing

Udnes, Henrik

Submission date: June 2023
Main supervisor: Prof. Gligoroski, Danilo, NTNU
Co-supervisor: Assoc. Prof. Kralevska, Katina, NTNU

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Blockchain as an activity tracker to prevent greenwashing

Student: Udnes, Henrik

Problem description:

In recent years, greenwashing has become ever more widespread among companies that sell their products to consumers. The background for this increase is the shift in focus towards sustainability. Investors and consumers have become more concerned with spending their money on sustainable products and brands that promote sustainable behavior. In order to maximize profitability and ethos among consumers, companies have employed the use of greenwashing. The global community has agreed that action needs to be taken in order to preserve the planet and the quality of life for future generations. Blockchain technology has seen a great evolution and adoption in different industries. More and more applications and use cases for this technology is being uncovered. This project explores the use cases for blockchain technology for tracking the activities within the value chain and their impact on the environment. The result of this thesis will be a proof-of-concept solution that will attempt to address the problems consumers are facing in a convoluted market consisting of many actors with varying intentions. The end goal is to propose a solution that will aid in bringing transparency to the consumer and other actors such as governments, investors and non-government organizations in order to discourage and prevent the act of greenwashing among companies. Relevant questions to be addressed are: Is blockchain technology a viable solution as a way to track and record activities in the value chain? What are the main drawbacks and possible difficulties in adopting such a solution?

Approved on: 2023-03-22

Main supervisor: Prof. Gligoroski, Danilo, NTNU

Co-supervisor: Assoc. Prof. Kralevska, Katina, NTNU

Abstract

In the past decade, there has been a global shift towards focusing on sustainability due to the climate crisis that we are facing. New regulations and measures have been implemented with the aim of promoting sustainable behavior among consumers and businesses. Investors prioritize green investments, and consumers, especially among the younger generations, have become more concerned with the impact the products and services they purchase have on sustainability.

Greenwashing is the act of misleading consumers regarding the environmental practices of a company or the environmental benefits of a product or a service. Following the socioeconomic shift towards sustainability, greenwashing has become more widespread. The purpose of greenwashing is to achieve increased ethos among consumers and investors, leading to increased sales and a stronger brand. Unfortunately, due to the lack of transparency in the supply chain, it is hard to detect and evaluate potential instances of greenwashing.

Blockchain technology has evolved rapidly in the past years, and there has been great interest in exploring its use for the supply chain. In this thesis, we explore how blockchain can be used for sustainability reporting in order to bring transparency to the supply chain, and thus make it possible to detect and hinder greenwashing. First, we determine the requirements and needs based on the background research. Then we introduce our proof-of-concept implementation, Gaia, which is a sustainability reporting application running on Hyperledger Fabric, an industry-oriented blockchain system. Finally, we evaluate the implementation based on the requirements and address the barriers and challenges of adopting it in practice.

Our findings show that the proof-of-concept could be used to detect greenwashing if used by parties along the supply chain. It also addresses the absence of a standardized system for exchanging sustainability data. Despite the potential benefits, there are numerous barriers and challenges we need to overcome for the solution to be viable for use in practice.

Abstrakt

I de siste tiårene, har det foregått et globalt skifte for fokuset på bærekraft som følge av den pågående klimakrisen. Nye lover og tiltak har blitt innført der hensikten er å få fobrukere og bedrifter til å handle bærekraftig. Investorer prioriterer å gi ut grønne lån, og forbukere, spesielt blant den yngre generasjonen, har blitt mer opptatt av påvirkingen varer og tjenester de kjøper har på bærekraft.

Grønnvasking er en bevisst villeding av forbrukeres oppfattelse av miljøpraksisen en bedrift har, eller de miljømessige fordelene av varer og tjenester de selger. Som følge av det sosioøkonomiske fokuset på bærekraft, har grønnvasking blitt stadig mer utbredt. Hensikten med grønnvasking er å oppnå økt etos blant forbrukere og investorer, noe som leder til et sterkere merkevare og økt salg. Mangelen på åpenhet i forsyningskjeden, gjør det vanskelig å oppdage og vurdere hva som er grønnvasking.

Blokkjedeteknologi har utviklet seg enormt de siste årene, og forskningsmiljøet har hatt stor interesse for anvendelser av denne teknologien innenfor forsyningskjeden. I denne oppgaven, utforsker vi hvordan blokkjedeteknologi kan brukes til bærekraftsrapportering i forsyningskjeden, noe som vil gi åpenhet og muligheten til å avdekke og forhindre grønnvasking. Vi starter med å gjøre et literaturstudie for å fastslå krav og behov til løsningen. Deretter, introduserer vi vår prototype løsning, Gaia, som er et bærekraftsrapporteringsprogram som kjører på Hyperledger Fabric, et industrirettet blokkjedesystem. Til slutt, gjør vi en vurdering av sluttløsningen med hensyn på de opprinnelige kravene, og vi diskuterer hindringer og utfordringer rundt bruk av systemet.

Resultatetene våre viser at løsningen vår kan brukes til å oppdage grønnvasking, om den brukes av partene i forsyningskjeden. Den utgjør også en løsning på mangelen på et standardisert system for utveksling av bærekraftsdata. Til tross for de potensielle fordelene ved løsningen, er det flere hindringer og utfordringer vi må overkomme for at løsningen skal være gunstig å bruke i praksis.

Preface

This thesis concludes my five-year study journey at NTNU in Communication Technology and Digital Security. I am grateful for these amazing years that have provided me with lots of great experiences and memories I will never forget. I have been lucky to meet some amazing people and made new friends for life. Now my next adventure begins; finally, I can commit myself fully to my job at Kvist.

Acknowledgements

I would like to thank everyone who has assisted me in the thesis work. In particular, I would like to thank my supervisors Danilo Gligoroski and Katina Kralevska for providing guidance and support. I would like to thank NTNU for providing me with an amazing program of study, with a wonderful staff. Also, I would like to extend my gratitude to my colleagues at Kvist. Thank you all for your unconditional support.

Contents

Contents vii

List of Figures xi

List of Tables xiii

List of Acronyms xvii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Methodology . . . 2
  - 1.2.1 Literature studies: Sustainability reporting and greenwashing 2
  - 1.2.2 Modeling of the problem domain and determining requirements 3
  - 1.2.3 Experimental implementation . . . 3
  - 1.2.4 Evaluation . . . 5
- 1.3 Contributions . . . 5
- 1.4 Outline . . . 6

2 Background 7
- 2.1 Sustainability . . . 7
  - 2.1.1 UN Sustainability Development Goals . . . 8
  - 2.1.2 European Green Deal . . . 8
- 2.2 Sustainability reporting . . . 9
  - 2.2.1 EU Requirements . . . 9
- 2.3 Tracking emissions . . . 10
  - 2.3.1 Categorizing emissions by scope . . . 10
  - 2.3.2 Supply chain data sharing . . . 12
- 2.4 Greenwashing . . . 13

3 Theoretical preliminaries 15
- 3.1 Cryptographic concepts . . . 15
  - 3.1.1 Hash functions . . . 15
  - 3.1.2 Public-key encryption . . . 16

3.1.3 Digital signatures . . . 17
3.1.4 Public key infrastructure (PKI) . . . 17
3.1.5 Multi-party computation (MPC) . . . 19
3.1.6 Multi-input functional encryption (MIFE) . . . 19
3.1.7 AdHoc MIFE . . . 20
3.2 Blockchain . . . 20
3.2.1 Data structure . . . 20
3.2.2 Decentralization . . . 21
3.2.3 Smart contracts . . . 22
3.2.4 Storing data off-chain . . . 22
3.2.5 Permissioned versus permissionless . . . 23
3.3 Hyperledger Fabric . . . 23
3.3.1 Network architecture . . . 24
3.3.2 Identity management . . . 24
3.3.3 Ledger . . . 26
3.3.4 Transaction architecture . . . 28
3.3.5 Chaincode . . . 29
3.3.6 Privacy . . . 31

4 Modeling and requirements **33**
4.1 Stakeholders and concerns . . . 33
4.1.1 Developers . . . 33
4.1.2 Organizations . . . 33
4.1.3 System operators . . . 34
4.1.4 Non-governmental organizations . . . 34
4.1.5 Consumers . . . 34
4.2 Business requirements . . . 34
4.3 Trust model . . . 36
4.4 Functional Requirements . . . 36
4.5 Domain model . . . 40
4.6 Quality requirements . . . 43
4.6.1 Security scenarios . . . 43
4.6.2 Extensibility scenarios . . . 44
4.6.3 Modifiability scenarios . . . 45
4.6.4 Usability Scenarios . . . 46
4.6.5 Availability Scenarios . . . 46
4.6.6 Recoverability scenarios . . . 47
4.6.7 Deployability scenarios . . . 47
4.6.8 Scalability scenarios . . . 48
4.7 Architecturally significant requirements . . . 48
4.8 Architectural drivers . . . 48
4.8.1 Business requirements . . . 48

4.8.2 Functional requirements . . . 49
4.8.3 Quality requirements . . . 50

5 Experimental implementation: Gaia 53
5.1 Description . . . 53
5.2 Blockchain selection . . . 53
5.2.1 Cost . . . 54
5.2.2 Sustainability . . . 54
5.2.3 Security . . . 55
5.2.4 Scalability . . . 55
5.2.5 Programming capabilities . . . 56
5.3 Chaincode . . . 56
5.3.1 Domain layer . . . 57
5.3.2 Contract layer . . . 59
5.3.3 Infrastructure layer . . . 59
5.4 Architectural tactics . . . 60
5.4.1 Security . . . 60
5.4.2 Modifiability . . . 63
5.4.3 Extensibility . . . 63
5.5 Hyperledger Fabric . . . 64
5.5.1 Network architecture . . . 64
5.5.2 Endorsement policy . . . 64
5.5.3 World-state database . . . 65
5.6 Scenarios . . . 66
5.6.1 Scenario 1: Registering a business activity . . . 66
5.6.2 Scenario 2: Two parties performing an interaction with the transfer of assets . . . 66

6 Evaluation and discussion 75
6.1 Functional requirements . . . 75
6.2 Quality requirements . . . 77
6.2.1 Extensibility . . . 77
6.2.2 Modifiability . . . 77
6.2.3 Deployability . . . 78
6.2.4 Scalability . . . 79
6.2.5 Security . . . 79
6.3 Application for greenwashing . . . 80
6.3.1 Product-level greenewashing . . . 80
6.3.2 Organization-level greenwashing . . . 83
6.4 Barriers . . . 86
6.5 Challenges . . . 88
6.6 Discussion . . . 89

7 Conclusion **91**
7.1 Future work 92

References **95**

List of Figures

1.1 Flow chart illustrating the methodology of the thesis. 2

2.1 Greenhouse gas emission scopes. Source: Figure 1.1 of [IfS11] 11
2.2 Entities and relationships in the greenwashing domain. 14

3.1 Chain of trust in public-key infrastructure. 18
3.2 Fabric network architecture for an example network consisting of three organizations. Source: [Foua] 25
3.3 Membership service providers in a Fabric Network. Source: [Foub] . . . 26
3.4 Blockchain data structure in Fabric. Source: [Fouc] 27
3.5 Transaction structure in Hyperledger Fabric. Source: [Foud] 28
3.6 Sequence diagram of execute-order-validate architecture. Source: [Foue] 30
3.7 Private data collections are bound to organizations and are stored off-chain. Source: [Fouf] 32

4.1 Trust model for the implementation. 37
4.2 Model of the sustainability reporting domain. 42

5.1 Chaincode software architecture 58
5.2 Class diagram of the domain layer. 58
5.3 Class diagram of the infrastructure layer. 60
5.4 Secure decentralized data aggregation. 62
5.5 Configured network architecture for Hyperledger Fabric. 65
5.6 Scenario 1: Business activity details submitted in the registration. . . . 67
5.7 Scenario 1: Registering business activity that produces three assets. . . 68
5.8 Code snippet: `registerActivity` implementation. 69
5.9 Scenario 2: Two parties performing an interaction with the transfer of assets. 70
5.10 Code snippet: Scenario 2 – `proposeInteraction` implementation. . . . 71
5.11 Code snippet: Scenario 2 – `confirmInteraction` implementation. . . . 72
5.12 Code snippet: Scenario 2 – `verifyProvidedInteraction` implementation. 73

6.1 The supply chain of Yogoa. 81

6.2 Organization-level greenwashing: Total CO2 emissions of Saastastic. . . 85
6.3 Organization-level greenwashing: CO2 emissions of Saastastic, excluding office operation. 87

List of Tables

4.1 Functional requirements for the application 38
4.2 Explanation for the domain model components presented in Figure 4.2. 41
4.3 Security scenario 1: Registering reporting data securely 43
4.4 Security scenario 2: Secure data aggregation 44
4.5 Security scenario 3: Privacy preservation after asset transfer. 44
4.6 Extensibility scenario 1: Add new sustainability outcome type. 44
4.7 Extensibility scenario 2: Implement a new query 45
4.8 Extensibility scenario 3: Add a new contract 45
4.9 Modifiability scenario: Modify existing command by introducing the new business rule. 45
4.10 Usability scenario 1: Enroll in the blockchain network. 46
4.11 Blockchain node failure. 46
4.12 Recoverability scenario: Transient network failure 47
4.13 Deployability scenario: Deploy a new version of blockchain application. 47
4.14 Scalability scenario: Deploy a new version of blockchain application. . . 48
4.15 List of architecturally significant requirements and explanations. 49

6.1 Product-level greenwashing scenario: Registered business activities. . . . 81
6.2 Product-level greenwashing scenario: Registered interactions. 82
6.3 Organization-level greenwashing: Registered activities. 85
6.4 Organization-level greenwashing scenario: Registered interactions. . . . 87

List of Acronyms

API Application programing intrface.

ASR Architecturally significant requirement.

B2B Business-to-business.

B2C Business-to-consumer.

CA Certificate authority.

DDD Domain-driven design.

DTO Data transfer object.

FR Functional requirement.

HTTP Hypertext transfer protocol.

M:M Many-to-many.

MIFE Multi-input functional encryption.

MPC Multi-party computation.

NGO Non-governmental organization.

PDC Private data collection.

PKE Public-key encryption.

PKI Public-key infrastructure.

QR Quality requirement.

REST Representational state transfer.

SDG Sustainable Development Goal.

SHA Secure Hash Algorithm.

TLS Transport Layer Security.

Chapter 1
Introduction

1.1 Motivation

The idea for this Master's thesis started with my interest in the domain of sustainability and how technology may help society. In recent years we have become more aware of the term "greenwashing". According to Miriam Webster Dictionary, it is *"the act or practice of making a product, policy, activity, etc. appear to be more environmentally friendly or less environmentally damaging than it really is"*. The American Heritage Dictionary of the English Language describes greenwashing as *"the dissemination of misleading information that conceals abuse of the environment in order to present a positive public image"*. Similarly, Investopedia defines it as *"the act of providing the public or investors with misleading or outright false information about the environmental impact of a company's products and operations"*.

Greenwashing is something that, unfortunately, many consumers are exposed to, which is the reason why I found this problem so motivating, given its potential impact. Also, in the past two years, my work has been focusing on the construction industry through my job as a systems architect in Kvist Solutions, which is digitizing the process of environmental certifications to promote and facilitate sustainable construction. This thesis is a collaboration with this company, and the resulting work will potentially be of value in the future as we may look to further build on this work.

Greenwashing has become ever more widespread among companies that sell their products to consumers. The background for this increase is the shift in focus toward sustainability. Investors and consumers have become more concerned with spending their money on products and businesses that focus on sustainability. In order to maximize profitability and ethos among consumers, businesses have employed the use of greenwashing. Unfortunately, due to the lack of transparency, it is hard to separate truth from fiction and hold businesses accountable for their claims.

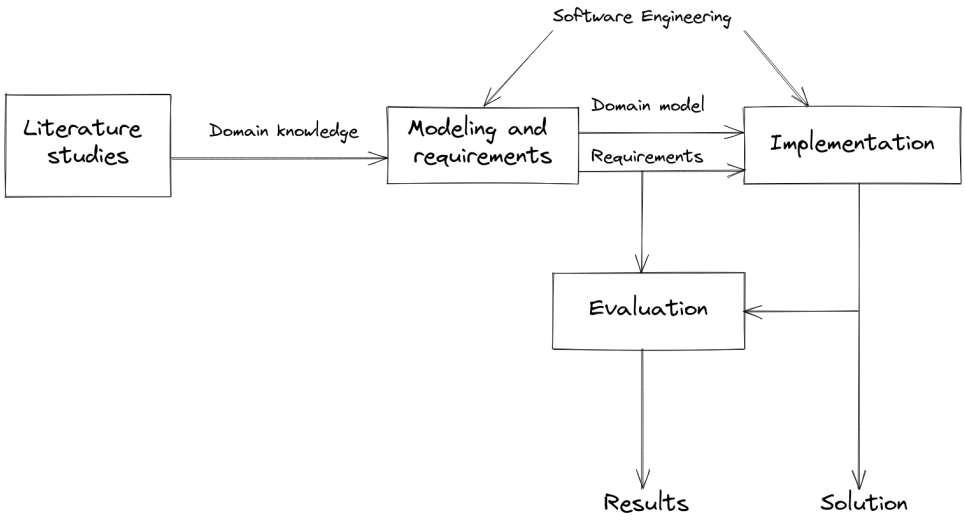

Figure 1.1: Flow chart illustrating the methodology of the thesis.

Blockchain technology has seen a great evolution and adoption in different industries. More and more applications and use cases for this technology are being uncovered. What is special about blockchain is how the data is stored immutably as a sequence of transactions. By doing so, the entire history of the blockchain is preserved. This gives rise to various applications where transparency and auditability are key requirements. This leads to the idea of exploring blockchain for sustainability reporting as a way to bring transparency to the supply chain and thus aid in exposing and preventing greenwashing.

1.2 Methodology

The methodology of this thesis can be divided into four stages, as displayed by the flow chart in Figure 1.1.

1.2.1 Literature studies: Sustainability reporting and greenwashing

In the first phase of the thesis, a literature study was conducted in order to establish an overview of the sustainability and greenwashing domain from a societal perspective. Google Scholar[1] was used extensively in this phase which aided in finding reliable and up-to-date information from the experts in the field. The result from this work served

[1]https://scholar.google.com

as the background for understanding the problem domain. This domain knowledge is the foundation for the high-level models and set of requirements that were produced in the modeling phase of the thesis.

Following the initial academic research at the conceptual level, we studied specific sustainability reporting frameworks that are in use today to better understand the details and nuances of these different standards. The goal of this research was to better understand the process and requirements of sustainability reporting.

1.2.2 Modeling of the problem domain and determining requirements

As a way to structure and gather an overview of the concepts that we learned from the literature studies, we took a model-driven approach. The modeling process was concerned with extracting and understanding the high-level concepts relating to the problem and then translating these concepts into a high-level entity-relationship view model of greenwashing and sustainability reporting with a societal perspective.

Using this model and the obtained domain knowledge from the research into sustainability reporting, we created a software requirement specification in which we determined the needs of the different stakeholders which we used to establish the requirements for the implementation. From these requirements, we created a software model of the domain that would be the foundation for the business logic of the implementation.

1.2.3 Experimental implementation

The implementation process was the most complex part of the work conducted. A systematic approach using the methodologies from the field of software engineering was taken. After establishing a clear understanding of the problem domain, and the business requirements for the solution, we translated this into a set of functional and non-functional requirements. The requirements were then analyzed to determine their impact on the architecture. Once a clear understanding of the architectural drivers was established, we began designing the architecture of the solution. We started with the overall system architecture by determining which blockchain to use. Then we moved on to designing the software architecture, and we ended up with a layered architecture. For the core layer of the application, which constitutes the entities and business logic, we followed the principles of domain-driven design based on the domain model we created in Chapter 4.

Domain-Driven Design

Domain-Driven design (DDD) [Eva04] is a software design approach that has become very popular for enterprise application development. DDD focuses on designing software that models the domain; classes, variables, and methods should all be named according to the domain language. Technical details should be left out, and only concepts that exist within the domain should be considered. This way, it becomes easier to learn and conceptualize the domain by modeling with the same language that the domain experts use. The domain language is also referred to as the ubiquitous language as the language is ubiquitous to the domain experts. In cases where the model is large, different subdomains can have concepts that use the same language, leading to ambiguities in the semantics of the model. To deal with this problem, a strategic pattern by domain-driven design is to create several models, with an unambiguous domain language, bounded to a specific context, referred to as the bounded context. Numerous tactical patterns have since been introduced, with the key concepts being *entities*, *value objects*, and *domain events*. Entities represent objects that have a life cycle with a unique identity. They usually have methods defining behavior that the entity can perform. Invoking such a method may change the state of the entity as well as emit domain events. An example of an entity could be a Customer in a banking domain. Value objects, on the other hand, are immutable and represent value concepts of the domain. Sticking with the banking domain, there could be a value object called `Money`, representing an amount of currency. Events of the banking domain could be related to `Account` objects, such as `AccountOpened`, `AccountMoneyWithdrawn`, `AccountMoneyDeposited`, and so on. Using these tactical building blocks, it is easier to systematically implement the domain logic of an application, which is the reason DDD has become so popular.

Tools

The following tools were used for the implementation:

- Visual Studio Code: Code editor
- GitHub: Code hosting and version control with Git.
- Hyperledger Fabric: Blockchain operating system that our solution runs on.
- TypeScript: Programming language used for the implementation.
- Awilix: Dependency container used to dynamically register and resolve dependencies in run-time.[2]

[2] https://github.com/jeffijoe/awilix

1.2.4 Evaluation

Following the implementation, we carried out an evaluation of the resulting solution according to the requirements that we started with. The purpose of the evaluation was to determine which requirements had not been fulfilled and ways to address them. In addition to evaluating the requirements, we examine the potential barriers and challenges of using the solution in practice.

1.3 Contributions

There are several contributions in this thesis, with the primary contribution being the proof of concept implementation.

Proof-of-concept implementation The proof-of-concept implementation is a blockchain application that is developed on top of the Hyperledger Fabric blockchain operating system. The application implements a sustainability reporting system that can be used to record business activities along with their associated sustainability impact. It also provides the ability to register assets that belong to the reporting business. The last and major use case is the ability for businesses to register interactions between them. An interaction can be used to model the usage of a service provided by a service provider or the purchase of goods from a supplier, causing a transfer of asset ownership.

The code of the implementation is published openly on GitHub and can be found at https://github.com/udnes99/Gaia. Since the implementation chapter was written, we have refactored the architecture by decoupling the business logic from Hyperledger Fabric altogether; the interface towards Fabric is only used as a data access layer. The code that is of relevance to Chapter 5 is located at the *master-thesis* branch[3].

Additional contributions While the implementation is the main contribution, there are additional contributions that we have made in the process of building the implementation:

- **Domain modeling**: We have used modeling extensively as part of the thesis work, and we produced two models:
  - Societal model of the greenwashing domain is constructed from the knowledge gathered during literature studies.
  - High-level domain model for the software implementation.

[3] https://github.com/udnes99/Gaia/tree/master-thesis

- **Software requirements specification**: A specification of the requirements created based on domain knowledge obtained from the background research.
- **Evaluation of end solution**: An evaluation of the end solution concerning the requirements to determine satisfaction.
- **Secure data collaboration proposal**: As part of the implementation, we propose a protocol by which data can be securely used for collaboration when computing aggregated data such as emissions along the supply chain.

