Master's Thesis in Media Informatics

Trusted Data Aggregation in Distributed Ledger Technology

Rheinisch-Westfälische Technische
Hochschule Aachen

Merve Sahin

October 7, 2021

Examiners:
Prof. Dr. Thomas Rose
Prof. Dr. Wolfgang Prinz

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Sahin, Merve — Name, Vorname/Last Name, First Name

399511 — Matrikelnummer (freiwillige Angabe) / Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit/Bachelorarbeit/~~ Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present ~~paper/Bachelor thesis~~/Master thesis* entitled

Trusted Data Aggregation in Distributed Ledger Technology

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Duisburg, — Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Duisburg, — Ort, Datum/City, Date

Unterschrift/Signature

Acknowledgements

Foremost, I would like to thank my Professor Thomas Rose for giving me this opportunity to work on such an interesting and exciting topic. He successfully upheld an environment in which I could contribute my own ideas on the subject. I appreciate his and Thomas Osterlands continuous guidance and help throughout the thesis.

I would also like to express my gratitude to my friends and family, who supported and motivated me throughout the whole process. They have always lent me an ear and shown interest in my work. A special thanks to my sister, Meryem, who encouraged me to pursue a masters degree and continuously pushed me to achieve more.

Abstract

With the rising popularity of Blockchain technology, Supply Chain Management falls more and more under the scope. Open collaboration and the flow of information across company boundaries plays a vital role, but is facing serious challenges due to a lack of standardization, transparency and trust. Stakeholders in the supply chain and end customers should be able to get insight into the origins of data contributions. Distributed Ledger Technology (DLT) is a promising solution for establishing trust and transparency without the reliance on a third party. It is an umbrella term for an ecosystem comprised of interconnected distributed systems, where data is permanently added based on a multi-party consensus mechanism. Blockchain, which conforms to a DLT, is not intended for storing large amounts of data, hence scalability becomes a serious impediment. Therefore, the thesis proposes a system architecture for a self-sovereign ecosystem, which bridges the communication between external stakeholders without encumbering the underlying Blockchain. The main idea is to track data contributions using Blockchain transactions, where raw data is stored off-chain (i.e. in databases) and its cryptographic hash stored on-chain (i.e. in smart contracts). This mechanism not only reduces the size of data, but also protects its privacy. The goal of this thesis is to apply this approach to the use case of inland waterway transportation, which already lacks digitization. The realized prototype uses the Ethereum Blockchain and defines a set of standardizations & communication protocols. Hence, the exchange of information between external stakeholders is efficiently automated. Moreover, document exchanges are digitized and encoded into smart contracts, which additionally serve the purpose of automating payments and