1.4 Outline

The thesis is comprised of seven chapters. Chapter 1 introduces the thesis—the motivation behind it, the methodology of the process, and its contributions. Chapter 2 provides background material that is related to the problem and research questions. Chapter 3 introduces theoretical material that is required for understanding the tools and concepts that are applied. In Chapter 4, we conduct preliminary work that involves modeling and establishing the requirements of the proof of concept solution. Then in Chapter 5, we describe the design and implementation process that is based on the requirements and drivers established in Chapter 4. Chapter 6 consists of evaluating the resulting solution and discussing results in light of the background material and the requirements that are defined in Chapter 4, and we evaluate its efficacy in the context of greenwashing. Finally, Chapter 7 concludes the thesis by answering the research questions and presenting future work.

Chapter 2
Background

Before we can start designing a solution, we need a clear understanding of the problem domain, which are the domains of sustainability reporting and greenwashing. A literature study was conducted on these topics. We will now present the findings obtained from the study.

2.1 Sustainability

Sustainability refers to the ability of human society and the natural environment to coexist in a way that meets the needs of the present without compromising the ability of future generations to meet their own needs. It involves balancing economic, social, and environmental considerations to create a sustainable and resilient system.

The concept of sustainability emerged in the 1980s as a response to growing concerns about the impact of human activities on the environment. Since then, sustainability has become a major global issue, and rightfully so. If we want future generations to be able to thrive on this planet as we have, action must be taken in order to prevent irreversibly damaging our environment.

Since the first use of the word sustainability in the context of the environment, the concept has been extended with two additional dimensions: social and economic. Hence, sustainability is often framed in terms of the "three pillars" of sustainability. The environmental pillar focuses on protecting natural resources, ecosystems, and biodiversity, while the economic pillar seeks to create a sustainable economy that meets the needs of all people without depleting natural resources or creating inequality. The social pillar aims to promote social justice and equity, ensuring that everyone has access to the resources and opportunities they need to thrive.

Sustainability is important because it is essential for ensuring the long-term well-being of both human society and the natural environment. By promoting sustainable practices, we can help reduce the impact of human activities on the environment,

ensure that natural resources are used responsibly, and create a more equitable and just society for all.

Following the introduction of the concept of sustainability, it has become a major focus point of the international community. Developed nations, in particular, have made great strides to implement a strategy with the intention of reaching the established sustainability goals.

This has resulted in numerous governing agencies coming up with protocols and goals. As a result, ever stricter laws and requirements are being introduced and imposed on businesses that incentivize sustainable behavior.

2.1.1 UN Sustainability Development Goals

As a result of the global alignment on sustainability, the UN introduced the *sustainability development goals* (SDGs) during the UN conference on sustainable development in 2012.[1]

The UN has defined 17 goals that address sustainable development [UN]. Among these, goals 9[2] and 12[3] are of relevance to this thesis. Goal 9 is concerned with innovation that will aid in sustainable development. Goal 12 focuses on sustainable consumption and production patterns within the supply chain, something the work of this thesis may contribute to.

2.1.2 European Green Deal

On December 11, 2019, the European Commission introduced the European Green Deal. It is a roadmap for how to make Europe climate-neutral by 2050, in which all sectors of the economy are covered. The key goals of the European Green Deal are:

- Europe becoming climate neutral by 2050;
- 3 billion trees planted by 2030;
- At least 55% less greenhouse gas emissions compared to 1990 by 2030.

In order to achieve these goals, the European Green Deal gradually rolls out new laws and regulations that are covering businesses and countries of the EU. These address a whole range of sustainability aspects such as toxic waste pollution,

[1] https://www.undp.org/sdg-accelerator/background-goals

[2] SDG 9: Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

[3] SDG 12: Ensure sustainable consumption and production patterns

deforestation, achieving a circular economy, and the development of new sustainable technologies, to name a few. The *European Climate Law* [21] was introduced by the European Commission as part of the European Green Deal and writes into law the climate objectives for the goal of becoming climate neutral by 2050. Imposed by this law is the requirement of member countries to reduce their greenhouse gas emissions by at least 55% relative to 1990.

Since the European Green Deal brings new laws and regulations imposed on the member countries of the EU, domestic laws and regulations will therefore follow. To ensure and encourage the regional and local authorities within the EU, the *European Committee of the Regions* launched the initiative *Green Deal Going Local* [Eur].

2.2 Sustainability reporting

Sustainability reporting is the disclosure of business activities and their impact that are relevant to the three pillars of sustainability. There are several benefits of sustainability reporting. First of all, is an increase in company reputation by being transparent, which in turn can greater investments from investors and increased ethos towards the customers, resulting in more sales. Not only is the practice of sustainability reporting beneficial towards external relations, but it can lead to a better relationship between the employees and the business due to the values sustainability focus displays.

External relations are not the only benefit of practicing sustainability reporting; it helps businesses gain insight into their resource usage, which can be used to identify improvement potentials, such as replacing inefficient legacy equipment with new, more efficient options or identifying ways to improve business operations.

New reporting requirements have followed the new laws and regulations that have been introduced to tackle the sustainability problem the world is facing. EU law requires all large companies as well as smaller enterprises to disclose information about the risks and opportunities for social and environmental issues, as well as their environmental and societal impact.

2.2.1 EU Requirements

On the 5th of January 2023, the Corporate Sustainability Reporting Directive (CSDR), introduced in the EU, came into effect [Com]. The directive made sustainability reporting mandatory for a broader set of large and small companies, approximately 50 000 companies in total. The year 2024 is the first year in which the new reporting requirements must be adhered to, and the reported data for 2024 will be published in 2025. The companies that are subject to following the reporting directive will

need to report in accordance with the European Sustainability Reporting Standards (ESRS). In addition to following the guidelines of the ESRS, it is mandatory for the reporting companies to have an audit of the reported sustainability information.

The ESRS has not yet been fully developed and is currently being drafted by the private association EFRAG, which was appointed to be the technical advisor of the ESRS for the commission as part of the CSDR. [EFR]. The ESRS defines a set of standards, which are either cross-cutting or topical. The cross-cutting standards define requirements that apply to all reporting entities, regardless of industry.

The topical standards are concerned with different aspects of sustainability and are grouped into environmental, social, and governance categories. In particular, the ESRS standard E1 is related to climate change and defines reporting requirements for greenhouse gas emissions and measures taken to reach the climate goals of the European Green Deal.

2.3 Tracking emissions

Tracking emissions is not a trivial task for a business, and several frameworks have been developed in the past decades to aid in estimating with greater accuracy. The problem with tracking emissions associated with any particular business is the origin of the emissions. It is not enough to consider the direct emissions that a business generates on-site, as it would give a skewed image based on the type of business being assessed. Consider a business whose core activity is renting out cars; this business is not directly causing any emissions if we assume that renting out cars is the only activity they do and they never drive their own cars. However, if we consider the downstream activity, which in this case is the driving of rented cars performed by their customers, the image is entirely different. Looking the other way, i.e. upstream, there are also the emissions caused by the production and delivery of the car to the rental company. Clearly, they are facilitating those emissions.

2.3.1 Categorizing emissions by scope

To understand the bigger picture of the environmental impact of a business, we need a way to systematize emissions. We classify the different types of emissions into scopes 1, 2, and 3. This classification was introduced by the GHG protocol [Pro11], which is the world's most widely used greenhouse gas accounting standard. Figure 2.1 illustrates the scope classification system for the supply chain. The scopes range from the most direct to indirect.

Scope 1 emissions are emissions that are caused directly by the activities of the business in question. These could be emissions generated from an industrial process

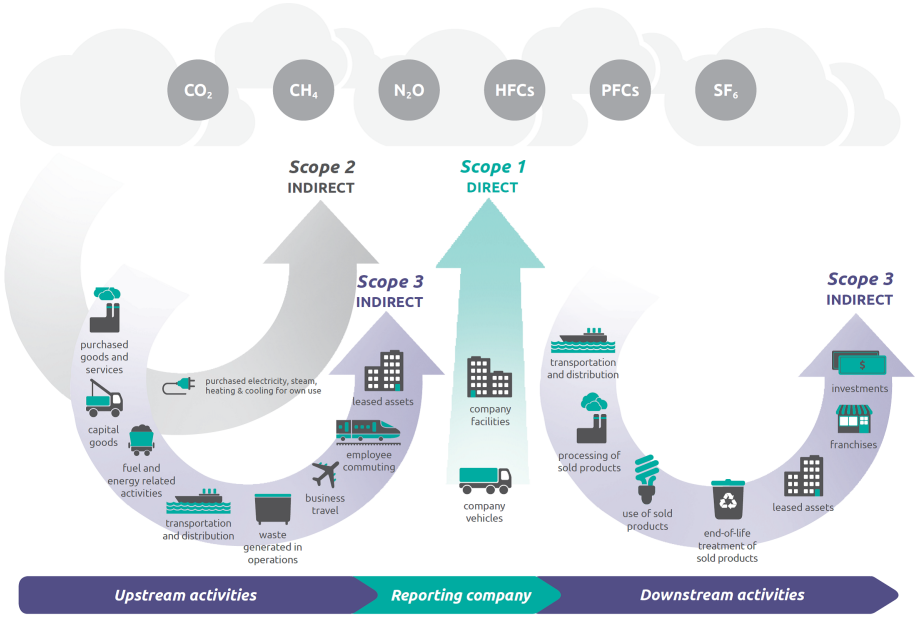

Figure 2.1: Greenhouse gas emission scopes. Source: Figure 1.1 of [IfS11]

or from fuel combustion in a motorized vehicle.

Scope 2 emissions are the emissions caused indirectly through the consumption of energy sources that the business purchases to enable their business activities, such as electricity and heating. Both scope 1 and 2 emissions are controlled by the business directly and can be measured.

On the other hand, downstream and upstream emissions, referred to as scope 3 emissions, are much harder to keep track of, as they are indirect emissions and are not directly controlled by the reporting business. Upstream emissions are the emissions involved in producing a good or service that a business purchases from a supplier. For example, the emissions generated from the outsourcing of manufacturing of a product would count as upstream scope 3 emissions. Downstream emissions can, for instance, be life cycle related, such as the emissions caused by end-of-life treatment of sold goods or the sale or renting out of equipment that generates emissions. It is clear that we need transparency in order to determine the total scope of 3 emissions accurately. The problem is there is no open and standardized system through which this information can be stored and computed in a secure manner.

To further motivate the importance of accurately knowing the scope 3 emissions,

let us look at the distribution of the emission scopes in the global economy. According to [HW18], which conducted an economy-wide analysis on the share of the different emission scopes and their growth trajectory by dividing the global economy into five sectors - energy supply, transport, industry, buildings, and agriculture and forestry, defined by the Intergovernmental Panel on Climate Change. Their findings show that global scopes 1, 2, and 3 emissions grew by 47%, 78%, and 84%, respectively, between 1995 and 2015, with the majority of the growth occurring in developing countries. They also found the industry sector to be the sector with the highest share of scope 3 emissions relative to the other two scopes while also being the sector with the highest growth of total (including all scopes) emissions.

Today, scope 3 emissions are not measured directly but rather estimated based on quantitative models based on industry standards and reference values. The only internationally accepted standard for carrying out scope 3 accounting is the *Scope 3 standard* created by the *Greenhouse Gas Protocol* [IfS11]. Unfortunately, these estimates are inherently uncertain to varying extents depending on the available data and the model being used. The model used for a calculation depends on the type of service or good being estimated. For instance, there are models for estimating transportation emissions based on parameters such as fuel consumption or distance traveled. Whereas another model used to estimate emissions of purchased goods take the composition of raw materials of the end product into consideration.

2.3.2 Supply chain data sharing

To accurately calculate scope 3 emissions and not rely on quantitative models that estimate based on reference values, the different actors of the supply chain need to collaborate by sharing data. A paper by Stenzel et al. [SW23] looks into the possibility of achieving this. They explore the benefits of sharing data, the obstacles of data sharing, and ways of overcoming them. Their work presents the following issues: lack of legal clarity and regulatory concerns, lack of data and action interoperability, and the high risk of sharing sensitive data.

Lack of legal clarity and regulatory concerns is a primary obstacle that needs to be addressed. Today, organizations are limited in their ability to re-share data that includes emission data received by their partners in the supply chain. Furthermore, more and more countries have started to implement data localization measures that impose restrictions on where data can be transferred. In addition, the data can include additional information that may be relevant to competitors. Overcoming these challenges will require governmental agencies, such as the United Nations and the European Union, to remove the legal barriers to data sharing.

Another obstacle that needs to be addressed is interoperability. According to the paper, the lack of interoperability is twofold. Firstly, a lack of standards to measure

GHG emissions. Secondly, there is no standardized IT infrastructure that can be used to share data.

The final obstacle they address is the risk involved in sharing sensitive data. Emission data can be used to perform reverse engineering that can reveal information about production processes and other corporate secrets. The authors propose using cryptographic schemes such as homomorphic encryption and secure multi-party computation to securely compute an aggregated result over private data without revealing the data itself.

2.4 Greenwashing

Following the increase in focus on new regulations for sustainability, the concept of greenwashing has become ever more prominent. Greenwashing is the act of misleading consumers regarding the environmental practices of a company (organization-level greenwashing) or the environmental benefits of a product or a service (product-level greenwashing) [DB11]. The end goal for greenwashing is an increased ethos which in turn leads to increased profitability. In particular, this has become more widespread within business-to-consumer markets in the past decades. This may come in several different shapes. For instance, unverifiable goals and claims are one example. Another is misleading statements regarding the production process for a given product that may implicate environmental friendliness, while in reality, information about the upstream business production processes is undisclosed, leading to a false impression in the consumer. Another example of misleading marketing is through numbers: a manufacturer could claim that their product is made of 50% additional recycled material when in reality, the total share of recycled material has increased from 2% to 3%. While technically true, this reveals no information about the product's eco-friendliness, and consequently, it is misleading the consumer. Another important issue is the impact of sourcing recycled material, which could be costly for the environment. These are all forms of greenwashing, and there is no way for consumers alone to verify and assess these claims. There are indeed Non-Government Organizations (NGOs) that try their best to uncover and disclose greenwashing to the public. Despite their best efforts, the underlying problem of traceability and lack of resources means they can only really uncover the tip of the iceberg of greenwashing.

Figure 2.2 is a conceptual model illustrating the domain of greenwashing using the involved entities and their relationships. Arrows between entities represent relationships, where the entity from which the arrow originates is the subject responsible for forming the relationship. This model serves as the foundation for understanding the greenwashing domain and is the basis for the implementation.

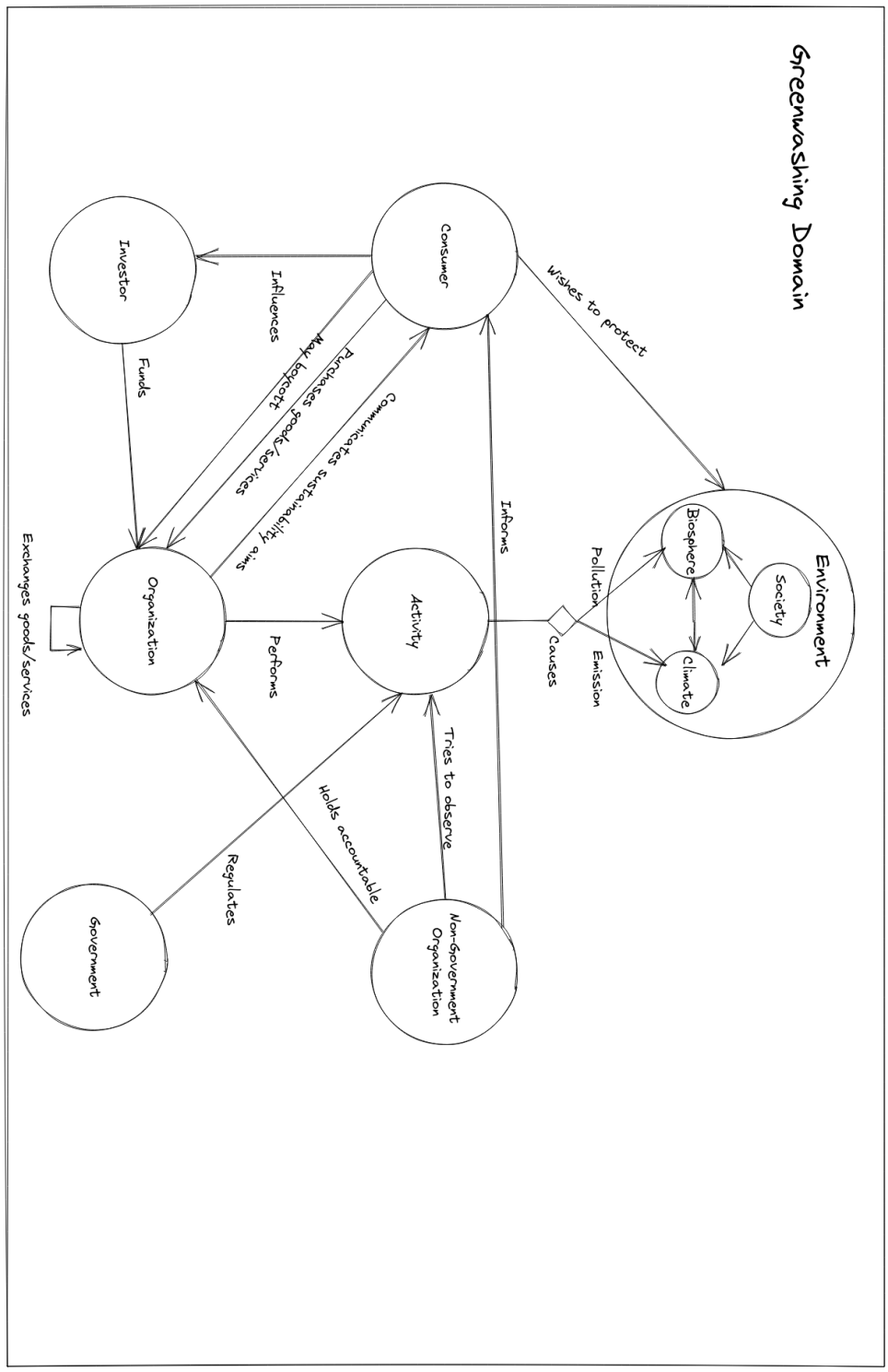

Figure 2.2: Entities and relationships in the greenwashing domain.

Chapter 3
Theoretical preliminaries

In this section, we will present the relevant theoretical background that the experimental results build upon. In particular, the domain of sustainability reporting and tracking is the core domain of our implementation providing solutions. At the technical level, we will be using tools and theories from the field of cryptography in order to address the information security requirements of the experimental implementation which itself is an implementation of a permissioned blockchain.

3.1 Cryptographic concepts

The presented solution employs the use of many different cryptographic concepts. In this section, we will briefly describe some of those concepts and their relationships. For an extensive overview of the use of cryptography in blockchain systems, we suggest the reader consult [RGK19].

3.1.1 Hash functions

Hash functions are one of the most important cryptographic primitives. It provides the basis for digital signatures and other cryptographic schemes. A hash function is a one-way function that takes in an input of variable length and outputs a deterministic fixed-length bit string, referred to as the fingerprint of the input.

More formally, we define a hash function as follows:

Definition Let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k$ be a function from the set of all possible binary strings to the set of all binary strings of length k. $\mathcal{H}$ is a hash function if for all input messages M, $\mathcal{H}(M)$ is deterministic, i.e. the computation $\mathcal{H}(M)$ always yields the same output.

A trivial example of this would be the function

$$\mathcal{H}(M) = 1$$

This function is deterministic, as it is always defined to output the value 1. However, this is not a cryptographically secure hash function. For that, we require our hash function to have the additional properties:

1. *Collision resistance*: It is infeasible to find two distinct messages M_1 and M_2 such that $\mathcal{H}(M_1) = \mathcal{H}(M_2)$. If the output is the hash function is uniformly distributed, and the length of the hash is k bits, we have collision resistance for up to $k/2$ length bit strings. This result follows from the solution to the birthday paradox, assuming a uniformly distributed output.

2. *Pre-image resistance*: Given the hash $\mathcal{H}(M)$, it is computationally infeasible to find an input M' such that $\mathcal{H}(M') = \mathcal{H}(M)$

3. *Second pre-image resistance*: Given the input message M, it is computationally infeasible to find another message $M' \neq M$ such that $\mathcal{H}(M') = \mathcal{H}(M)$

The secure hash algorithms (SHA) are a family of cryptographic hash functions that have been published by the *National Institute of Standards and Technology* (NIST). They have gone through extensive auditing to ensure the security requirements are met. Hence, they are the most widely used in practice.

Use cases Cryptographic hash functions have a wide variety of use cases. A simple use case is integrity protection which is achieved by running a hash function on an unverified input whose expected hash fingerprint is known. If the hash of the computed input does not match the expected fingerprint, we can be sure that the input is not equal to the expected value. This is used all throughout communication protocols where the integrity of the exchanged data must be protected and any corruption or tampering must be detected.

Importantly, cryptographic hash functions serve as cryptographic primitives which can be used to construct advanced cryptographic schemes such as digital signatures, message authentication codes, and zero-knowledge proofs, to name a few.

3.1.2 Public-key encryption

Public-key encryption (PKE), otherwise known as asymmetric encryption, is a cryptosystem in which pairs of related keys are used for encryption and decryption. The encryption key, which is referred to as the public key, is used to encrypt data and can be made publicly available. Anyone can encrypt data using the public key. On the other hand, the corresponding private key is kept secret so that encrypted messages can only be decrypted by the owner of the private key.

Contrary to asymmetric encryption, regular symmetric encryption uses a shared key for both encryption and decryption. Encryption and decryption using symmetric schemes are more efficient in terms of speed and space complexity, but we need some way to establish a shared secret key in a secure manner. Therefore, public key encryption and symmetric encryption are often used complementary, where PKE is used to provide a secure way to establish ephemeral keys for the communications session. One of the most widely used key exchange protocols today that accomplishes this is the Diffie-Hellman protocol. ([DH76])

3.1.3 Digital signatures

A digital signature scheme is a protocol that provides functionality to digitally sign and verify data. Digital signature schemes are a specific type of public key cryptosystem, where pairs of private (signing) and public (verification) keys are used. A signer uses their private signing key, k_s, in order to generate a signature on a given message M. The verifier can then verify the signature for the provided message using the public key, k_v.

In general, the message that is being signed is the hash fingerprint of the data that is actually being signed. This is in practice equivalent to signing the data itself, as long as we are using a cryptographically secure hash function, as described in the section about hash functions. By signing the hash of the data instead of the data itself, the signatures are more efficient to generate and also more space-conserving.

3.1.4 Public key infrastructure (PKI)

Public key infrastructure is a key management architecture in which cryptographic keys and digital certificates are managed in a hierarchical trust model. A digital certificate is a data structure representing the public information associated with its related private key. In addition to the public key, a certificate can also contain identifying information, such as the organization to which it belongs, domain names, etc. This makes digital certificates a great way to implement authenticated encryption.

In PKI, one or more parties provide a source of trust by serving as a certificate authority (CA). The responsibility of a CA is the issuance and management of certificates. In order for a CA to issue a new certificate, it does so by either signing a provided certificate or generating a new one, with its own private signing key. The signature serves as a stamp of validity. During the signing procedure, an expiration timestamp is also supplied, as certificates should not remain valid forever. If an unexpired certificate issued gets compromised, or if the issuance was incorrect, the CA can revoke the issued certificate by adding the certificate to a certificate revocation list (CLR) which it will sign to ascertain that the updated list is correct.

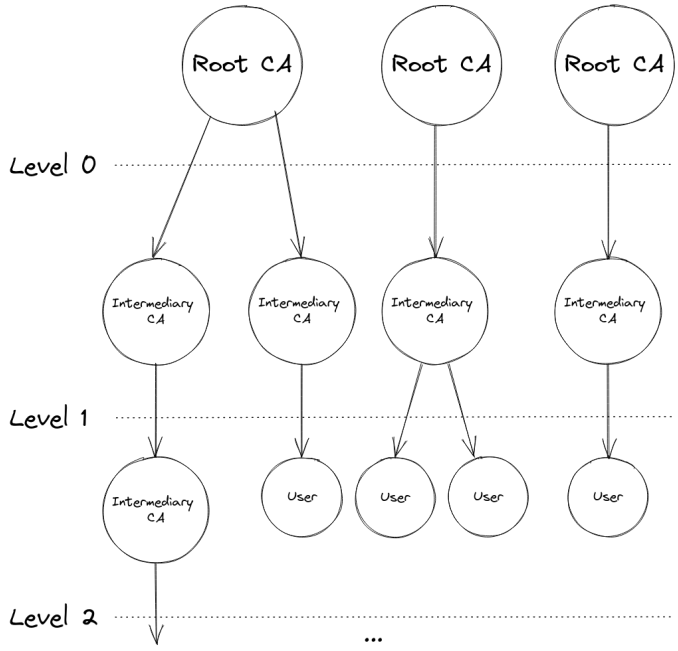

Figure 3.1: Chain of trust in public-key infrastructure.

A certificate authority can be the authority of another (smaller) certificate authority. A non-root CA is referred to as an intermediary CA. CAs form a hierarchy of trust, where the CA at the top of the hierarchy is referred to as the root CA. Every CA in the chain of trust is responsible for certifying and issuing certificates to parties of the next level in the trust hierarchy. This chain of trust is represented in figure 3.1; the highest level of trust is at the root level where the root CAs reside. Each subsequent level below is either an intermediary CA or an end user, and their certificates are signed by a CA in the level above.

Every certificate issued by a CA will be indirectly endorsed by central authorities higher up in the hierarchy because their certificates have been signed by those above, except for the certificate of the root CA. This allows us to segment the CA architecture based on different use cases, where the highest level of trust is at the top of the chain.

By segmenting trust this way, we only need to store the certificate of a few root CAs instead of all certificates that we are going to use. This is exactly how web browsers work when displaying the green padlock for a website it deems trusted; upon visiting a website, the web server will send its certificate to the browser. Subsequently, the browser will inspect the signatures that are present in the certificate and check

to see that a signature issued by a trusted CA is present. If so, the web browser will display a green padlock, otherwise, the site is deemed untrusted.

While it is advantageous to have a trust model that creates a hierarchy of trust from a usability and scalability perspective, it comes with the cost of introducing central points of attack. If a CA gets compromised, then the validity of the descendant certificates is compromised. There have been demonstrated numerous instances in which a CA has issued a certificate with the wrong claims to the wrong subject. This can have dramatic consequences from a security perspective. For example, when Microsoft publishes new software for the Windows operating system such as updates or drivers, these are all signed by Microsoft in order to authenticate the origin of the software. Windows will deny the installation of unsigned drivers. However, if a certificate claiming the identity of a trusted provider gets into the wrong hands, it can be used to circumvent the security protocol and install malicious software for the time being until the security breach has been discovered, at which point the certificate will be revoked by the CA.

3.1.5 Multi-party computation (MPC)

Multi-party computation (MPC) is a set of protocols that allow multiple interacting parties to collaborate on computing a joint result based on inputs that are supplied by each party, while the individual inputs preserve confidentiality. There are several ways of constructing an MPC scheme. In general, MPC relies on the assumption of having a stable communication channel between the involved parties. One common approach is for each party to divide their input into several shares, and then distribute these to the other parties in a randomized fashion. Then each party locally computes the partial result and the full result is obtained by combining the partial results.

3.1.6 Multi-input functional encryption (MIFE)

Multi-input functional encryption (MIFE) is a type of cryptographic scheme that allows multiple parties to compute specific functions over multiple encrypted inputs, without revealing the inputs.MIFE builds on the concepts of functional encryption, which is a generalization of traditional public-key encryption that allows authorized parties to compute a specific function over encrypted data without revealing the data itself. MIFE extends this concept to multiple inputs, enabling the computation of a function over multiple encrypted inputs while preserving the privacy of the inputs and the function being computed. An example of such as use case would be a set of competitors wanting to learn about the market they share. Instead of relying on a trusted third party, or each other, and risking confidential data being leaked, they can use MIFE.

3.1.7 AdHoc MIFE

Adhoc MIFE [ACF+19] further builds on MIFE and addresses the limitations of traditional MIFE. While traditional MIFE requires that the function, f, is fixed as well as the number of inputs of f, the arity of f is set a priori. This limits the encrypted data only to be used for a particular computation with a fixed number of inputs. Consequently, it is not well suited for a decentralized environment where we want to reuse the same encrypted data for different computations involving different participants. Adhoc MIFE, on the other hand, is a scheme that uses a combination of MIFE and two-round MPC to obtain a construct that addresses the limitations of traditional MIFE. The setup step of ad-hoc MIFE, which generates a master secret key, is performed independently by each party. Each party then encrypts its data using its own master key. Upon the need to perform a joint computation on encrypted data provided by a set of participants, each participant issues a decryption key derived from the master key as well as the specific function that is used for the computation. This decoupling between encryption and computation is the key contribution of ad-hoc MIFE, and it opens up the possibility for exciting use cases in decentralized situations.

3.2 Blockchain

The issues that need to be addressed are clear from the research into greenwashing and sustainability reporting. We need a way to achieve traceability and transparency within the supply chain. The research question is "How can we achieve this?". Our solution requires a way to store the information from the supply chain in a secure manner. To be secure, it, first of all, needs to be integrity-protected. In other words, the information must be tamper-proof after it has been recorded. Secondly, we need a way to verify the source of the information, by some sort of authentication mechanism. Finally, the information needs to be stored in a resilient and reliable manner in order for us to achieve durability. These requirements should sound familiar to a particular technology that originated in 2008 and has become widely adopted since—*Blockchain*. Blockchain technology is fundamentally based on the use of cryptographic hash structures combined with digital signatures. Since its introduction, it has been widely used in agriculture [LTIW23], supply chain [LHK+22], education [HKGF21] and healthcare [RGK+20; HWH+20; HKG+20] to mention some.

3.2.1 Data structure

A blockchain is a distributed information ledger composed of blocks that are cryptographically chained together through the use of hash functions. This chain of blocks can be viewed as a singly linked list in which each record points to the previous one. The link between the blocks is formed using hash functions. Every block, except for

the *genesis block*—the first block in the chain—includes the hash fingerprint of the previous block as part of its data. By doing so, the hash of a block is recursively dependent upon the hash of all the previous blocks. This ingenious way of employing hash functions makes it impossible to modify a block without needing to sequentially recompute the hash of every subsequent block in the chain, thus providing integrity guarantees. As a result, the longer the chain succeeding the block we wish to modify, the more resources and time it requires to perform the modification.

At the block level, each block is composed of a set of transactions that contain a set of instructions describing the changes the transaction introduces. Transactions are signed before being included in a block using the private key of the transactor. This way, we can authenticate and validate that a transaction is correct with respect to the transactor. The signature provides integrity protection for the transaction and makes the transaction immutable because any modification would require computing a new signature, which would require access to the private key used to sign the transaction.

To summarize, the chain formed by the block hashes provides immutability for the entire ledger and its history, where every new block contributes to the integrity protection of past blocks. Whereas digital signatures authenticate and protect the integrity of the individual transactions contained within a block. Combining hash functions and digital signatures in this way is the breakthrough blockchain made and is the defining characteristic of a blockchain.

3.2.2 Decentralization

As mentioned, a blockchain is a *distributed* information ledger, meaning it is stored and distributed among multiple nodes participating in the network. This is referred to as decentralization because the blockchain is not stored at a single, central node. Through decentralization, we obtain another defining property of blockchain, durability. Durability means that the network is able to withstand faults and errors that could occur at a node, such as a natural disaster or hardware breakdown, without taking down the entire network. Durability is not the only quality attribute that decentralization provides; we also achieve increased availability, because there are multiple nodes that provide access to the blockchain. Contrary to centralized databases, all of the participants of the blockchain, hold the data, which yields transparency and verifiability.

The problem of achieving consensus In a decentralized system where nodes collaborate on maintaining the state of the blockchain, how do we come to an agreement on what the correct state is? A distributed system is chaotic in nature; communication will be unreliable, clocks will be out of sync, and in an open network,

some nodes can be malicious while masquerading as legitimate, and nodes go down unexpectedly. How do we resolve race conditions? For example, if two new blocks are simultaneously proposed to the network. These are all problems that are related to the problem of achieving *consensus* among the nodes of the network.

Different blockchain implementations solve consensus in different ways. The first blockchain implementation called *Bitcoin*, used to proof-of-work (PoW) as its consensus mechanism ([Nak09]). In essence, PoW defines the basis for consensus through computational work. Every block that is added to the chain, needs to satisfy a puzzle that determines how much computational work is on average needed to produce a valid block. Since every block is linked to the previous block, the computational work that went into a particular block is tied to all its predecessors. This way, proof-of-work can determine the correct and valid chain to be that which is the longest in terms of cumulative proof-of-work. However, the biggest problem with this consensus mechanism is that it can consume an enormous amount of power. Other mechanisms such as proof-of-stake and proof-of-space are other consensus mechanisms intended to lower the energy consumption required to secure the network.

3.2.3 Smart contracts

Smart contracts are an essential part of modern blockchains. The term smart contract was coined by Nick Szabo in 1994 [Sza96]. He defined a smart contract as a computerized transaction protocol that executes the terms of a contract. The purpose of a smart contract is to automate processes according to the rules defined by the contract. In blockchain technology, smart contracts are containers of executable code that are stored on the blockchain. Contracts are executed as part of transactions. The underlying blockchain technology defines the capabilities of smart contracts. Bitcoin introduced a simple scripting language that could be used to run smart contracts. However, Bitcoin only allowed the deployment of simple and stateless smart contracts. *Ethereum* [But+14] is a blockchain that was created for running sophisticated smart contracts. It implements a logical computer running on the decentralized blockchain called the *Ethereum Virtual Machine* (EVM). This virtual machine is stateful and its state is derived from the transaction log. By running smart contracts in a stateful virtual machine, smart contracts on Ethereum can read and persist state, opening up the possibility to create stateful smart contracts. The advent of Ethereum revolutionized smart contracts, opening up many new possibilities which have led to numerous applications of decentralized applications.

3.2.4 Storing data off-chain

Storing data off-chain is sometimes necessary due to privacy or confidentiality concerns. Even though data is not stored in the blockchain, we can still get some of the benefits

of the immutability property of blockchains. Instead of storing confidential data on-chain, we store the hash of the data which serves as proof of existence at the point in time the transaction was added to the ledger. In case we need to verify that the confidential data in fact existed in the past, for example for auditing purposes, the hash of the original data can be recomputed and compared against the hash on-chain. Matching hashes indicate that the provided data was the data originally used in the transaction.

3.2.5 Permissioned versus permissionless

The initial implementations of blockchain technology were so-called permissionless blockchains. This means that any entity could join the blockchain network anonymously without the need for a regulating central authority. Major drawbacks of having an open network are that we need a stringent consensus mechanism in order to protect against dishonest participants and the lack of coordination due to the entire network being decentralized and homogeneous limits the scalability of the network as well as reducing cost efficiency.

Permissioned blockchains, on the other hand, are blockchains in which the participants of the network are regulated by one or more central authorities. This allows for a much more flexible trust model as well as a more optimized network structure. For this, we need a way to authenticate the members of the network. In general, this can be solved through the use of PKI. The operator of the network issues digital certificates to each member organization to grant access at the organization level. Then, an organization will further issue digital certificates to its members in order to provision access to the network

In a permissioned network, nodes need not be homogeneous; they can be designated for particular functions, and coordinate amongst themselves to a greater extent, yielding a much more efficient and scalable network. By regulating the participants of the network, they are incentivized to be honest because of the repercussions that could follow from acting maliciously. This allows for a less stringent consensus mechanism, which allows for a more performant and cost-effective network. Therefore, permissioned blockchains are great for collaboration within industries and B2B situations, or for applications where regulation and authentication of members are a requirement, such as online banking.

3.3 Hyperledger Fabric

Hyperledger Fabric[1] is an open-source blockchain operating system part of the Hyperledger Foundation. The Hyperledger Foundation is an open-source collaboration

[1] https://www.hyperledger.org

foundation that is hosted by *the Linux Foundation*. Their mission is to create open-source enterprise-grade blockchain solutions. They host numerous blockchain projects and manage their life cycles. Projects that are classified as *graduated* are well-defined and actively maintained.

Hyperledger Fabric is one of their graduated projects, which is regarded as the first truly extensible blockchain system for running distributed applications [ABB+18]. What makes Fabric unique, is its flexibility by supporting modular consensus protocols. This allows the system to be customized for specific use cases and trust models. The flexibility enables Fabric to be used for a wide variety of applications, making it the de facto standard for industry solutions. One of the key properties of Fabric is its ability to run distributed applications written in traditional general-purpose programming languages. Thus applications that are developed for Fabric can be done so in a traditional way with languages and tools developers are familiar with.

3.3.1 Network architecture

Fabric networks are composed of one or more channels. A channel can be regarded as a *subnet* of the entire network, and only authorized organizations are allowed to join. Each channel has its own ledger, which is comprised of the channel blockchain and state database. In each channel, there are one or more peer nodes which are nodes that are running chaincode and maintaining the state of the ledger. Each peer node belongs to a single organization. In addition to peer nodes which are stateful, there is another kind of network entity called the *ordering service*, which is stateless. The job of the ordering service is to order transactions that are going to be committed to the ledger. Finally, we have the certificate authorities which are network entities that manage the certificates of the channel members. They serve as the source of trust in the network.

An example of a network architecture is given in figure 3.2. In this instance, we have three organizations, `R0`, `R1`, and `R2`. Each of these organizations has its own certificate authority, `CA0`, `CA1`, and `CA2` that is responsible for issuing credentials in the form of digital certificates to the peers belonging to their organization. Depicted in the figure is the channel, `C1`, which all of the organizations have joined. `CC1` refers to the channel configuration of channel 1, which we can see is agreed upon and shared by all of the organizations in the network. Only organizations `R1` and `R2` have peers

3.3.2 Identity management

Intrinsic to permissioned blockchains is the need for identity management and access control. In order to secure the network to ensure that only authorized entities are able to join and participate in the network, we need an authority that issues valid

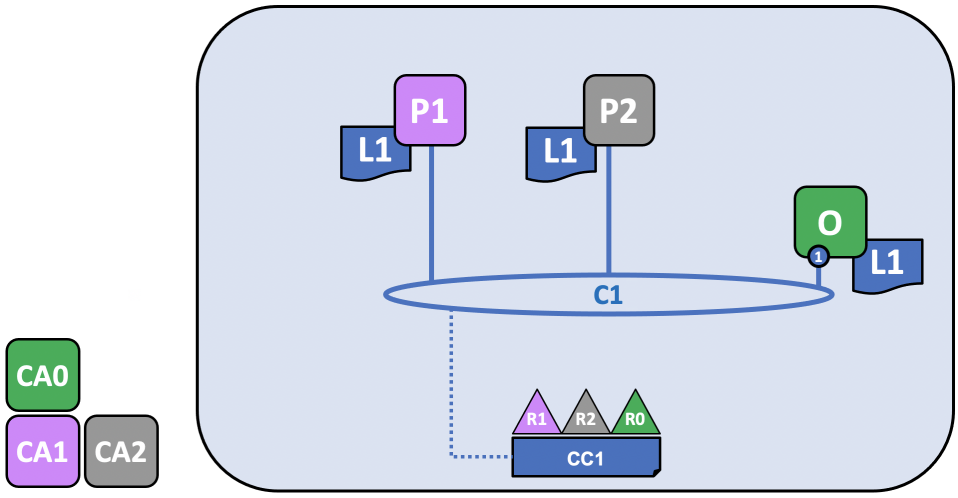

Figure 3.2: Fabric network architecture for an example network consisting of three organizations. Source: [Foua]

identity proofs. In Fabric, this is implemented using digital certificates (X.509) with traditional public-key infrastructure. There are many different kinds of entities in a Fabric network. Some are internal and serve specific functionality to keep the network up and running such as machine entities like peers and orderers, and humans such as administrators. Others are external and interact with the network such as client applications that consume services provided by chaincode. In any case, every entity must hold a valid certificate used for identification as well as additional metadata such as roles and permissions used for authorization. Because Fabric uses PKI, the trust model can be configured according to the needs of the network. Some networks may have one root CA per organization, while others may have a common root CA with intermediary CAs for each organization.

While PKI is used to provision and manage the digital identities of the entities within the network, the *membership service provider* (MSP) is responsible for deciding which identities are allowed to join a specific trust domain. In addition, the MSP is also responsible for mapping identities to their respective role in the trust domain. Fabric has two types of MSPs that differ in their scope and way of implementation, the local MSP and the channel MSP.

Local MSPs are implemented at every node and are defined for clients and nodes of the network. They are implemented as a set of configuration files stored locally at nodes they apply to in the local file system. It defines who is allowed to operate on a

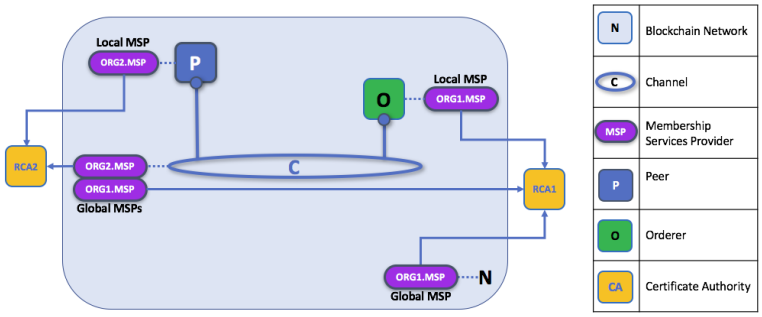

Figure 3.3: Membership service providers in a Fabric Network. Source: [Foub]

particular node, for example, who is a peer administrator.

The channel MSPs, on the other hand, are defined globally in the channel configuration. They define privileges at the channel level, such as permission to participate in the channel, as well as perform administrative actions. Consequently, every organization that participates in the network must have its own channel MSP.

Figure 3.3 illustrates an example network in which we have two organizations that are part of the same channel. Both ORG1 and ORG2 have their own channel MSP and each uses their own CA for identity management. ORG1 owns the ordering node, whereas ORG2 owns the peer node, as indicated by the local MSPs of the nodes.

3.3.3 Ledger

In Hyperledger Fabric, the ledger refers to both the blockchain and the world state. The blockchain is an append-only log containing a chain of blocks that are linked together immutably using hash functions. Because the blockchain is an append-only log of blocks and transactions where writes are the primary operation, it is stored as a regular file. Each of the blocks contains a header that includes the block number, the hash of the block data, as well as the hash of the previous block header. The block data contains a list of transactions that have been ordered by the ordering service. Finally, there is a section of the block that is reserved for metadata and is not included in the hash computations. The metadata contains the certificate and signature of the block creator, along with a bitmap containing a valid/invalid indicator for each of the transactions in the block. The metadata also includes a hash of the cumulative state updates up until and including the block which is used to detect forks in the world state.

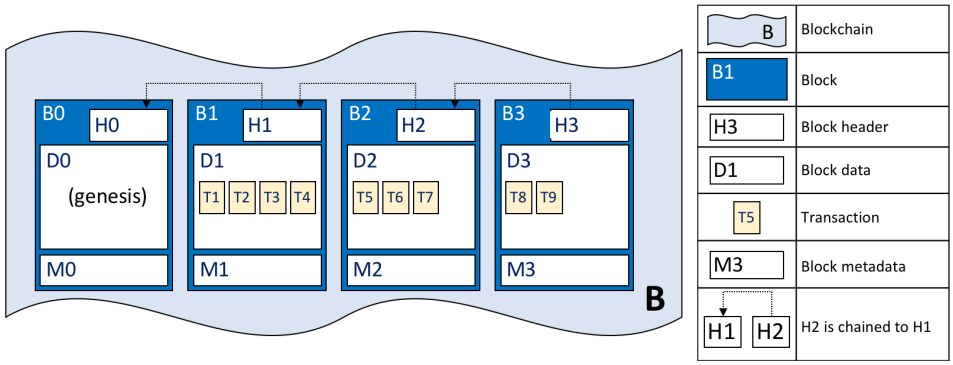

Figure 3.4: Blockchain data structure in Fabric. Source: [Fouc]

Transactions are historic records that capture the *changes* to the world state. They are the source of truth for the world state; should the need arise, the world state can be reconstructed by applying the transactions of the blockchain in order. Figure 3.5 displays the transaction format in Fabric. Let us break down each of the fields. Similarly to the block format, every transaction contains a header, which contains metadata about the context of execution, such as the name and version of the chaincode that proposed the transaction. The signature field contains a digital signature of the transaction details that was generated by the client application invoking the chaincode. Since the signature was generated using the private key of the client application, it provides integrity protection, protecting the transaction against tampering. Input parameters that were supplied to the chaincode that generated the transaction are stored in the proposal field. After the chaincode has been executed, the content of the response field is generated. It contains the read-write set (rw-set) which describes the world state before and after the execution of the chaincode. Finally, the endorsements field contains a list of signed transaction responses from the endorsers. It is used to check that the endorsement policy of the involved keys is satisfied.

The world state represents the current state of each of the keys up until and including the transactions from the latest block. In other words, the world state is a projection of the sequence of transactions in the blockchain. It is stored separately in a key-value database outside of the blockchain. The reason for storing the current state in a separate key-value database is for improved performance as well as the possibility of storing off-chain data, which we will look further into when discussing privacy features. When chaincode is executed, it uses the world state when performing

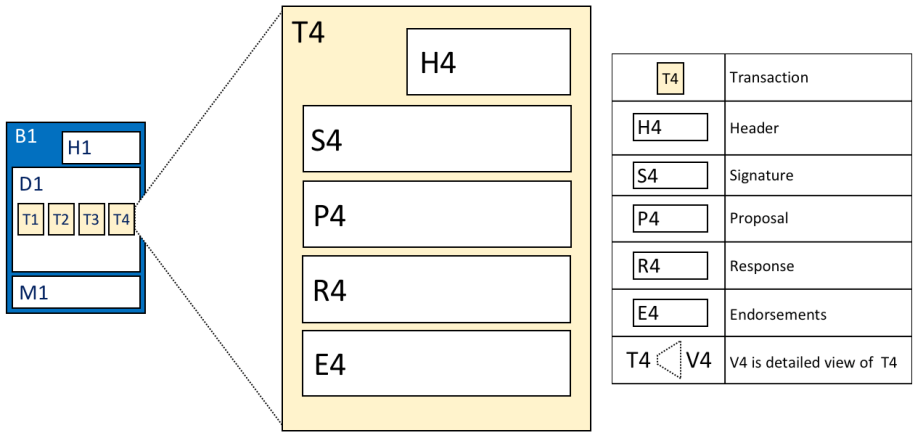

Figure 3.5: Transaction structure in Hyperledger Fabric. Source: [Foud]

data reads and writes, which is much more efficient using a database optimized for key-based lookups instead of the blockchain. Fabric natively supports either using *CouchDB* [23a] or *LevelDB* [23b] for the world state. CouchDB provides more advanced indexing functionality than LevelDB, which can be used perform to complex queries.

3.3.4 Transaction architecture

The transaction architecture of Hyperledger Fabric is the major differentiator of Fabric compared to prior blockchains. Historically, all blockchains have used a *order-execute* transaction architecture. As the name implies, transactions are first ordered, then they are executed and state changes are persisted. There are major drawbacks to this architecture. Since transactions are ordered before being executed, the transactions have to be deterministic and executed sequentially at every node. Consequently, we are not able to parallelize transactions and hence we cannot scale horizontally.

Fabric introduces a new transaction architecture called *execute-order-validate*. In contrast to *order-execute*, transactions are executed by first being simulated at a set of required nodes, called the endorsing nodes, defined by the endorsement policy. The transaction simulation results in a read-write set that contains the sets of keys that are read and written by the chaincode. These read-write sets are digitally signed using the private key of the endorsing peers, which ensures the integrity and authenticity of the transaction simulation. Each endorsing node sends its rw-set to the ordering service for ordering. After ordering has been completed, the ordering

service broadcasts the rw-set along with the digital signatures of the endorsing nodes to all peers of the network. Finally, each peer validates the transaction by ensuring there are no conflicts in the rw-set provided by each endorsing peer, as well as validating that the endorsement policy has been met by examining the provided signatures. Once validated, the state of the ledger is updated in each node according to the rw-set.

Figure 3.6 illustrates the execute-order-validate architecture of Fabric. In this example, there is one client that begins the transaction flow by submitting a proposal, three endorsing peers that are responsible for transaction simulation, one ordering node, and one committing peer. We will now give a detailed explanation of each step:

1. Signed client proposal is sent to each of the endorsing peers.
2. The result from the simulation executed at each endorsing peer is sent back to the client, and the endorsements are collected according to the endorsement policy.
3. Final endorsements are broadcast to the ordering service.
4. Ordering service orders the transaction into a block and broadcasts the block to all peers. Each peer individually verifies that the endorsement policy is satisfied, and appends the block to their copy of the blockchain. Finally, the readset is validated and not in conflict with other transactions, the writeset changes are committed to the world state database.

By decoupling the transaction execution from ordering, transactions can be executed in parallel, resulting in the ability to scale the network horizontally. This allows Fabric to scale to thousands of transactions per second, depending on the configuration, making it a great blockchain for industry solutions.

3.3.5 Chaincode

Chaincode refers to an application that is deployed to a Fabric network. Specifically, it is a program written in one of the supported languages that have been packed inside a docker image and deployed to a specific channel of the network. Every peer that is a part of the channel, runs their own instance of the chaincode within a docker container. Chaincode is the business logic of the blockchain application and is analogous to what is usually referred to as *smart contracts* in other traditional blockchains. Traditional blockchain systems are usually limited to simple domain-specific languages that provide simple instruction sets. Programs written and stored

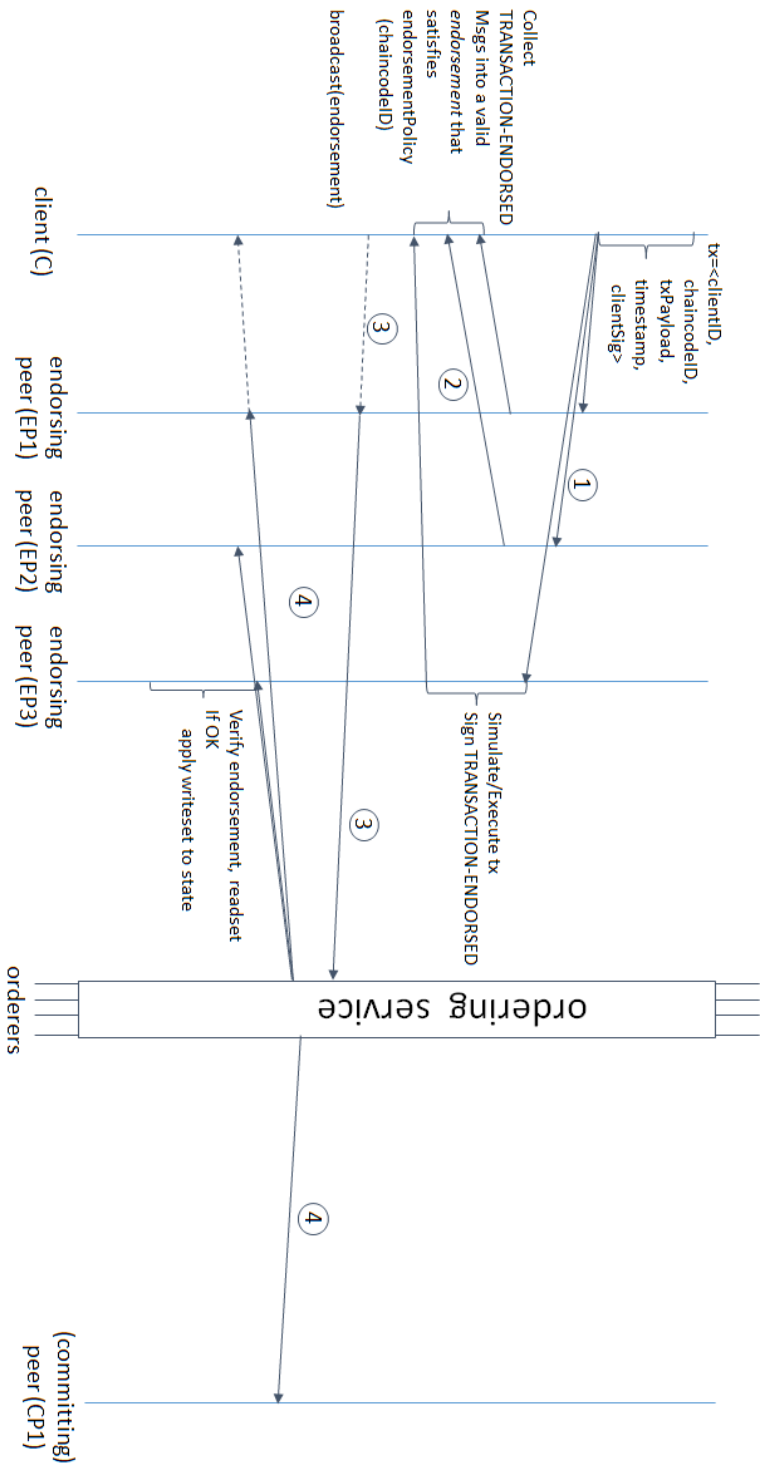

Figure 3.6: Sequence diagram of execute-order-validate architecture. Source: [Foue]

on these blockchains, often referred to as scripts, are usually very expensive to execute, often relying on cryptocurrency as the means of paying for program execution. In addition to being expensive, scripts are limited in their size and must be composed of relatively few instructions, compared to traditional programming languages, due to the need for sequential execution at every node. Fabric, on the other hand, supports general-purpose programming languages such as *JavaScript*, *Go*, and *Java*, to name a few. This is thanks to Fabric's architecture and protocol for executing transactions. Complex blockchain applications are no longer a luxury or impossibility, but cheap and efficient to implement on Fabric.

3.3.6 Privacy

Since Fabric is a blockchain for industry applications where different entities that do not necessarily trust each other may be participating, privacy preserve features are a big part of the toolkit provided by Fabric. Two key privacy features provided are called *private data collections* and *transient data*.

Private data collection (PDC) Private data collections are collections of data that are stored off-chain, while the hash of the data is stored on-chain serving as a proof of existence at the point in time. If the need should arise, the pre-image of the hash can be revealed and the hash can be recomputed and we can verify that it matches the hash that is on-chain. PDCs are bound to organizations, meaning that the data of a PDC is only stored in the nodes belonging to the owners of the PDC—as illustrated by figure 3.7. Consequently, in order to modify or query the data stored in a PDC, the peer must belong to an organization with access to the PDC. Each organization has a built-in PDC, that requires no additional configuration or setup, referred to as the implicit private data collection.

Another feature that Fabric provides is the ability to prune private data collections. Pruning a key that is stored in a PDC means removing the state associated with the key. This is useful if we need to remove sensitive or personal information due to compliance. Even though the state is permanently removed, the hashes of the state are still a part of the blockchain.

Transient data Often used in combination with private data collections is the concept called transient data. When invoking a chaincode function, a field called *transient* can be sent along with the regular function arguments. Unlike the regular arguments, the value of the transient field is not included in the transaction and is thus not stored on-chain. The data is only available during the execution of the chaincode. During a chaincode call, the transient data is only available to the peers that are endorsing the transaction and is never persisted. This, combined with using

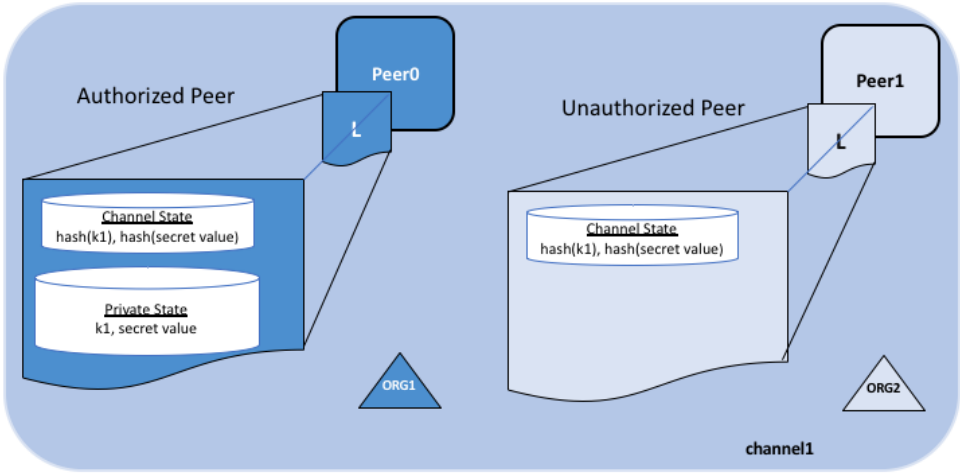

Figure 3.7: Private data collections are bound to organizations and are stored off-chain. Source: [Fouf]

private data collection for storage, enables us to write confidential data to Fabric, without revealing it to other network participants.

Chapter 4
Modeling and requirements

Before embarking on the journey of implementing the end solution, we first need to establish the needs and requirements that are expected to be met. In this section, we will define the relevant stakeholders and their concerns which will set the guidelines for the requirements. Some of these requirements will be architectural drivers that will impact the system architecture.

4.1 Stakeholders and concerns

Stakeholders constitute the different actors that are involved in a software project. They have different roles, incentives and perspectives of the resulting product, and their needs and concerns are translated into functional and quality requirements of the system. The stakeholders that we present in this section follow from the background research into the greenwashing domain and sustainability reporting, as illustrated in Figure 2.2.

4.1.1 Developers

The developers are those who contribute to the solution by writing and maintaining the code. Their primary concern is modifiability and extensibility. As the functional requirements evolve and new features are requested, the solution should be easy to extend to fulfil the new requirements. When existing requirements change leading to the need to modify existing functionality, this should be as frictionless as possible; i.e., the functionality should be packed into cohesive software components in a well-defined architecture, thereby minimizing coupling and maximizing modifiability.

4.1.2 Organizations

Organizations, specifically reporting businesses, are the primary users of the blockchain solution. Their primary concern is the security of the system, due to the nature of the data that they are recording in the system. Corporate secrets and data that

can reveal competitive edges must be protected against other actors, especially from competitors. The system is intended to be used by multiple types of businesses within different industries with varying degrees of technical competence. Usability is therefore an important focus point. Because the system is decentralized and each reporting business runs their own instance of the blockchain, the deployability of the system is also an important aspect that must be addressed, to ensure the lowest barrier to entry.

4.1.3 System operators

The system operators are the providers and administrators of the blockchain solution. Their primary role is management of the members of the network. This involves issuing, revoking and renewing digital certificates used to authenticate and identify the network participants. Operators could be governments at the national or international level. Their primary concern is first and foremost the security of the system. The confidentiality of the private data must be maintained, and the integrity of the recorded data must be maintained. In the event of a partial loss of nodes in the network, the system should endure, and no data is lost. This property of survival is referred to as durability. If the system is intended for use by a large quantity of businesses, it is vital that the network is able to scale with the increase in users.

4.1.4 Non-governmental organizations

Non-governmental organizations (NGOs) are potential users of the blockchain that will not necessarily use it to record business activities and report sustainability information. They are more concerned with being passive observers in the network, such as looking at the environmental footprint of a business or products they sell. Their primary concern is the integrity of the data they are presented.

4.1.5 Consumers

Consumers are the people that are purchasing goods and services from businesses. Similarly to the NGOs, they are merely observers and not submitting any data directly to the system. Consequently, they are also primarily concerned with the correctness and integrity of the data, such as the environmental impact of a particular product. The general public does not possess the technical competence to use a low-level blockchain application. Hence they will need some other party to implement a frontend through which blockchain data can be accessed.

4.2 Business requirements

Business requirements are the highest level requirements from which the functional and quality requirements follow. They take into account the different stakeholders, the

product vision, and other business-related aspects such as cost and time to market. In this project, we are the "business", and the product is the experimental implementation we are developing. The experimental solution aims to bring transparency to the supply chain and create the basis for a standardized system with which businesses can report on sustainability. We envision an open system that is not owned by any single entity but rather maintained and operated democratically by system operators. It would be done using an open-source model, where everyone could contribute and potentially positively impact sustainability. Any reporting business enrolled in the system can use it to report business activities, register assets, and their interactions with other companies. The data reported by any business must be protected and impossible to access neither by other companies nor by the system operators. We need a way to jointly compute aggregated data using data from multiple companies in order to calculate upstream and downstream sustainability impact while maintaining the confidentiality of the individual data inputs.

In the ideal world, every business in every part of the supply chain uses a shared system to track its sustainability impact. If that were the case, we could accurately determine the sustainability impact of products and services businesses provide. This would be helpful in both business-to-business and business-to-consumer cases—businesses could get insights that would help them make sustainable choices, investors could make better decisions by investing in more sustainable companies, and consumers could gain insight into the environmental impact of end products and services they purchase. As a result, it would be hard for businesses to perform greenwashing.

The idea of having an open system that every business would use is impossible to realize in practice. Nevertheless, it is not unlikely that there will come a time when most industrial countries require accurate reporting by the majority of businesses, and not only the biggest enterprises, which is the case today within the EU. If we are going to have a system that is feasible to use on a large scale, it needs to be cost-efficient, highly scalable and reliable, and easy to integrate. Of course, it will never be the case that every small business would be able to integrate directly with such a system. However, this could open up future business opportunities where some specialize in providing sustainability reporting as a service by reporting into the shared system on behalf of smaller businesses. Larger enterprises would likely prefer integrating directly with the system to ensure their data is secured without relying on a trusted third party.

4.3 Trust model

The solution is not intended to be used by a single governing agency. Therefore, we propose a trust model in which multiple independent parties—be it national or international governing agencies—collaborate on operating the blockchain. This is implemented by having one root CA for each operator. Subsequently, each operator can decide on the trust model in their own trust scope. A network operator must invite every organization that wishes to join the network. They will have their own CA validated by the CA of the operator to which they belong. The operators are responsible for authenticating the business wishing to join through external means outside the system. Businesses are free to select further their trust model, which is scoped at the organization level. This is especially useful for larger organizations that want to subdivide their trust hierarchy based on departments within the business. Ultimately, we have a flexible trust model bound to a single central authority but a decentralized collaboration. Figure 4.1 displays an example trust hierarchy with three operators that follows the proposed trust model.

4.4 Functional Requirements

The functional requirements refer to requirements that define the behavior and use cases of the system. Functional requirements are user oriented and their goal is solving user needs and business problems. These requirements are based on the background research that we conducted during the early phase of the thesis work. From it, we gained an understanding of the domain of sustainability reporting. The most central concern for our solution is environmental reporting, specifically the reporting of greenhouse gases. We also looked into existing corporate sustainability reporting standards to understand how they define the reporting process. With our solution, we want to obtain the greatest level of accuracy, which is achieved by granularity, reporting at the business activity level instead of the holistic organizational level.

This solution also tries to address the problem of tracking emissions downstream and upstream within the supply chain by allowing users to register mutual interactions between businesses and linking assets to the activities they were produced from, consumed by, and involved in. By doing so, we can establish a relationship between assets and their usage, which forms an activity log that can be used to link sustainability impact to assets across the supply chain. It also provides an audit trail for each asset, from the sourcing of raw materials to the finished end product, referred to as *cradle-to-gate*. The implementation of this functionality is entirely hypothesis based as there does not seem to be any existing solution that provides this way of tracking emissions at this level of granularity between multiple actors in the supply chain.

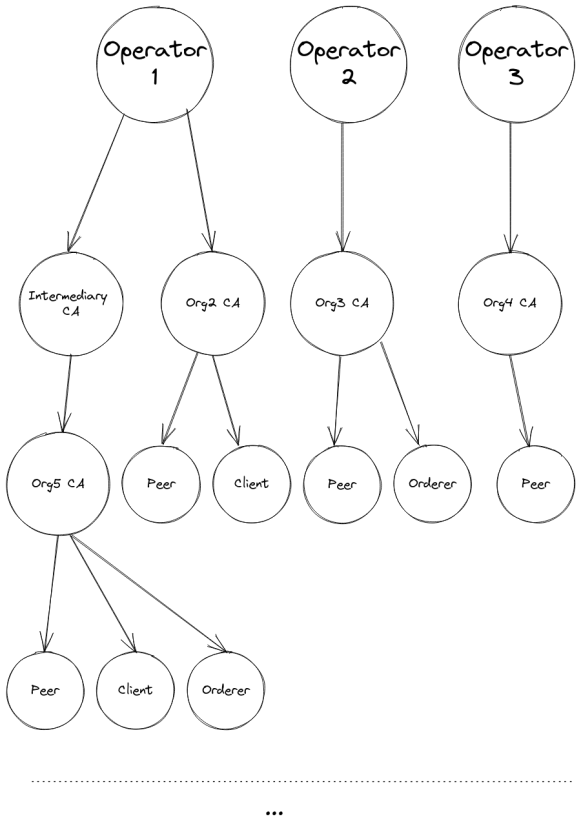

Figure 4.1: Trust model for the implementation.

Table 4.1: Functional requirements for the application

ID	Requirement	Comments
FR1	Reporting business should be able to register business activities and their associated environmental sustainability impact defined as a set of outcomes.	This is the core use case.
FR 1.1	When registering a business activity, assets that were produced from the activity can be specified. These assets will subsequently be recorded in the system and associated with production activity.	
FR 1.2	When registering a business activity, assets that were consumed or partially consumed can also be specified. This will update the associated assets and register the consumption activity.	
FR 1.3	An activity can be composed of other activities to represent a larger process. Thus, sub-activities can also be specified during the registration.	
FR2	Reporting business should be able to register business assets.	
FR2.1	If the business asset is composed of other assets, component assets can be specified (either as a fraction or as a whole), which will consume the provided assets (FR7).	In the case where the provided components have already been consumed, the registration fails.
FR2.2	The business activity that produced the asset can be specified, which will be added to the asset record.	
FR2.3	During asset registration, activities that the asset has been a part of can be specified.	
FR2.4	Activity(ies) that have (partially) consumed the asset can be specified, which will be added to the asset record.	
FR3	Two organizations are able to register an interaction between them.	Required for establishing links between organizations.

ID	Requirement	Comments
FR3.1	Either of the organizations involved in the interaction can re-propose the interaction with different details to resolve errors.	
FR3.2	Only the organization that received the last proposal is able to confirm the interaction.	For example, if A proposes to B, then B is only able to confirm the interaction unless B issues a re-proposal, in which case A can confirm, but B cannot anymore.
FR3.3	Before the interaction has been confirmed, the party who did not propose the last proposal can decline the proposal.	
FR3.4	Before the interaction has been confirmed, the proposer can cancel their proposal.	
FR3.5	Transfer of assets can be specified in the interaction.	Interactions can be used for any type of business interaction, such as a trade, sale, or donation, where the transfer of assets could happen.
FR4	Asset ownership can be transferred to another organization.	Ownership management of assets is not a high priority.
FR5	View all interactions.	Organizations should only be able to view interactions they are a part of.
FR5.1	View incomplete interactions.	
FR6	View business activities.	Organizations can only view their own activities.

ID	Requirement	Comments
FR7	Assets can be consumed by activities or upon registering a composite asset.	Assets consumed by activities can be used to model the processing of materials into a new end product, in which case new assets are produced. Composite assets are assets that are composed of other assets, for example, a computer containing computer parts.
FR8	View total sustainability impact.	Details about what is indirect (upstream/downstream) should also be returned.
FR9	View asset sustainability impact.	The asset sustainability impact is calculated by taking the average of the impacts of each activity the asset (and its component assets) has been involved in along the supply chain.

4.5 Domain model

Before starting to implement the code of the solution, we are going to take a model-driven approach using domain-driven design. We start by identifying the entities and their relationships based on the functional requirements that were defined previously (see Table 4.1). From this, we have constructed a high-level model reflecting the domain model which is required to implement the business logic defined by the requirements.

We present the model in Figure 4.2. Each circle represents an entity or a value object, whereas the diamond represents a relationship between entities. A detailed view of the components displayed in the model is given in Table 4.2, which lists each component, its type, and the description of what it represents.

Table 4.2: Explanation for the domain model components presented in Figure 4.2.

Component	Type	Description
Organization	Entity	Reporting organization that is registered in the system.
Activity	Entity	Entity representing a business activity belonging to one or more organizations.
Outcome	Value object	Model of an outcome related to sustainability. The set of outcomes for a particular activity represents the total impact an activity has. There are different types of outcomes representing events such as the emission of greenhouse gases, the consumption of electricity, and so on.
Asset	Entity	Representation of an asset owned by a business, whose ownership can be transferred. An asset can either be atomic or composed of multiple constituent assets.
Interacts	Relationship	Represents an interacts relationship between two organizations. Cardinality is Many-to-many (M:M).
Involves	Relationship	As part of an interaction between two organizations, we can define activities that are part of the interaction, which will subsequently be registered at both organizations. Used to represent a collaboration activity. Cardinality is 1:M
Produces	Relationship	When registering an Activity, we can specify assets that were produced by the activity. An asset can only be produced by one production activity, hence cardinality 1:M
Uses	Relationship	Represents the use (not consumption) of an asset in a business activity, e.g. a diesel engine to perform work.
Consumes	Relationship	Some activities, such as production or end-of-life activities, may require the consumption of assets, for example, the processing of raw materials that result in a new product.

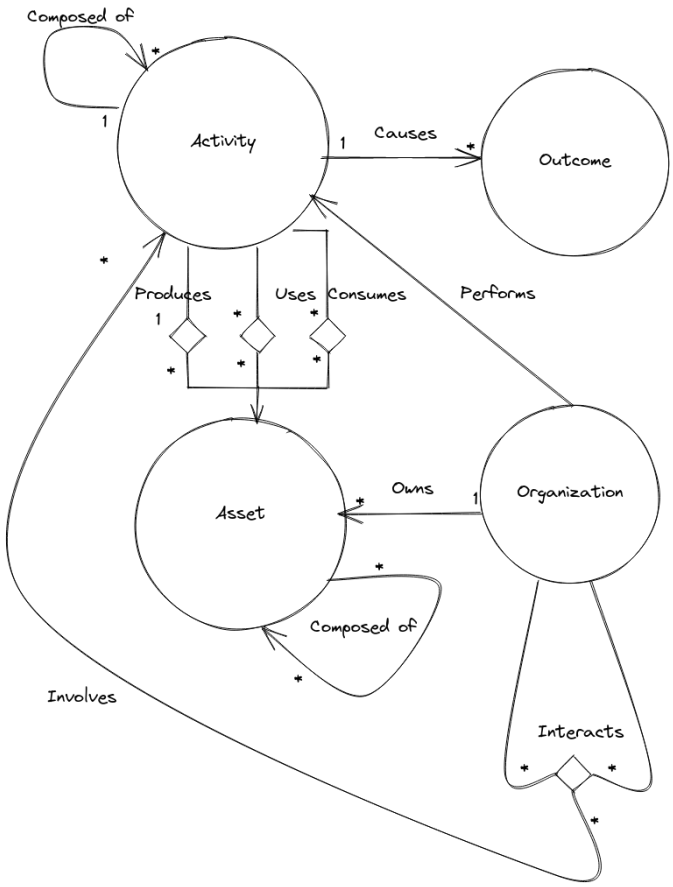

Figure 4.2: Model of the sustainability reporting domain.

4.6 Quality requirements

Quality requirements are nonfunctional requirements for the system. Unlike functional requirements, they do not define business logic requirements that solve problems related to the domain. Instead, they define requirements for the quality attributes of the system, usually related to performance, security, and availability. The quality requirements follow directly from the needs defined by the involved stakeholders. Different stakeholders have different roles and responsibilities for the system and are therefore concerned with different aspects of the system.

Developers are concerned with all of the requirements defined because they are the ones that are making architectural choices that can impact the ability to satisfy the requirements. Whereas the stakeholders responsible for system operation are more concerned with the quality attributes related to cost, scalability, and the security of the system. Meanwhile, the users that will be inputting data into the system, i.e. the reporting businesses, are mostly concerned with the security, reliability, and usability of the system. In this section, we will present the quality attribute requirements by defining quality scenarios.

4.6.1 Security scenarios

Table 4.3: Security scenario 1: Registering reporting data securely

ID	**S1**
Source	Reporting business
Stimulus	Submits reporting data to the system
Artifacts	Blockchain application, client, internet connection
Environment	Runtime
Response	Data is encrypted in transit using authenticated encryption. The confidential data is stored off-chain (hash of data stored on-chain), only accessible to the reporting business
Response value	Confidentiality, availability, and integrity maintained.

Table 4.4: Security scenario 2: Secure data aggregation

ID	**S2**
Source	Reporting business
Stimulus	Wants to perform a use case which requires aggregating data across their supply-chain
Artifacts	Blockchain application, client
Environment	Runtime
Response	Relevant data is shared by each organization along the supply chain. The provided data is encrypted and unavailable by itself. The encrypted data is aggregated using a mechanism that preserves the constituent data.
Response value	Confidentiality is maintained.

Table 4.5: Security scenario 3: Privacy preservation after asset transfer.

ID	**S2**
Source	Reporting business
Stimulus	Transfer asset to another business in the system
Artifacts	Blockchain application, client
Environment	Runtime
Response	Asset record is added to the new owner and is only accessible to the new owner. Activity records belonging to the previous owner is not passed on to the new owner.
Response value	Confidentiality, integrity, and availability maintained.

4.6.2 Extensibility scenarios

Table 4.6: Extensibility scenario 1: Add new sustainability outcome type.

ID	**E1**
Source	Developer, System operator
Stimulus	Wants to add a new sustainability outcome type
Artifacts	Code
Environment	Development time
Response	New value class is implemented
Response measure	Minutes to implement

Table 4.7: Extensibility scenario 2: Implement a new query

ID	**E2**
Source	Developer, System operator
Stimulus	Wants to add a new query to an existing smart contract
Artifacts	Smart contract code
Environment	Development time
Response	New function is implemented and tested
Response measure	Minutes to implement and test

Table 4.8: Extensibility scenario 3: Add a new contract

ID	**E3**
Source	Developer, System operator
Stimulus	Wants to implement a new smart contract
Artifacts	New contract code
Environment	Development time
Response	New contract is implemented and tested
Response measure	Hours to implement

4.6.3 Modifiability scenarios

Table 4.9: Modifiability scenario: Modify existing command by introducing the new business rule.

ID	**M1**
Source	Developer
Stimulus	Wants to add a new business rule to existing command due to new business requirement
Artifacts	Command code in smart contract
Environment	Development time
Response	Existing command behavior is modified to satisfy new requirement
Response measure	Minutes to modify and test new behavior

4.6.4 Usability Scenarios

Table 4.10: Usability scenario 1: Enroll in the blockchain network.

ID	**U1**
Source	Reporting business
Stimulus	Business wishing to join the network is able to read the technical documentation and enroll in the network after being issued credentials from their operator.
Artifacts	Blockchain solution and documentation.
Environment	Run-time
Response	Provide user with well written documentation.
Response measure	Hours to setup and enroll.

4.6.5 Availability Scenarios

Table 4.11: Blockchain node failure.

ID	**A1**
Source	Node
Stimulus	A node part of hosting the blockchain fails and becomes unusable.
Artifacts	Node
Environment	Normal operation; runtime
Response	Other nodes provide redundancy and keeps the network operational, and the broken node is replaced.
Response value	No downtime

4.6.6 Recoverability scenarios

Table 4.12: Recoverability scenario: Transient network failure

ID	**R1**
Source	Node
Stimulus	Network connection is lost
Artifacts	Network Interface Card (NIC) in blockchain node.
Environment	During runtime
Response	Node is on stand-by until network connectivity is regained. Once back online, the node will recover by synchronizing with the rest of the nodes.
Response value	Recovery is successful within hours, depending on the duration of the failure.

4.6.7 Deployability scenarios

Table 4.13: Deployability scenario: Deploy a new version of blockchain application.

ID	**D1**
Source	System operator
Stimulus	Wishes to deploy a new version of the blockchain application.
Artifacts	Blockchain application, blockchain network
Environment	During runtime
Response	New version of blockchain application is deployed to the network and the old version is not used anymore.
Response measure	Minutes to deploy new version.

4.6.8 Scalability scenarios

Table 4.14: Scalability scenario: Deploy a new version of blockchain application.

ID	**SC1**
Source	System operator
Stimulus	Wishes to deploy a new version of the blockchain application.
Artifacts	Blockchain application, blockchain network
Environment	During runtime
Response	New version of the blockchain application is deployed to the network and the old version is not used anymore.
Response measure	Minutes to deploy new version

4.7 Architecturally significant requirements

Architecturally significant requirements (ASR) are the requirements, functional or quality attribute related, that impact the system architecture. They need to be uncovered in order to establish the architectural drivers for our solution. Usually, of most concern to the system architecture, are the quality requirements, which we defined in terms of the quality attribute scenarios. However, functional requirements can also have a significant impact. We present the ASRs for our implementation in Table 4.15. The ID of the first column corresponds to the ID of the requirement as presented in the overview of functional requirements (Table 4.1) and the quality scenarios (Tables 4.3-4.14).

4.8 Architectural drivers

Architectural drivers constitute the requirements that force us to make decisions for the architecture in the early phase of the design process. It is paramount to map these out accurately so that we can make informed decisions, avoiding costly mistakes. We are going to list the drivers from each category of requirements, in the order from most high-level—the business requirements—to the low-level —quality requirements.

4.8.1 Business requirements

The business requirements that have a great impact on the architecture, are mostly concerned with the system architecture. As we are going to be implementing a solution using blockchain technology for the domain of sustainability reporting, it is

Table 4.15: List of architecturally significant requirements and explanations.

Requirements	Explanation
S1, **S2**	For us to be able to preserve confidentiality and let each reporting business store their confidential data off-chain and be able to retrieve it seamlessly, we need a blockchain that supports this type of data model. Furthermore, each business needs to provide its own nodes that will store confidential data.
E1, **E2**, **E3**	In order to have the application be extensible, we need a software architecture that allows to seamlessly add new functionality without additional overhead.
D1	There are many different kinds of blockchains that differ in how they implement the deployment, especially redeployment/upgrade of smart contracts. This requirement requires us to select a blockchain that allows for ease of deployment.
SC1	The scalability requirement imposes restrictions on what blockchain we are able to use. Most blockchains struggle with scalability due to the nature of how the blockchain state is maintained.

important that we select a blockchain that is sustainable. This means we cannot use a blockchain that uses consensus mechanisms that require lots of resources, such as proof-of-work, which is not energy efficient.

Not everyone is going to have access to the system; only verified businesses performing sustainability reporting, system operators as well as NGOs. Consequently, we must have a system that authenticates its users, limiting us to the class of permissioned blockchains. Furthermore, based on the trust model we have defined, we need a system that is able to fulfill this requirement.

Perhaps the most important business requirement we need to fulfill in order for the possibility of the solution to be adapted en masse is the metric all businesses are concerned with—cost. If the system is costly to use, it will be hard to incentivize businesses to use the solution, which is in direct contradiction to the economic pillar of sustainability, making the solution itself unsustainable.

4.8.2 Functional requirements

Functional requirements define the complexity of the business logic that our solution is going to implement. Since this business logic will be implemented using smart contracts on a blockchain, it sets requirements for the programming model of the

selected blockchain. We cannot use a blockchain that is overly restrictive in its ability to implement smart contracts. That would constitute both a short-term risk for the implementation of the current functionality (Table 4.1) as well as in the long term. It is hard to predict the evolution of the solution in the future as new needs and requirements may arise. Thus, we need a blockchain that allows us to scale our application complexity. Moreover, our application is stateful, which means our blockchain should provide the ability to query and persist state.

4.8.3 Quality requirements

Notorious for having a great impact on the architecture are the quality requirements. As they are cross-cutting concerns, they usually impact both the system architecture as well as the software architecture. We previously defined the quality requirements of most significance in our uncovering of the architecturally significant requirements (Table 4.15). They are, by definition, architectural drivers. However, the ASRs are not the only quality requirements that drive the resulting architecture. We are going to walk through each quality attribute whose defined requirement scenarios impact the architectural decisions.

Modifiability and extensibility Starting out with the quality attributes that impact the software architecture, are modifiability and extensibility. In order for us to satisfy the requirements defined in Tables 4.9 and 4.6-4.8, we need a software architecture that is organized in a clear structure that minimizes coupling and maximizes cohesion. In addition to implementing the appropriate software architecture, we can use design patterns at the component level to further improve the structure of the code.

Security The security requirements are cross-cutting and impact both the system architecture as well as parts of the software implementation. We will walk through the affected areas, going from the lowest-level details upwards, starting out with the network communications.

Network traffic that leaves the trust boundary of the system and is transmitted over the internet has to be encrypted and authenticated. Otherwise, we are vulnerable to various attacks such as man-in-the-middle and wiretapping, which would compromise confidentiality, and tampering which would compromise the integrity of the data.

Because a blockchain is a distributed ledger that is replicated to all participating nodes, we cannot store confidential data directly in the public ledger, as it would be available to everyone. Hence, we need a mechanism by which we can selectively store confidential data off-chain, while still achieving the cryptographic properties provided by the blockchain. This can be achieved by using a separate data store

for off-chain data while storing the hash of the confidential data on-chain as proof of existence. Another way is to store the data encrypted on the blockchain, but unfortunately, this will make it hard to query the data.

Scalability One of the most criticized points of the current widely used blockchains is their ability to scale. Two of the most successful blockchains today, *Bitcoin* and *Ethereum*, only have a throughput of 3-4 and 15 transactions per second (TPS), respectively. Furthermore, the read performance of blockchain databases is worse than that of non-blockchain databases like YouTube and Google. The reason for the challenges in scalability is the trade-off between performance, security, and trust [SC21]. These findings are concerned with public and open blockchains. However, as we are going to be using a permissioned blockchain, we have greater flexibility for the trust assumptions that heavily influence the scalability.

Chapter 5
Experimental implementation: Gaia

In this section, we will present the experimental implementation that has been the main contribution of this thesis. The name of the implementation is *Gaia*.[1] We will start by presenting a high-level description of what the implementation provides and the process through which it was developed. The code is hosted at GitHub: https://github.com/udnes99/Gaia/tree/master-thesis. There, the reader can also find technical documentation. The master branch contains the most up-to-date code, and the *master-thesis* branch contains the code from the thesis.

5.1 Description

The solution we present is a sustainability reporting application running on the Hyperledger Fabric blockchain operating system. The main features of the application are the ability to record business activities and their environmental impact. These activities can also be associated with the production and consumption of assets belonging to the business. Using this information, we can calculate the average emissions for a given asset. Another feature this solution provides is the ability for businesses to collaborate by registering interactions between them. This could be used to register involvement in an activity that is performed by another organization, such as the usage of a service like transportation. By recording the trail of activities each asset is involved in, we can calculate both the upstream and downstream impact for an asset.

5.2 Blockchain selection

In the previous chapter we established the requirements for our solution, and we determined which were architectural drivers. Many of these drivers imposed requirements on the blockchain of choice. The methodology for the selection process

[1] Gaia is a goddess in Greek mythology that resembles the personification of the earth.

was researching the current state-of-the-art of permissioned blockchains and looking at the potential candidates. From our research, we found Hyperledger Fabric by the Hyperledger Foundation to be the most promising candidate. Fabric is an open-source blockchain operating system that can be used to deploy distributed ledger applications implemented in familiar general-purpose languages. This stands in great contrast to most blockchains that provide a limited programming ability. Its innovative transaction architecture offers performance at scale, as well as supports a flexible trust model through its network architecture that uses the public-key infrastructure for identity management. To ensure that Fabric is indeed the correct choice, we will examine its ability to fulfill the requirements.

5.2.1 Cost

Since Fabric is a blockchain operating system and not a public blockchain that relies on a native cryptocurrency to pay for the execution of smart contracts, the cost is inherently the cost of the infrastructure. The cost of executing a transaction depends on the resources used to carry out the execution, and it scales with the complexity of the business logic. In that sense, it is similar to a regular distributed application. Unlike a normally distributed application, however, the transaction log is replicated and stored at every peer that is a part of the network. The transaction log is stored as a regular file, and the space complexity is $O(n)$ where n is the total number of transactions that have been executed. In addition to the transaction log, the world state also needs to be persisted at every node. Luckily, storage is inexpensive, but in a scenario where the network grows large, we risk the barrier for some businesses to directly join the network to become too large. However, if some decide to join as reporting service providers, they can provide access on behalf of several businesses, which removes infrastructure-related barriers.

5.2.2 Sustainability

Sustainability is a primary concern that the selected blockchain needs to fulfill, otherwise, we compromise the intent of the solution, to begin with. Because Fabric makes trust assumptions, unlike open blockchains, our consensus mechanism does not need to rely on resource-demanding algorithms for security. Traditional cryptocurrencies that use proof-of-work, have had an exponential increase in the power consumption required for mining, due to their competitive nature. In Fabric, however, the resource usage scales in a similar fashion to regular distributed applications; linear in the throughput of the system. However, unlike regular distributed applications that are often deployed to the cloud, where the region of deployed nodes can be decided based on factors like carbon footprint, the nodes in Fabric are provisioned by the participants of the network. This means that the sustainability impact of each node varies with its location and underlying infrastructure, which in turn determines the

source of electricity and energy efficiency. This is the best we can hope for in a decentralized environment.

5.2.3 Security

The security of the system is paramount; the network needs to be secure, and confidential data cannot be stored on-chain. In Fabric, every entity that interacts with the network, either from within or outside, is authenticated using digital certificates. The security of this scheme depends on the underlying trust model which is implemented using PKI. Network traffic is encrypted using TLS, preventing data from being compromised in transit. Organizations using the solution will be recording their business activities and assets, as well as interactions with other organizations. In other words, highly confidential data. This data cannot be stored on-chain, as it would be accessible to everyone in the network. Fabric was designed with these kinds of use cases in mind and provides privacy features to address this. Private data collections allow organizations to store their private data off-chain in the world-state database of the organization's peers. This is inaccessible to other organizations unless they connect to a peer belonging to the owner and are explicitly granted access. In conclusion, Fabric satisfies our security requirements.

5.2.4 Scalability

Fabric is intended for use in enterprise-grade applications that must perform at scale. This is made possible by its unique transaction architecture, which is one the biggest differentiators of Fabric compared to prior blockchains. The *execute-order-validate* architecture allows transactions to be executed in parallel by a subset of the nodes of the network as defined by the endorsement policies. In other words, the transactions are sharded based on the endorsement policy. As a result, transaction throughput scales horizontally with the number of nodes in the network.

Transaction throughput is not the only aspect of scalability; we also need to consider storage. As we explained in the cost discussion, both the transaction log of the blockchain and the world state are stored at every node. While the transaction log is stored as a single file, the world state is stored in a traditional NoSQL key-value database that natively scales horizontally. As the size of the transaction log grows, each node will either need to vertically scale its storage capacity as the hardware capacity is exceeded, or use a distributed file system. Additionally, private data is only stored at nodes of the organization to which the data belongs, which further partitions the world state. Thus organizations that record a lot of data in the system, will require greater storage capacity than those with less. Storage is very inexpensive, so this should not be a big problem, as a lot of organizations already store unstructured data on-premises.

5.2.5 Programming capabilities

Programming on blockchains has traditionally been quite cumbersome due to the inherent restrictions of the transaction architecture and consensus models. Programs deployed to traditional blockchains have to be implemented using restricted blockchain-specific programming languages in order to ensure the consistency of the blockchain state when the program is executed at every node within the network. Because every node has to execute the program, the programs are also limited in their size and number of instructions per transaction in order to evenly share resources and prevent denial of service attacks. Fabric is unique in its ability to deploy smart contracts written in traditional languages such as *JavaScript*, *Go*, and *Java*. The development process is similar to the development of regular applications and developers need not worry about restrictions on the size and number of instructions per transaction. This also means that our application can evolve as time goes on without us having to worry about reaching a limit in complexity. The only caveat is that the code cannot rely on in-deterministic functions (e.g., random number generators) when generating state, as this would lead to conflicts between the peers that are executing the transactions. This is not really a problem though, as such data can be generated client-side and supplied during the invocation of the smart contract function.

5.3 Chaincode

The chaincode refers to the application that is deployed to Fabric. Our chaincode is implemented in TypeScript [2]. TypeScript is an extension to JavaScript that has the same syntax but with added type support, which makes it both safer and easier to use. Since TypeScript compiles JavaScript, it runs in the same run time environments and has the same characteristics. The single-threaded asynchronous programming model means developers need not worry about synchronization and other semantics required by multi-threaded languages, which are often sources of bugs.

One of the most important aspects of software development is having a clearly defined architecture. The software architecture defines the structure of an application by setting the rules and conventions that the developers must follow. Rules can be which types of software entities are allowed to interact, and what modules and components are allowed to have a dependency relationship; i.e., clear system boundaries. The primary goal of our architecture is for the code to be modifiable and extendable without causing too many side effects, as defined in our quality attribute requirements (**E1**-**E3**, **M1**). This is achieved by having an architecture where we focus on maximizing cohesiveness and minimizing coupling.

[2] https://www.typescriptlang.org

In our solution, we have opted for a layered architecture. A layered architecture is an architecture where components are arranged into clearly defined layers. Each layer serves a specific role. Importantly, in our layered architecture, the level of abstraction increases further in the layer. The three layers of our architecture are, from the highest to the lowest level of abstraction: *domain layer*, *contract layer*, and *infrastructure layer*. Figure 5.1 is a visual representation of the described architecture.

In order to reduce coupling and not break abstraction, our rule for coupling is that a layer cannot depend on a layer that is further out. For example, the domain layer cannot depend on the contract layer, but both the contract layer and the infrastructure layer can depend on the domain layer. By following this simple rule, we can ensure some level of predictability for the consequences of modifying a particular layer. Intuitively, when we modify the highest-level code, i.e., the domain layer, that contains the central business logic, we expect the changes to affect the other lowe level layers. On the other hand, if we were to modify the infrastructure layer, which contains low-level components, our business logic will not be affected. This dependency rule and order of abstraction between the layers, align with this expectation.

5.3.1 Domain layer

The domain layer is the innermost layer, and is the layer where all business entities and rules live. It has the highest level of abstraction and the most pure code. Only concepts from the domain model should be reflected in the code; i.e., class names, methods, and variable names must belong to the ubiquitous language of the domain. When designing and implementing this layer, we are following the principles of domain-driven design. This means that the relationships and business logic realized by the code should be real concepts that exist in the domain. Methods should be behavior-oriented and reflect real actions. For example, an *Asset* entity with a method used to change its owner, should have the method named *transferTo* instead of *setOwnerId* to better capture the domain.

Our implementation of the domain layer is based on the domain model (figure 4.2) we constructed during the preliminary research into sustainability reporting. The resulting implementation is visualized using a class diagram in figure 5.2. It is not that much different from the conceptual model, and this view has a higher level of detail. Note that the arrows represent direct class dependencies and not general relationships. Most of the relationships of the domain model are implemented referentially by storing identifiers of the referenced entity instead of referencing the actual instance. This is the reason why there seem to be fewer relationships than in the original model.

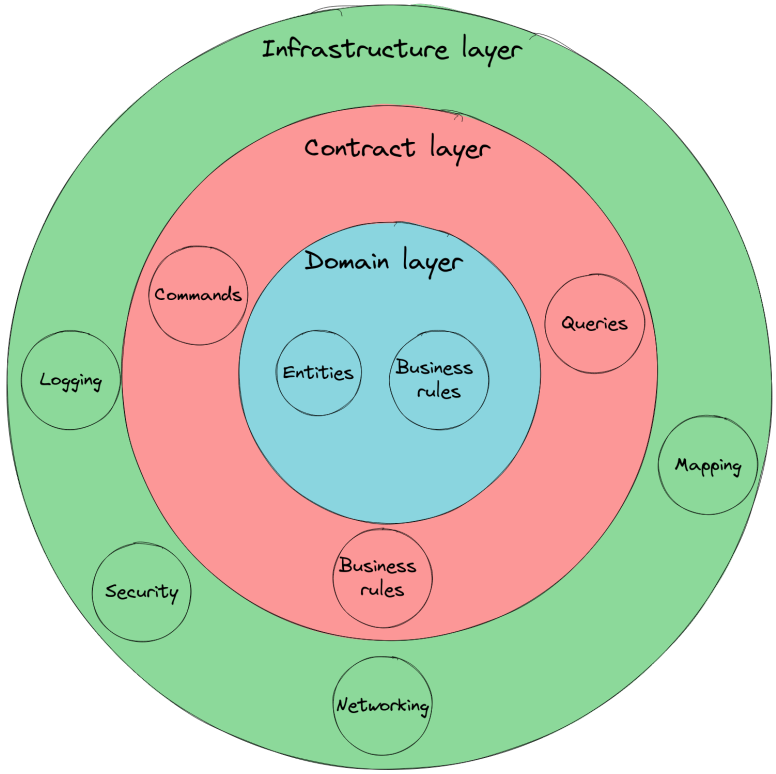

Figure 5.1: Chaincode software architecture

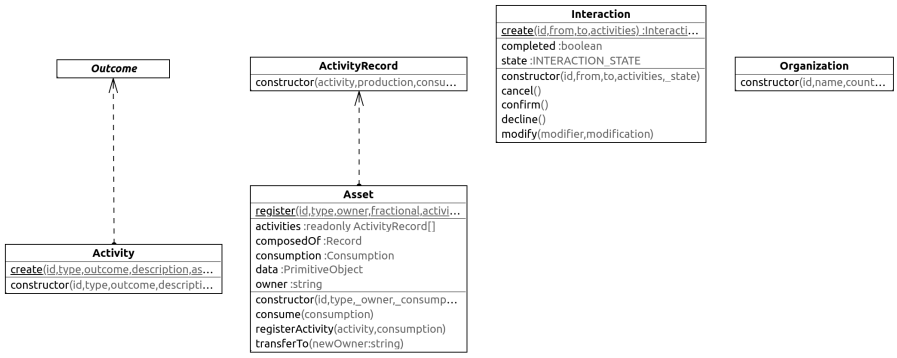

Figure 5.2: Class diagram of the domain layer.

5.3.2 Contract layer

For us to be able to interact and perform actions within our application, we need someplace where the use cases of our application are implemented. Since our solution is a blockchain application that is composed of multiple smart contracts that actually contain the use cases, we decided on naming it the Contract layer.

A use case can be any interaction the user of the application wants to perform, which is either a *command* or a *query*. While a command is an action that has side effects causing changes to the state of the system, a query is merely a request to read some data, causing no changes in the system. In order for a command or query to be carried out, the application layer will interact with the domain layer by retrieving the entities necessary to carry out the use case. Then the use case logic will perform the required operations on the entities that are involved in the operation through their defined methods and save the updated entities. In addition to having business rules within the domain layer that are bound to domain entities, additional use-case-specific business rules are present within the application layer. These rules are implemented and enforced within each specific use case.

5.3.3 Infrastructure layer

While both the domain and contract layers implement business logic and provide value to the end user, the infrastructure layer is where all the low-level details lie. The entire goal of the infrastructure layer is to implement functionality that supports the business logic. We can view the infrastructure layer as the glue of the application. An analogy would be the room of a house, which serves a specific purpose— business logic. For the room to function, we need to have the supporting infrastructure in place, such as the door, isolation, plumbing, etc.—the infrastructure.

The system boundary begins at this layer, i.e. at the door; all external communications arrive here first and can be further delegated to the contract layer to perform business logic. For example, a web application with a REST API would have an HTTP server component in this layer. Other examples of infrastructure components could be related to validation, monitoring, databases, security, low-level cryptographic libraries, and so on.

In our solution, we use the infrastructure layer primarily to perform the task of validation and mapping between data transfer objects (DTOs) and the corresponding software entities. To decouple the contracts from specific mappers, we have implemented a generic abstract service called *ISerializer* which delegates the mapping operation to the correct mapper based on the type that is being serialized or deserialized.

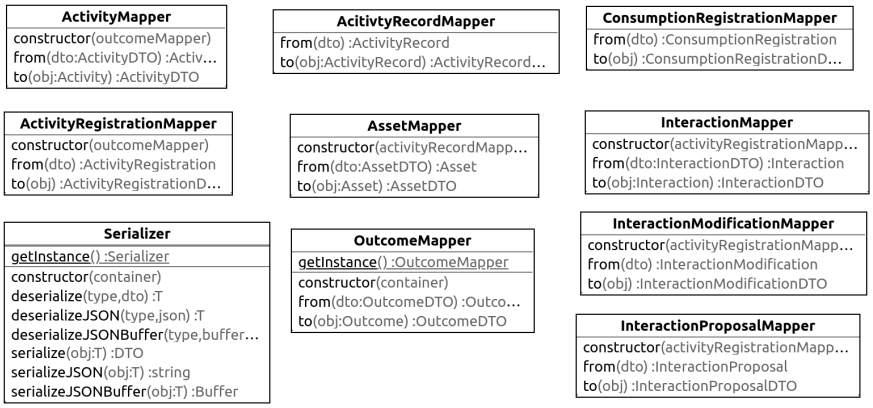

Figure 5.3: Class diagram of the infrastructure layer.

An overview of the infrastructure components is presented in Figure 5.3. We have omitted the data transfer object interfaces from this diagram to make the illustration more readable.

5.4 Architectural tactics

Architectural tactics are the architectural decisions made with the goal of satisfying one or more quality requirements. While the architectural drivers and the ASRs can affect the entire architecture by forcing major architectural decisions concerned with the architectural patterns early on in the project, the architectural tactics are actions taken for specific quality requirements that are less impactful to the overall architecture.

5.4.1 Security

S1: Storing reported data securely In order to satisfy security scenario 1, we use Hyperledger Fabric's privacy features. The implicit private data collection of the reporting business is used to store the private data off-chain, and we use the transient data feature to transmit the arguments of the invoked chaincode. The transient data will not be included in the ledger, and will only remain available to the endorsing peers of the organization registering the data.

S2: Securely aggregating private data Aggregating private data is required for multiple use cases in the system. FR8 and FR9 rely on aggregating private data from multiple organizations in order to obtain the total downstream and upstream sustainability impact.

The first and most straightforward way to obtain aggregated data from the different data sources is to have a central decryptor that is queried on-demand whenever a computation needs to be performed. While simple to implement, having a central decryptor requires all of the participants to place their trust in one party. This would be seen as too great of a risk for many firms. It also raises the question of who should be this central decryptor, which could be impossible to agree on in an environment where multiple system operators are involved. Trust is not the only issue; if we want to use a central decryptor, confidential data will need to be stored encrypted on-chain so that it is accessible to the decryptor. Having a central decryptor constitutes a single point of attack for an adversary, and if the decryption keys are compromised, the private data that is stored on-chain will be revealed to the attack.

Instead of having a central decryptor, we can employ a decentralized setup. The cryptographic scheme adhoc MIFE allows organizations to collaborate by sharing encrypted sustainability secured with encryption keys only they have access to. To perform a computation on data belonging to multiple organizations, the peers of each organization will need to be queried in order to retrieve the required data as it is stored in private data collections.

Collaboration is required in order to determine and query the relevant organizations. This is due to the lack of a holistic view of the supply chain; every organization only knows about its direct interactions with its partners. Thus in order to obtain data from organizations upstream and downstream, each organization in the supply chain network will need to query their partners. Figure 5.4 illustrates an example of the described process. A user belonging to Organization 1 wants to perform an aggregation query to obtain information across the supply chain. Actors denoted by `P`, correspond to partners between organizations across the supply chain. `P1` is a direct partner of `ORG1` and `P1.1` is a partner of `P1`, and so on. First, the user queries a peer belonging to its own organization. It then sets up a functional encryption scheme and the parameters are sent to the partners. The process is then repeated recursively for the partners and encrypted data from each individual actor is finally returned to organization 1. Finally, Organization 1 performs the computation on the encrypted data using functional encryption, and the obtained result is returned to the user.

Included in the query is information about the function which is being computed

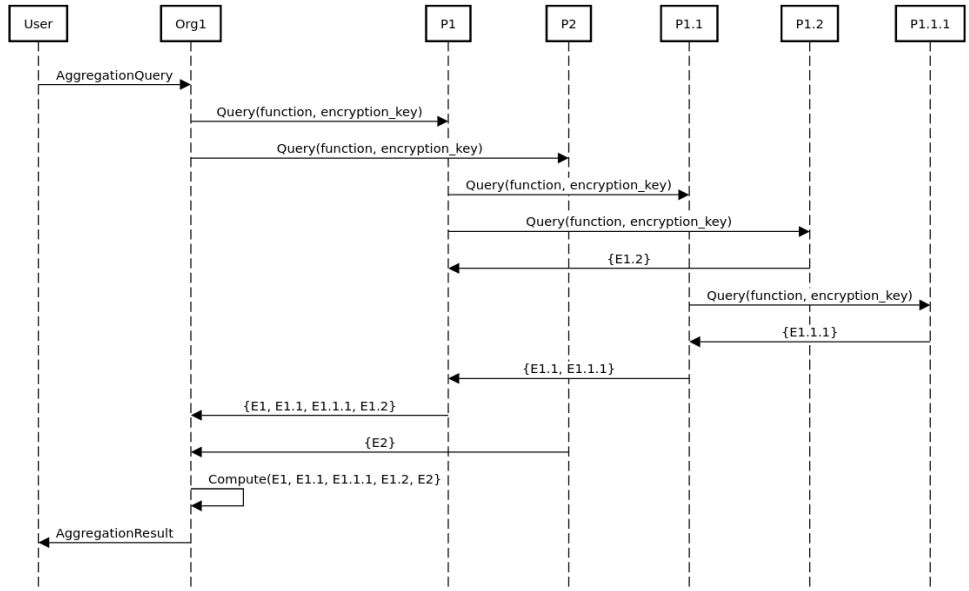

Figure 5.4: Secure decentralized data aggregation.

and the set of activities whose sustainability data is needed. The peer will then query its own world state for the relevant data, and encrypt it before returning the result. Along with the encrypted data, partial decryption keys are provided which are used to calculate the result without revealing the actual data inputs. As a result, there is no centralized security risk, and organizations can decide to whom and what to share when generating partial decryption keys.

S3: Preserving privacy when transferring asset Our solution provides the ability to transfer assets between organizations. Each asset registered in the system stores information about the asset such as its composition (if composed of other assets), what activities the asset has been a part of, as well as additional metadata. This information is stored in the private data collection of the owner of the asset. When transferring an asset to another organization, we cannot copy this information to the record added to the new owner's private data collection. Instead, we store the previous owner along with any other data the new owner wishes to provide. The record of the previous owner is updated to include information about the new owner. By storing owner information, we have a trail of owners associated with the given asset, but at the same time, we maintain privacy, since any given historic owner only knows the previous owner and the next owner.

5.4.2 Modifiability

Modifiability refers to the amount of work required to modify existing software components to introduce or remove functionality. We can measure modifiability in terms of the coupling and cohesion of the software components. Thus the tactics we have selected are intended to increase cohesion and reduce coupling.

Maintaining semantic coherence Semantic coherence means that modules that are coupled together, must be coherent in their responsibilities. This means that the modules should only be coupled if their responsibilities work together and are related. The aim is to have these responsibilities work together to achieve a broader goal. Maintaining semantic coherence thereby increases the cohesion of the software components. Our layered architecture defines clear responsibilities of their constituent components, which helps us maintain semantic coherence.

Abstract common service Instead of depending on a particular implementation of a service, we can define an abstraction and use polymorphism. A concrete example of this tactic in our solution is when to serializing and deserializing objects. Different objects will require different logic to perform serialization and deserialization. However, we do not want to couple our code directly to these components. Instead, we have defined an abstract service called *ISerializer*, whose contract provides methods to perform serialization and deserialization. The component which implements the *ISerializer* service, will internally use the concrete mappers used to convert between objects and their serialized format.

Encapsulation Encapsulation is the restricting of access to the state of a component for outside components. From a modifiability perspective, the purpose of encapsulation is to reduce coupling by preventing coupling to the internal details of a component; instead, we define public methods and properties that form a contract. Now external components are only coupled to the contract, hence it should remain as stable as possible to prevent cascading changes when modifying the component.

5.4.3 Extensibility

Extensibility refers to the ability to extend the existing functionality of the system without needing to modify existing code. Both modifiability and extensibility are aspects of the maintainability of the code.

Inheritance Inheritance is a technique in object-oriented programming that is used to define an abstraction for a common set of classes using a base class. This class defines a common interface for each of the inheriting subclasses. Subclasses can in turn be inherited, forming an object inheritance hierarchy. Using inheritance, we

get polymorphism which lets us write code that relies on the abstract superclass. This lets us add new subclasses that inherit from the abstraction without needing to modify code that only relies on the abstraction. The drawback of inheritance is that it couples the sub-classes to the abstraction. Therefore it is important that the inheritance relationship is semantically correct, to make sure our inheritance relationships are cohesive.

Composition Composition is orthogonal to inheritance and is a different method to obtain extensible code. Instead of creating abstractions using base classes with common functionalities, for then to extend the functionality using specialized sub-classes, we can use composition. With composition, we create constituent classes with clearly defined responsibilities and relationships. Then, in order to achieve the functionality of the whole, we compose the functionality by letting classes use other classes within the composition to obtain the required behavior. Classes that use other classes in the composition will be coupled, and it is important they have high cohesion.

Note that inheritance and composition are not mutually exclusive. They can be used together if it is suitable for the given problem.

5.5 Hyperledger Fabric

The backbone of our solution is the blockchain operating system, Hyperledger Fabric. Fabric is modular which allows it to be customized to each use case. In this section, we will discuss the decisions we have made for the configuration of Fabric.

5.5.1 Network architecture

The configured network architecture is displayed in figure 5.5. We use one channel where the blockchain application is deployed, which is shared by all of the participants in the network. Each member organization provides its own peers, CA, and ordering service. The world state is the same for all peers in an organization but differs between organizations due to private data.

5.5.2 Endorsement policy

The endorsement policy defines the set of required nodes that have to simulate the execution of a transaction in order for the transaction to be considered valid and committed to the blockchain. In applications where there are different parties involved in a shared process where the state of the process should be accessible to every party, it could be useful to define a custom endorsement policy, for example

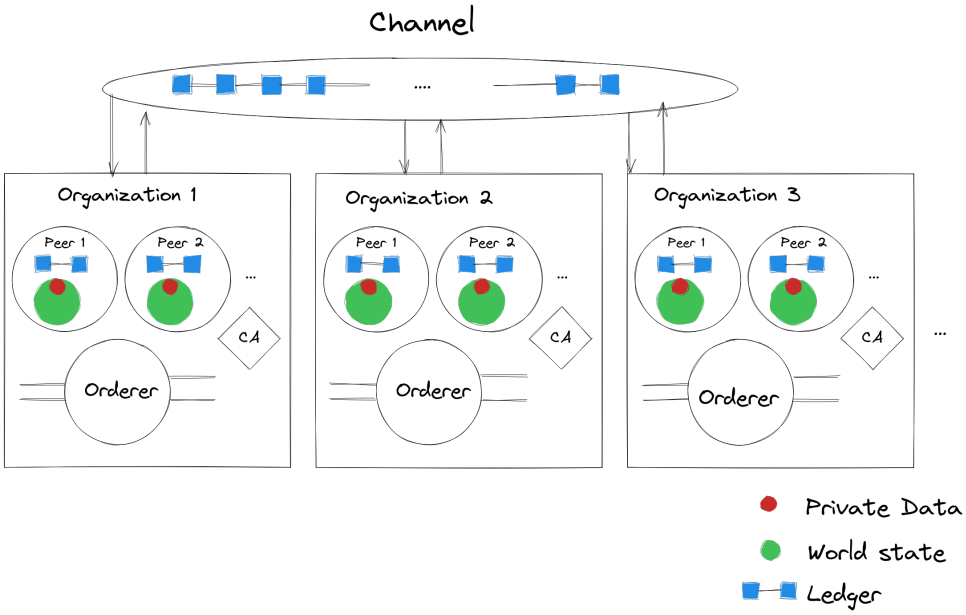

Figure 5.5: Configured network architecture for Hyperledger Fabric.

requiring two out of three parties to endorse the transaction. This ensures that even if one of the parties is dishonest, it can be detected by the other endorsement party.

In our case, most of the data to be inputted into the system is private to the provider of the data. Hence, our endorsement policy needs only be confined to one endorser—the peer for whom the chaincode was first invoked. The only exception is the use case where two parties register a joint interaction. Luckily, Fabric handles this for us; by default, if a transaction is written to a private data collection, it must be endorsed by the organizations that have access to the private data collection. Thus in the interaction use cases, where we write to the private data collection of both participants, Fabric requires both to endorse the transaction, which is exactly what we want in a scenario where we cannot assume both parties trust each other.

5.5.3 World-state database

Fabric comes with support for using either CouchDB or LevelDB as the world-state database out of the box. Both are key-value stores, that are optimized for key-based lookups. While LevelDB is a barebone simple key-value store that only supports key-based queries, CouchDB provides additional features such as indexing and parameterized queries. As a proof-of-concept, with a small world state, we

do not need this advanced querying, and performing key-based lookups is enough. Considering that LevelDB requires no additional setup with Fabric, we have decided on keeping things simple and opted for LevelDB as our world-state database.

5.6 Scenarios

We will now present a selection of use case scenarios to demonstrate the flow of action in the implementation. For each scenario, we will provide a sequence diagram that demonstrates the interaction between the different components of the system. Relevant code will be presented to aid in the explanation.

5.6.1 Scenario 1: Registering a business activity

The main use case that is used to report sustainability information is the registration of business activities. This is implemented by the `ActivityContract`. An organization registers a business activity. The registered activity is a business activity that produces phones. In this instance, three phones are produced, and as an example, the environmental impact of the activity is the emission of $40kg$ of CO_2.

Organization 1 issues the `registerActivity` with the arguments in figure 5.6. The scenario is displayed in figure 5.6. Org1 invokes the `registerActivity` command of the `ActivityContract`. Then the data is deserialized into an `ActivityRegistration` object. It then checks to make sure that there is no existing activity with the same ID. Because this activity produces three assets, the `ActivityContract` invokes the `registerAssets` command of the `AssetContract`. The `AssetContract` performs similar validation and upon the successful registration of the produced assets, the new activity is saved in the system.

5.6.2 Scenario 2: Two parties performing an interaction with the transfer of assets

An important use case scenario that is needed to establish links between the organizations and assets throughout the supply chain is the recording of interactions. `InteractionContract` implements the business logic for performing interactions. It provides methods to propose, query and confirm pending interactions. As part of an interaction, we can also include transfers of assets. The asset transfer functionality is implemented by the `AssetContract`, which is invoked by the `InteractionContract`.

In our scenario, organization 1 proposes an interaction with Organization 2, which includes the transfer of some assets from Organization 1. This interaction represents a transaction between the two parties, where Organization 2 becomes the new owner of the provided assets.

```
{
    "id": "123",
    "type": "PhoneProduction",
    "outcome":
    [
        {
            "type": "GreenhouseGasEmitted",
            "gas_type": 0,
            "unit": 1,
            "magnitude": 40
        }
    ],
  "assets":
  {
    "produced":
    [
        {
            "id": "1",
            "type": "Phone",
            "fractional": false,
        },
        {
            "id": "2",
            "type": "Phone",
            "fractional": false,
        },
        {
            "id": "3",
            "type": "Phone",
            "fractional": false,
        }
    ]
  }
}
```


Figure 5.6: Scenario 1: Business activity details submitted in the registration.

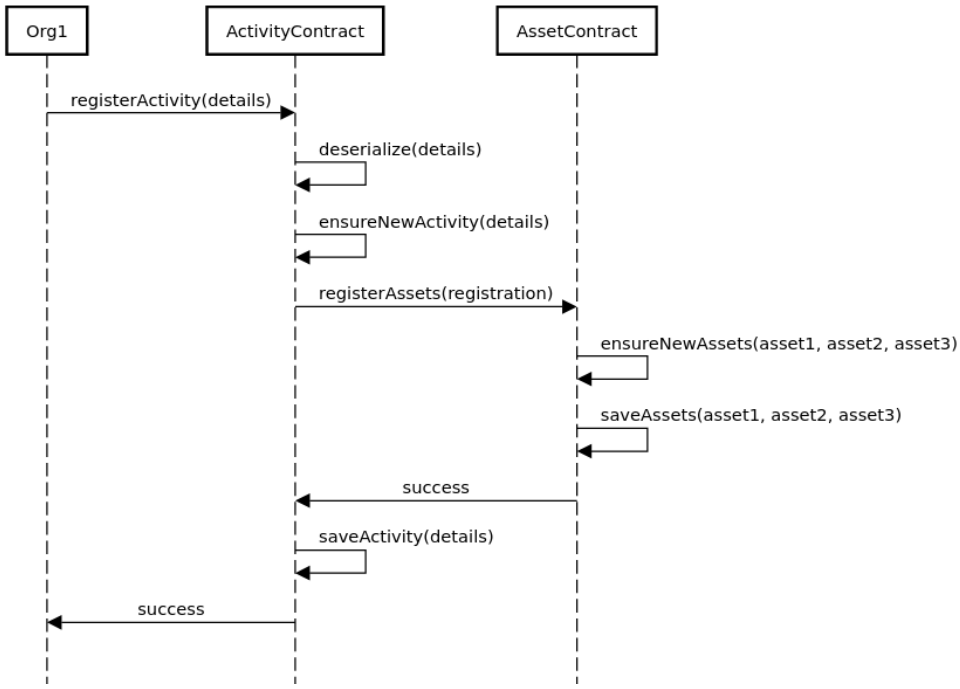

Figure 5.7: Scenario 1: Registering business activity that produces three assets.

Figure 5.9 illustrates the flow of interactions between the involved components. Note that this scenario is composed of two separate transactions; one in which Organization 1 proposes the interaction, and the other where Organization 2 confirms the interaction which subsequently transfers the assets.

The code implementations of `proposeInteraction` and `confirmInteraction` can be seen in figures 5.10 and 5.11, respectively. Since this use case involves more than one private data collection, we are only able to read the on-chain hashes of the private data and not the data itself. To ensure that both parties agree on the final interaction when it is being confirmed, we compare the hash of the latest interaction state which is provided by the confirmer. This is implemented by the helper method `verifyProvidedInteraction` which can be seen in figure 5.12. In this method, we compute the *SHA256* hash of the state of the provided object and compare it to the private data hash.

```
if(await this.exists(ctx, registration.id))
    throw new Error("An activity with the provided id already exists.")

if(registration.activities && !((await
↪  Promise.all(registration.activities.map(x => this.exists(ctx,
↪  x)))).every(x => x)))
    throw new Error("Unknown activity specified");


let assetCount = 0;

if(registration.assets)
{
    if(registration.assets.produced)
    {
        await AssetContact.getInstance().registerAsset(ctx, new
        ↪  AssetRegistration(registration.assets.produced.map(x => ({id:
        ↪  x.id, type: x.type, fractional: x.fractional, composedOf:
        ↪  x.composedOf, activities: [new
        ↪  ActivityRecord((<ActivityRegistration>registration).id,
        ↪  true)]})));
        assetCount += registration.assets.produced.length;
    }

    if(registration.assets.consumed)
    {
        await AssetContact.getInstance().consumeAssets(ctx, new
        ↪  ConsumptionRegistration(registration.assets.consumed,
        ↪  registration.id));
        assetCount += Object.keys(registration.assets.consumed).length;
    }

    if(registration.assets.involved)
    {
        await AssetContact.getInstance().registerInvolvement(ctx, new
        ↪  InvolvementRegistration(registration.assets.involved,
        ↪  registration.id));
        assetCount += registration.assets.involved.length;
    }

}
const newActivity = Activity.registerNew(registration.id,
↪  registration.type, registration.outcome, registration.description,
↪  assetCount,registration.data, registration.activities);
await this.saveActivity(ctx, newActivity);
return {activity: Serializer.getInstance().serializeJSON(newActivity)};
```

Figure 5.8: Code snippet: `registerActivity` implementation.

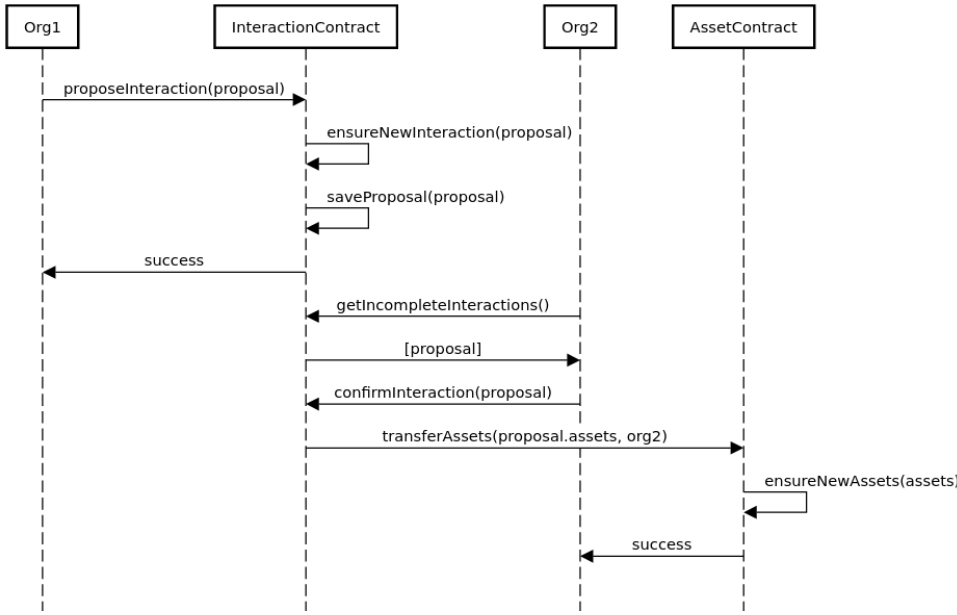

Figure 5.9: Scenario 2: Two parties performing an interaction with the transfer of assets.

```
public async proposeInteraction(ctx: Context, interactionProposal:
↪  InteractionProposal)
{

    const interaction = Interaction.create(interactionProposal.id,
    ↪  ctx.clientIdentity.getMSPID(), interactionProposal.to,
    ↪  interactionProposal.activities, interactionProposal.transfers);

    const [hash1, hash2] = await
    ↪  Promise.all([ctx.stub.getPrivateDataHash(`_implicit_org_${interaction.from}`,
    ↪  `interaction-${interaction.id}`),
    ↪  ctx.stub.getPrivateDataHash(`_implicit_org_${interaction.to}`,
    ↪  `interaction-${interaction.id}`)])

    if(hash1?.length !== 0 || hash2?.length !== 0)
        throw new Error("An interaction with the given ID already
        ↪  exists.");

    await this.saveInteraction(ctx, interaction);
}
```

Figure 5.10: Code snippet: Scenario 2 – `proposeInteraction` implementation.

```
public async confirmInteraction(ctx: Context, interaction: Interaction)
{
    await this.verifyProvidedInteraction(ctx, interaction);
    if(interaction.completed)
        throw new Error("Interaction has already been completed and
        ↪  cannot be modified.");

    const confirmer = ctx.clientIdentity.getMSPID();

    if((confirmer === interaction.to && interaction.state ===
    ↪  INTERACTION_STATE.RECEIVER_REPROPOSAL) || (confirmer ===
    ↪  interaction.from && interaction.state ===
    ↪  INTERACTION_STATE.SENDER_REPROPOSAL))
        throw new Error("The interaction cannot be confirmed by the last
        ↪  party that proposed it.")
    interaction.confirm();
    await this.saveInteraction(ctx, interaction);
    if(Object.keys(interaction.transfers).length > 0)
    {
        await Promise.all(Object.keys(interaction.transfers).map(x =>
        ↪  (async () =>
        {
            const to = x === interaction.from ? interaction.to :
            ↪  interaction.from;
            await AssetContact.getInstance().transferAsset(ctx, new
            ↪  TransferRequest(to, interaction.transfers[x], x));
        })()));
    }
}
```

Figure 5.11: Code snippet: Scenario 2 – `confirmInteraction` implementation.

```
private async verifyProvidedInteraction(ctx: Context, interaction:
↪  Interaction)
{
    const clientMsp = ctx.clientIdentity.getMSPID();
     if(![interaction.to,
     ↪  interaction.from].includes(ctx.stub.getMspID()))
         throw new Error("The executing peer is not allowed to endorse
         ↪  this.");
    if(![interaction.to, interaction.from].includes(clientMsp))
        throw new Error("The client with msp " + clientMsp + " does not
        ↪  belong to an MSP involved in the provided interaction");

    const sha256 = createHash("sha256");
    sha256.update(this.serializer.serializeJSON(interaction));
    const providedInteractionHash = sha256.digest();

    const [hash1, hash2] = await
    ↪  Promise.all([ctx.stub.getPrivateDataHash(`_implicit_org_${interaction.from}`,
    ↪  `interaction-${interaction.id}`),
    ↪  ctx.stub.getPrivateDataHash(`_implicit_org_${interaction.to}`,
    `interaction-${interaction.id}`)]);

    if(!providedInteractionHash.equals(hash1) ||
    ↪  !providedInteractionHash.equals(hash2))
        throw new Error("The provided hash does not match the hash of the
        ↪  current interaction state.");

}
}
```

Figure 5.12: Code snippet: Scenario 2 – `verifyProvidedInteraction` implementation.

Chapter 6
Evaluation and discussion

6.1 Functional requirements

FR1: Register business activities This core use case has been implemented according to the requirements. When registering an activity, one or more sustainability outcomes can be included. Currently, the solution only supports a selection of a few kinds of outcomes. However, it is very easy to extend by implementing additional outcome value objects. Aside from this, every other requirement and sub-requirement has been fulfilled. However, further investigation is needed to determine whether there is a need for the ability to register activities that are composed of other activities. It was hypothesized that this would be a useful feature, but it remains to be confirmed.

FR2: Register business assets The implemented functionality to fulfill FR2 seems to be a good approach to modeling how assets can be described in the real world. With the ability to specify whether an asset is fractional or not, we can support the registering and tracking of assets whose consumption pattern can be continuous, which can be the case for fungible assets such as raw materials. On the other hand, assets that are not fractional, are either not consumed or consumed in their entirety. This can be used to model the end-of-life of manufactured products, or the embedding of processed goods within a composite end-product, such as sensors in cars.

FR3: Register interaction between two organizations FR3 is realized by the **InteractionContract** The intention of the interaction registration is to provide the ability to track the exchange of goods and services between businesses. This is needed if we want to track the indirect sustainability impact of such transactions. The resulting implementation lets the users associate an interaction with one or more activities to indicate participation in the activity. If the interaction is an exchange of goods, the transfer of assets can be included in the interaction. Currently, interactions are limited to two parties. This means that a real-world interaction between more

than two parties must be registered as multiple two-way interactions between the participants. Should the need arise, it could be extended to an arbitrary number of parties.

FR4: Transfer asset ownership The implementation of asset ownership transfer is implemented as a standalone feature in the **AssetContract**. Initially, it seemed practical to provide the user the ability to transfer asset ownership directly. Later we realized this can be problematic for the tracking of the assets. Instead, it seems more sound to require the transfer of assets to be done as part of an interaction (FR3).

FR5: View all interactions A simple method to retrieve all interactions and proposals from the system is provided to the end user. In the future, additional querying capabilities should be implemented to give users the ability to filter out, sort, and retrieve a subset of the interactions that are of relevance. Pagination must also be implemented in order to support a large number of interactions.

FR6: View business activities Similarly to FR5, the current feature for retrieving business activities is simple and only retrieves the entire collection of activities. We propose the same measures; querying capabilities with filtering and sorting, and pagination.

FR7: Consume assets The purpose of supporting the registration of asset consumption is to provide a way to represent the usage of assets such as activities where a business uses assets to produce other assets as well as the combustion of materials. The implementation allows assets that are registered as fractional to be consumed in fractional amounts, whereas non-fractional assets must be consumed in their entirety. Modeling consumption this way seems to be a good fit for the real-life scenarios we have foreseen.

FR8: View total sustainability impact Calculating the total sustainability impact of an organization requires traversing the supply chain for each asset they own and have owned. Additionally, it also requires traversing the activities they have registered and their interconnections in the supply chain. In other words, this is a complex and resourceful computation to carry out. Unfortunately, due to time constraints, we have not been able to implement this feature.

FR9: View sustainability impact of asset Similarly to FR8, we have not been able to implement this feature as the same issues with FR8 will need to be addressed.

6.2 Quality requirements

6.2.1 Extensibility

For our solution to be able to handle future requirements, and for the complexity of the application to scale well, it must be extensible. We have employed a combination of composition and inheritance to facilitate this.

Composition is used extensively in the contract layer where the use cases for the system are implemented. Some of the use cases rely on invoking existing use cases. This is where composition comes to the rescue; instead of duplicating the logic across the use cases, we compose the functionality by letting the contracts re-use logic from one another where needed. An example of the use of composition is in the `registerActivity` use case implemented by the `ActivityContract`. During activity registration, the user can also provide information about assets that were involved in the activity. However, this can also be done separately after the activity has been registered and has been implemented as a use case by the `AssetContract`. Thus we reuse this functionality, by having the `ActivityContract` invoke relevant functions implemented by the `AssetContract`.

Inheritance, on the other hand, is only used a few times. To avoid the need to reimplement cross-cutting concerns in each contract, we have defined a base class `AbstractContract` which implements deserialization of the arguments that are provided to the invoked chaincode. Additional functionality that needs to be added to all contracts can be implemented using the base class.

Our use of composition and inheritance has proved to facilitate a code base that is easily extensible. We employed these modifiability tactics from the beginning of the implementation, and building on the solution has been an unencumbered process.

6.2.2 Modifiability

Ensuring high modifiability is important to minimize friction that could arise in the event that changing existing functionality is necessary. The tactics we have applied have proven successful in this task.

Maintaining semantic coherence has been a primary concern from the beginning of implementation, and only software components that have related responsibilities are allowed to be coupled. For the layered architecture, we decided on implementing further semantic cohesion by defining clear boundaries between the different classes of software components. Additionally, the dependency rule which states that layers can only depend on layers further in, reduces the level of coupling. As a result, we have a code base that has a high level of cohesion and low coupling.

Encapsulation further reduces the possibility of tightly coupling modules together by hiding internal implementation details. As the implementation is developed in an object-oriented programming language, we have built-in support for encapsulation through access modifiers and classes. The contracts in our contract layer, for example, have private helper methods that are used internally, and public methods that let other modules call them. These methods have clearly defined names that state their intent; their internal implementation is not important to the user of the method as long as its functionality is correct. We have used this extensively to compose and re-use business logic, by letting the different contracts call each other.

6.2.3 Deployability

Deploying a new version of a decentralized system is quite similar to traditional applications, with some additional administrative overhead. First, the new chaincode is packaged and prepared for deployment by an organization. Then the chaincode will have to be manually installed by each organization.

Because the installation of the new chaincode is done independently by each organization, it requires a way by achieving consensus upon the proposal of a new deployment to ensure that all nodes commit to using the new deployment. In the channel definition, we can define the application life cycle endorsement policy, which governs the consensus of the chaincode life cycle. The endorsement policy defines the number of required organizations that need to approve the new chaincode deployment. In the event that an organization is unwilling to accept and install the new chaincode definition on their peers, even though the endorsement policy is satisfied, they will be unable to use the chaincode of the channel.

Additionally, we can require that a set of predetermined organizations provide a signature for the new chaincode definition that is going to be deployed. These organizations can act as administrators of the chaincode and govern its business logic. This configuration aligns well with our proposed organizational structure for the operation of the system, which involves having governing agencies act as system operators that manage the network.

Other than the need to coordinate the member organizations during the deployment process, Fabric is built to fully handle the life cycle of the chaincode which makes deploying the application to a Fabric Network seamless. The manual installation of new chaincode in each member organization can further be automated by the use of automation tools such as scripts. Thus the administrative overhead can be reduced to a minimum.

6.2.4 Scalability

The scalability of the solution is fully determined by the Fabric network, which in turn depends on how it is configured and the usage patterns of the deployed application. The configured endorsement policy along with the transaction logic, will determine which peers are required to endorse (i.e. execute) the transaction.

Since our solution mostly involves one organization in each transaction, except for the use cases where two organizations interact, the transaction throughput is bound locally to each organization. These scalability properties are favorable because organizations contribute to the scaling of the network according to their own needs; organizations that require a large throughput are most likely larger enterprises that have the resources to provision the required infrastructure. By contrast, smaller organizations will not be performing that many transactions, and hence will not be required to provision many resources to participate in the network. Thus the costs of scaling are distributed fairly among the participants.

6.2.5 Security

Ensuring that the implementation satisfies the security requirements is paramount. Thanks to the privacy features of Hyperledger Fabric, we have been able to successfully implement secure storage of reported data; the private data is siloed at the organizational level since the private data collections are bound to each organization. Of course, if the peers themselves are compromised, the data can be obtained. Hence the security of the private data is the responsibility of the organization to which it belongs.

The most challenging security requirement is the secure aggregation of data from multiple organizations. While we have not implemented a solution to the problem, we explored two potential ways of achieving this; either using a central trusted aggregator or by means of decentralized collaboration using functional encryption. While using a central aggregator is simple to implement, it does not provide strong enough security guarantees due to the centralized risk and trust requirements.

Decentralized collaboration using functional encryption, on the other hand, seems like a promising solution to the problem that fits well within a decentralized environment. It is, however, more difficult to implement, and requires further research in order to set up a secure protocol. Additionally, it requires different organizations to talk to each other's peers, which could introduce additional networking overhead. Despite this, it seems to be the most viable solution from a security perspective.

Preserving privacy when moving data between organizations is a necessity. The only current use case where this applies is the transfer of asset ownership. Our

solution is to not copy all of the information about an asset when transferring it to another organization. Instead, we let the historic owners have their own view of the asset. Each owner can have their own private data in their asset record. Upon the transfer of an asset, the previous owner records the new owner in their private data, and conversely, the new owner records the previous owner. This way, we can preserve the ownership trail while preserving privacy because the record of ownership is only known between parties that have exchanged the asset.

6.3 Application for greenwashing

We will now explore the application of the solution to scenarios of greenwashing. Although the current implementation does not provide the means to share data between organizations, we will assume access to all data, to demonstrate the potential uses.

6.3.1 Product-level greenewashing

As an example, we are going to simulate a scenario of product-level greenwashing. The scenario is as follows. Yogoa is a producer of yogurt that sells its products to grocery stores. Yogoa and all of the other suppliers along their supply chain are using Gaia to track and record their assets and business activities. Yogoa's supplier just started selling recycled plastic, at a higher price compared to non-recycled plastic. Even though the price for recycled plastic was higher, Yogoa decided to make a switch and use it in their carton. To promote this change, they changed the packaging and added a green label stating "*Now made with recycled plastic.*", as it would be good marketing. The question is—has the environmental impact of the product been reduced?

We will assume that Yogoa processes its own carton packaging at its facilities during the manufacturing of its yogurt products. We will assume the constituents of the carton to be paper and plastic. For this example, we will only consider the packaging of the finished product and not the yogurt itself. The supply chain is shown in Figure 6.1. Each batch of yogurt produces 50000 units and requires 1 metric tonne of paper and 10kg of plastic.

Table 6.1 shows the business activities that each actor has recorded from January to March. The transactions between Yogoa and its suppliers have been recorded in Gaia as interactions, which are shown in Table 6.2.

Calcuating the impact Let us look at the environmental impact of the packaging before and after making the switch to recycled plastic. The total impact of the first batch produced with non-recycled plastic, which includes the upstream emissions, is

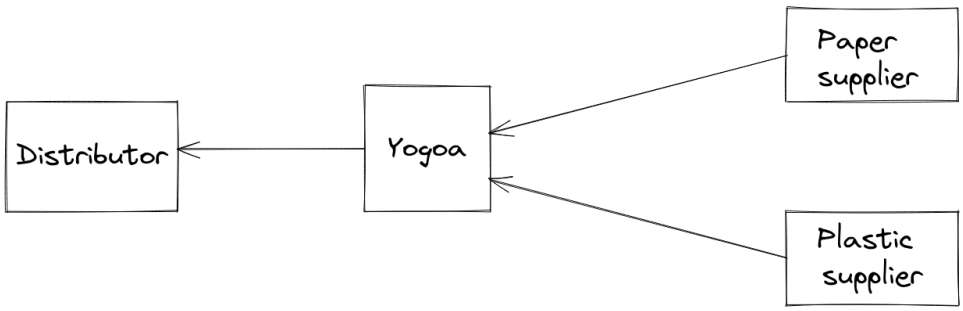

Figure 6.1: The supply chain of Yogoa.

Table 6.1: Product-level greenwashing scenario: Registered business activities.

Date	**Organization**	**Activity**	**Impact**
January 2023	Paper Supplier	Production of 1 tonne of paper - batch #45602.	1500kg CO2.
January 2023	Plastic Supplier	Production of 1 tonne of plastic - batch #WX5831.	1600kg CO2
February 2023	Yogoa	Production of 50000 packages.	100kWh electricity
February 2023	Paper Supplier	Production of 1 tonne of paper - batch #46010.	1500kg CO2.
February 2023	Plastic Supplier	Production of 1 tonne of plastic - batch #WX5832.	1600kg CO2
February 2023	Plastic Supplier	Production of 1 tonne of recycled plastic - batch #ZX1003.	2100kg CO2
March 2023	Yogoa	Production of 50000 packages using recycled plastic.	100kWh electricity

Table 6.2: Product-level greenwashing scenario: Registered interactions.

Date	Organization 1	Organization 2	Description
January 2023	Yogoa	Paper supplier	Purchase of 1 tonne of paper.
January 2023	Yogoa	Plastic supplier	Purchase of 100kg of non-recycled plastic.
February 2023	Yogoa	Paper supplier	Purchase of 1 tonne of paper.
February 2023	Yogoa	Plastic supplier	Purchase of 100kg of recycled plastic.

given by

$$I_1 = I_{\textbf{paper}} + I_{\textbf{plastic}} + I_{\textbf{production}}$$

Each impact component is obtained from the registered business activities in table 6.1. Since Yogoa has purchased an entire batch of paper, we have

$$I_{\textbf{paper}} = 1500\text{CO}_2\text{kg}$$

, which is the full impact of the paper production activity.

On the other hand, $I_{\textbf{plastic}}$ is the fraction of the batch that is processed, which is 10kg out of 1000kg. Thus

$$I_{\textbf{plastic}} = \frac{1600\text{kgCO}_2}{1000kg} \cdot 10\text{kg} = 16\text{kgCO}_2$$

The final component, which is the impact of the processing of the paper and plastic, turning it into the final packaging, is measured in terms of consumed electricity. In order to obtain the emissions in $CO2$, we will convert the consumed electricity using the average emissions per kWh at the location of the processing facility, which we are going to assume to be $35\text{gCO}_2/\text{kWh}$.

$$I_{\textbf{production}} = 10\text{kWh} \cdot \frac{35\text{gCO}_2}{\text{kWh}} \cdot 50\text{kWh} = 1750\text{g} = 1.75\text{kg}$$

In total, the impact of the packaging is

$$\begin{aligned} I_1 &= I_{\textbf{paper}} + I_{\textbf{plastic}} + I_{\textbf{production}} \\ &= 1500\text{kgCO}_2 + 16\text{kg} + 1.75\text{kg} \\ &= 1517.75\text{CO}_2\text{kg} \end{aligned}$$

Thus the average environmental impact for each unit of packaging, using *non-recycled* plastic, is $1517.75/50000 = 30.3\text{gCO}_2$.

Let us perform the same calculations for the second batch, which uses recycled plastic instead.

$$I_2 = I_{\textbf{paper}} + I_{\textbf{recycled plastic}} + I_{\textbf{production}}$$

$$I_{\textbf{recycled plastic}} = \frac{1600\text{kgCO}_2}{2100kg} \cdot 10\text{kg} = 21\text{kgCO}_2$$

The impacts of the paper and processing were the same compared to the previous batch, so the calculated impact is

$$\begin{aligned} I &= I_{\textbf{paper}} + I_{\textbf{recycled plastic}} + I_{\textbf{production}} \\ &= 1500\text{kgCO}_2 + 21\text{kgCO}_2 + 1.75\text{kgCO}_2 \\ &= 1522.75\text{kgCO}_2 \end{aligned}$$

Which per unit corresponds to $1522.75/50000 = 30.5\text{gCO}_2$, which is more than when using non-recycled plastic! What we have just demonstrated is the ability to reveal greenwashing using Gaia.

6.3.2 Organization-level greenwashing

We will now look at an example of using Gaia for organization-level greenwashing. In this example, we will assume the following scenario. Saastastic is a software-as-a-service (SaaS) business that provides software to its customers. Their profile is green and their core values are sustainability-oriented, and they claim to be committed to finding the most sustainable partners.

Saastastic has been using the cloud provider called JediCloud to host their SaaS application. JediCloud is a cloud provider that focuses on low carbon emissions and sustainable infrastructure. For this reason, their infrastructure is located in countries whose energy primarily comes from renewable sources. JediCloud is quite a bit more expensive than its competitors due to the added costs of the location of its infrastructure.

Unfortunately, Saastastic has been struggling with its operational costs and is hardly turning a profit. In a recent meeting, the administration determined that they need to use a cheaper hosting provider. They decided on using another cloud

provider called SithCloud, which provides the same services as JediCloud, but at less than half the price.

Let us examine the impact this has had on the environmental footprint of Saastastic. We are going to assume that Saastatic and its supply chain have recorded their business activities in Gaia for the past six months, with the change of cloud provider occurring at the end of March 2023. Tables 6.3 and 6.4 list all the business activities that have been recorded in the past six months. Currently, there is no support for the ability to specify the amount an organization participates in another organization's activity, which could be used to model the individual organizations' utilization of a shared service. Nevertheless, we will assume this feature has been implemented. The interactions include the percentage of utilization of the purchased cloud service relative to the total utilization of all the customers. To obtain the total environmental footprint of Saastastic, we will consider the footprint of the cloud services they have purchased and footprint of the their offices. The impact for a given month k is given by

$$I_k = I_{k_{\mathbf{office}}} + I_{k_{\mathbf{cloud}}}$$

Where $I_{k_{\mathbf{office}}}$ is obtained directly from the registered office operation activities, and converted to CO2 equivalent.

$I_{k_{\mathbf{cloud}}}$ is a bit more nuanced to calculate since it is a shared service provided to several users. However, the share of the service utilization is specified in the list of interactions. Hence, the impact associated with Saastastic for the cloud service is given by

$$I_{k_{\mathbf{cloud}}} = \text{Utilization} \cdot \text{Total impact of cloud service}$$

The total CO2 emissions of Saastastic in the past six months are plotted in Figure 6.2. As we can see, their emissions increased after March, which is due to the switch to SithCloud. The emissions include the office operation, which actually decreased in the first half of the six-month period, due to warmer weather. If we only consider the emissions from the cloud provider, the increase becomes even more apparent, as can be seen in Figure 6.3

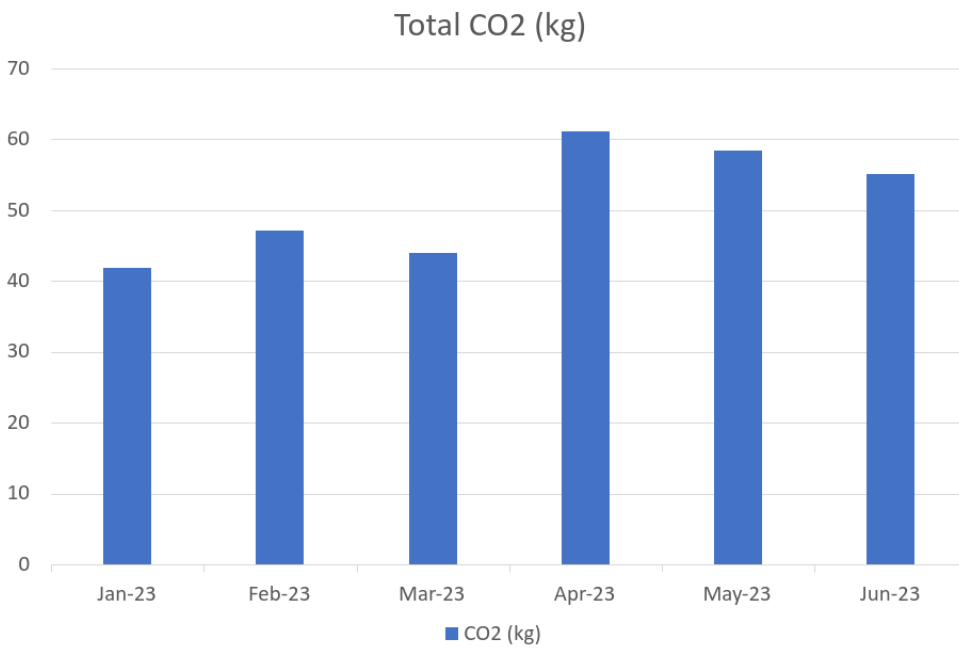

Figure 6.2: Organization-level greenwashing: Total CO2 emissions of Saastastic.

Table 6.3: Organization-level greenwashing: Registered activities.

Date	Organization	Activity	Impact
January 2023	JediCloud	Cloud hosting, region 1.	1000kg CO2
January 2023	JediCloud	Cloud hosting, region 2.	1100kg CO2
January 2023	SithCloud	Cloud hosting, region 1.	1810kg CO2
January 2023	SithCloud	Cloud hosting, region 2.	2170 CO2
January 2023	Saastastic	Office operation	908 kWH
February 2023	JediCloud	Cloud hosting, region 1.	950kg CO2
February 2023	JediCloud	Cloud hosting, region 2.	1220kg CO2
February 2023	SithCloud	Cloud hosting, region 1.	1920kg CO2
February 2023	SithCloud	Cloud hosting, region 2.	2120kg CO2
February 2023	Saastastic	Office operation	1020 kWH
March 2023	JediCloud	Cloud hosting, region 1.	1150kg CO2
March 2023	JediCloud	Cloud hosting, region 2.	1250kg CO2

Date	Organization	Activity	Impact
March 2023	SithCloud	Cloud hosting, region 1.	1973kg CO2
March 2023	SithCloud	Cloud hosting, region 2.	2080kg CO2
March 2023	Saastastic	Office operation	860 kWH
April 2023	JediCloud	Cloud hosting, region 1.	1030kg CO2
April 2023	JediCloud	Cloud hosting, region 2.	1170kg CO2
April 2023	SithCloud	Cloud hosting, region 1.	2021kg CO2
April 2023	SithCloud	Cloud hosting, region 2.	2200kg CO2
April 2023	Saastastic	Office operation	740 kWH
May 2023	JediCloud	Cloud hosting, region 1.	1050kg CO2
May 2023	JediCloud	Cloud hosting, region 2.	1200kg CO2
May 2023	SithCloud	Cloud hosting, region 1.	2010kg CO2
May 2023	SithCloud	Cloud hosting, region 2.	1980kg CO2
May 2023	Saastastic	Office operation	650 kWH
June 2023	JediCloud	Cloud hosting, region 1.	950kg CO2
June 2023	JediCloud	Cloud hosting, region 2.	1200kg CO2
June 2023	SithCloud	Cloud hosting, region 1.	1900kg CO2
June 2023	SithCloud	Cloud hosting, region 2.	2000kg CO2
June 2023	Saastastic	Office operation	600 kWH

6.4 Barriers

Self-hosting Perhaps the greatest barrier against adoption is the need to self-host and set up the infrastructure required to join the blockchain network. Consequently, the organization will need to maintain additional IT infrastructure which will lead to costs, and administration requirements that can be beyond their technical capabilities. A potential solution to the problem is having intermediaries that connect to the blockchain on behalf of other organizations to offload the technical overhead. Unfortunately, this requires trust in a third party, as they will need to handle the data of the reporting organization. Another perspective is that this could lead to business opportunities for service providers that could provide services for on-premises setup and maintenance of the blockchain infrastructure. This would cost money, but remove the technical requirements.

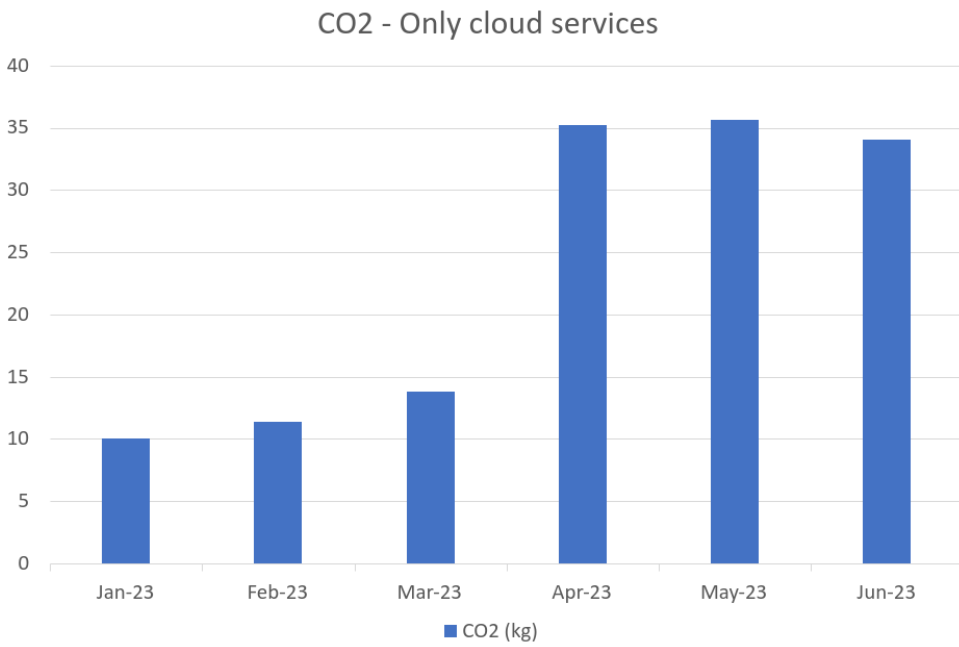

Figure 6.3: Organization-level greenwashing: CO2 emissions of Saastastic, excluding office operation.

Table 6.4: Organization-level greenwashing scenario: Registered interactions.

Date	Organization 1	Organization 2	Description
January 2023	Saastastic	JediCloud	Purchase of cloud services. (Region 1, 0.1% utilization)
February 2023	Saastastic	JediCloud	Purchase of cloud services. (Region 1, 0.12% utilization)
March 2023	Saastastic	JediCloud	Purchase of cloud services. (Region 1, 0.12% utilization)
April 2023	Saastastic	SithCloud	Purchase of cloud services. (Region 2, 0.16% utilization)
May 2023	Saastastic	SithCloud	Purchase of cloud services. (Region 2, 0.18% utilization)
June 2023	Saastastic	SithCloud	Purchase of cloud services. (Region 2, 0.17% utilization)

Skepticism and ignorance towards blockchain Blockchain is still perceived as novel technology even though it has been around since 2008. Very few people

actually understand the properties and therefore the value of blockchain technology. This makes it harder to convince people of the reasons to it should be used. Furthermore, the first application of blockchain technology which was its inception was in a cryptocurrency. Since then, a vast number of cryptocurrencies have been launched. Unfortunately, due to the lack of regulation in the cryptocurrency space, numerous scams have been conducted through sham cryptocurrencies, which has further worsened the public's perception of blockchain. Blockchain has also been associated with criminal usage, especially in the dark web. In reality, the majority of cryptocurrency users are merely investing with the hopes of earning a profit. Nevertheless, the actions of the minority have led to skepticism.

Regulations The use cases that involve the sharing of data from multiple actors across the supply chain could be impossible to implement due to regulations that impose restrictions on what data can be shared with whom. It could also restrict the duration for which certain data can be saved. Regulations could be different across the nations involved in a supply chain, where some are stricter than others. Thus in some cases, not all the data for performing aggregated computations is available. Further research into the legal concerns of sharing and storing data will need to be conducted.

6.5 Challenges

There are numerous challenges that need to be solved before a system of this nature is feasible to use in practice.

Decentralized administration The nature of having a decentralized system in which there is no single regulator, requires the operators of the network to be aligned when implementing changes. Different governing agencies could have different requirements for the system, which means that it could be hard to have one deployment of the network. In theory, separate versions of the solution could be deployed to separate networks, with their own blockchain ledgers and state. In order to collaborate across the different network instances, integration through a common agreed-upon interface could be a way to solve this.

Data accuracy If the calculations made by the system are going to provide value, they need to be as accurate as possible. This is entirely determined by the accuracy of the data that is reported into the system. Unfortunately, the world is imperfect; people make mistakes, some are dishonest, and faulty equipment can give inaccurate readings. One of the key properties of the system is the ability to perform audits on the data that has been recorded in the system, which could make it possible to uncover erroneous reporting. Of course, it is not feasible to perform an audit of each and

every organization using the system, but the audit requirements could be determined by the size of the organization. As the sustainability impact is proportional to the size of the organization, performing audits on the biggest organizations, would be the most impactful to the overall accuracy of the data in the system.

Collaboration The premise for having the ability to calculate the downstream and upstream emissions along the supply chain for a particular good or service is having the different organizations involved collaborate on the calculation by providing the relevant data. To secure the confidentiality of the data, we propose using a functional encryption scheme, which makes it possible to exchange encrypted data to be used for a particular calculation, while not revealing the data itself. However, since the data is entirely in the hands of each organization, as it is stored off-chain in their own nodes, they could in theory refuse to collaborate. The only way collaboration could be forced would be through regulations and laws. The problem is that the supply chain is global and is not bound to any single government, which means that it is hard, if not impossible, to adopt and enforce such a law.

6.6 Discussion

The primary objective of implementing the proof-of-concept was to determine if using blockchain for sustainability reporting could aid in preventing or revealing greenwashing by making it possible to determine the sustainability impact of the goods and services sold by businesses. Does the solution provide the functionality to achieve this? In theory, if every organization along the supply chain accurately reported their business activities, assets, and their associated sustainability impact, we would have the data required to calculate the emissions associated with a product or service throughout the entire supply chain.

Unfortunately, it is infeasible to realize this in practice. It would require far too many resources to enforce its usage at the global scale unless some other non-regulatory incentive existed that made businesses use the system voluntarily. As climate change increases and the consequences become even more impactful, the world may become even more united in the effort against climate change. Should the world ever implement a ubiquitous requirement for sustainability reporting to fight climate change, our implementation could serve as a potential inspiration.

Although it is unrealistic that this system could be implemented globally it could still provide value; sustainability impact could be reported to the system using a combination of today's approximation methods when required. Organizations that use the system are able to accurately determine their scope 1 and 2 emissions, which they will report when registering business activities. In case their suppliers or customers do not use the system they cannot register their interactions that serve as

the basis for scope 3 emissions. Instead, they can fall back to the traditional methods for approximating.

Chapter 7
Conclusion

This thesis has been an exploration into the potential application of blockchain for sustainability reporting as a way to prevent greenwashing. Our hypothesis was that the properties of blockchain technology would facilitate a solution that provides transparency and auditability to the supply chain when used by its actors.

The main contribution of this thesis is a proof of concept application that implements sustainability reporting running on top of the Hyperledger Fabric blockchain operating system. Fabric implements a permissioned blockchain that is designed for industry applications at scale. It provides privacy features that we have used extensively in our solution that ensure that the reported sustainability data is stored securely, and only accessible to the organization to which it belongs. The private data is stored off-chain at the nodes of the organization, and the hash of the data is recorded on-chain which serves as immutable evidence, which can be used for auditing purposes.

An open system that is industry-agnostic could help us achieve interoperability throughout the supply chain by providing a common interface for sustainability reporting. Having interoperability lets us exchange information in a standardized way, which opens up the possibility for collaboration. We propose to explore the use of functional encryption as a secure way for actors in the supply chain to collaborate by exchanging data in an encrypted format that can be used for computing aggregated data while preserving confidentiality. This could in theory be used to accurately calculate scope 3 emissions, as opposed to estimating using quantitative models and reference values. Vendors could hypothetically provide accurate sustainability information to consumers for the products and services they were selling, which could greatly reduce the occurrence of greenwashing.

Unfortunately, the idea of having a standard system used globally is an unachievable goal. Currently, there are far too many barriers. There is a lack of regulation, which would be required to impose the use of a standard system. The difference

in technical competence and infrastructure is far too great between nations, and there needs to be an incentive for businesses to use such a system. While the system could be used at the national level, the interconnectedness of the world economy means that the lack of information from the rest of the supply chain would render the collaborative aspect greatly impaired.

Despite the barriers, our proposed solution could one day be feasible to implement. The world is changing rapidly, and one of our greatest challenges is climate change. Perhaps someday, the world would be aligned and technologically mature enough that a system of this nature could be used.

7.1 Future work

Data removal The key defining property of the blockchain is immutability, which means that once data has been added to the ledger, it cannot be removed. This can be problematic if the data that is stored in the ledger needs to be deleted due to external factors such as regulations. If a business located in the EU inputs data that is considered personal information, it will need to be compliant with GDPR. Consequently, the personal data cannot be stored in their systems indefinitely. Luckily, Hyperledger Fabric has built-in functionality that lets us prune data that is stored off-chain, while the only thing remaining is the hash of the data on-chain. The problem with data removal is that it could lead to holes in the data set required to carry out calculations. However, it could be implemented in a way that preserves sustainability data which would not constitute personal data.

Testing Testing is an important methodology in software engineering that help us ensure that business logic is implemented correctly and that the system is working as intended. In our solution, business logic can be tested by writing tests for each of the implemented methods of the contracts. Since the contracts interact with the domain layer, this will also test the behavior of the domain entities. Mocks should be used in place of infrastructure code such as Hyperledger SDKs in order to isolate the tested functionality. Since the solution is implemented in TypeScript, we propose using the popular testing framework JEST [1].

Extending sustainability outcome support The sustainability impact that is associated with a business activity is modeled as a set of outcomes in our domain model. In the code, the set of outcomes is the set of value objects that extend the `Outcome` base class. Currently, the solution provides very limited support for outcome types: `ElectricityConsumed`, `ElectricityProduced`, and `GreenhouseGasEmitted`.

[1] JEST is an open-source testing framework: https://jestjs.io/

Further research into the three pillars of sustainability and potential ways to model outcomes of business activities that affect aspects of them should be explored.

Fractional participation in activity We discovered a new requirement during the evaluation. Currently, it is not possible to specify the amount of participation in an activity. This is needed if we for example want to model a shared service consumed by multiple customers, e.g. cloud computing, but the amount of resource utilization by each customer is not the same.

Secure data sharing The key value proposition of the solution is to provide insights into the supply chain by means of aggregating the data in the system. Functional requirements FR8 and FR9 are examples of scenarios where this functionality is needed. Our proposal is to traverse the organizations along the supply chain which hold data required for the given calculation. This requires us to implement a protocol that lets data be shared for use in aggregation calculations while protecting the data. We propose exploring the use of functional encryption which seems to be a good fit for the described protocol.

CouchDB as world-state database Because calculating the total sustainability impact requires aggregating all business activities and assets that are present along the supply chain of the organization in question, it can potentially require performing many database reads. Without the ability to perform parameterized queries on the entities in the world-state database, a full scan of the entire database along with filtering implemented in the application is required every time we wish to obtain a subset of the entities based on some predicate or to perform aggregation. The world state database, CouchDB, provides indexing features that allow us to perform parameterized queries and optimize performance. Instead of using LevelDB, CouchDB should be used. A mapping of the data access pattern should be conducted to establish what indices are needed; unnecessary indexing should be avoided, as it will degrade write performance and take up unnecessary storage.

User-facing frontend Since our application is intended to be used by consumers and NGOs to help them detect greenwashing, a publicly accessible frontend should be developed and deployed. This can be a traditional web application that is hosted in the cloud. The backend of the web application will be needed to integrate and connect to our blockchain application and will serve as the facade.

References

[21] *Regulation (EU) 2021/1119 of the European Parliament and of the Council of 30 June 2021 establishing the framework for achieving climate neutrality and amending Regulations (EC) No 401/2009 and (EU) 2018/1999 ('European Climate Law')*, Jul. 2021. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32021R1119.

[23a] «Apache couchdb». Online; accessed 15-May-2023, The Apache Software Foundation. (2023).

[23b] «Google/leveldb leveldb: Is a fast key-value storage library written at google that provides an ordered mapping from string keys to string values». Online; accessed 15-May-2023, Google. (2023).

[ABB+18] E. Androulaki, A. Barger, *et al.*, «Hyperledger fabric: A distributed operating system for permissioned blockchains», in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[ACF+19] S. Agrawal, M. Clear, *et al.*, *Ad hoc multi-input functional encryption*, Cryptology ePrint Archive, Paper 2019/356, https://eprint.iacr.org/2019/356, 2019. [Online]. Available: https://eprint.iacr.org/2019/356.

[But+14] V. Buterin *et al.*, «A next-generation smart contract and decentralized application platform», *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[Com] E. Commission. [Online]. Available: https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/corporate-sustainability-reporting_en.

[DB11] M. A. Delmas and V. C. Burbano, «The drivers of greenwashing», *California Management Review*, vol. 54, no. 1, pp. 64–87, 2011. [Online]. Available: https://doi.org/10.1525/cmr.2011.54.1.64.

[DH76] W. Diffie and M. Hellman, «New directions in cryptography», *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[EFR] EFRAG, *First set of draft esrs*. [Online]. Available: https://www.efrag.org/lab6.

[Eur] European Committee of the Regions, *Green Deal Going Local*. [Online]. Available: https://cor.europa.eu/en/engage/Pages/green-deal.aspx?utm_source=SharedLink&utm_medium=ShortURL&utm_campaign=Green+Deal+Going+Local.

[Eva04] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[Foua] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/network/network.html.

[Foub] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/membership/membership.html.

[Fouc] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/membership/ledger.html#ledger.

[Foud] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/ledger/ledger.html#transactions.

[Foue] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/txflow.html.

[Fouf] H. Foundation. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/private-data/private-data.html.

[HKG+20] A. Hasselgren, K. Kralevska, *et al.*, «Blockchain in healthcare and health sciences—a scoping review», *International Journal of Medical Informatics*, vol. 134, p. 104 040, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138650561930526X.

[HKGF21] A. Hasselgren, K. Kralevska, *et al.*, «Medical students' perceptions of a blockchain-based decentralized work history and credentials portfolio: Qualitative feasibility study», *JMIR Form Res*, vol. 5, no. 10, e33113, Oct. 2021. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/34677137.

[HW18] E. G. Hertwich and R. Wood, «The growing importance of scope 3 greenhouse gas emissions from industry», *Environmental Research Letters*, vol. 13, no. 10, p. 104 013, Oct. 2018. [Online]. Available: https://dx.doi.org/10.1088/1748-9326/aae19a.

[HWH+20] A. Hasselgren, P. K. Wan, *et al.*, «GDPR compliance for blockchain applications in healthcare», *arXiv preprint arXiv:2009.12913*, 2020.

[IfS11] W. R. Institute and W. B. C. for Sustainable Development, *Greenhouse gas protocol: Corporate value chain (scope 3) accounting and reporting standard: Supplement to the GHG protocol corporate accounting and reporting standard*, en. 2011.

[LHK+22] S. Liu, G. Hua, *et al.*, «What value does blockchain bring to the imported fresh food supply chain?», *Transportation Research Part E: Logistics and Transportation Review*, vol. 165, p. 102 859, 2022.

[LTIW23] Y. Li, C. Tan, *et al.*, «Dynamic blockchain adoption for freshness-keeping in the fresh agricultural product supply chain», *Expert Systems with Applications*, p. 119 494, 2023.

[Nak09] S. Nakamoto, «Bitcoin: A peer-to-peer electronic cash system», May 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf.

[Pro11] G. G. Protocol, «Greenhouse gas protocol», *Sector Toolsets for Iron and Steel-Guidance Document*, 2011.

[RGK+20] J.-A. H. Rensaa, D. Gligoroski, *et al.*, «Verifymed-a blockchain platform for transparent trust in virtualized healthcare: Proof-of-concept», in *Proceedings of the 2nd International Electronics Communication Conference*, 2020, pp. 73–80.

[RGK19] M. Raikwar, D. Gligoroski, and K. Kralevska, «SoK of used cryptography in blockchain», *IEEE Access*, vol. 7, pp. 148 550–148 575, 2019.

[SC21] A. I. Sanka and R. C. Cheung, «A systematic review of blockchain scalability: Issues, solutions, analysis and future research», *Journal of Network and Computer Applications*, vol. 195, p. 103 232, 2021.

[SW23] A. Stenzel and I. Waichman, «Supply-chain data sharing for scope 3 emissions», *npj Climate Action*, vol. 2, no. 1, p. 7, 2023.

[Sza96] N. Szabo, «Smart contracts: Building blocks for digital markets», *EXTROPY: The Journal of Transhumanist Thought,(16)*, vol. 18, no. 2, p. 28, 1996.

[UN] UN, THE 17 GOALS | Sustainable Development, [Online]. Available: https://sdgs.un.org/goals, (last visited: Nov. 15, 2022).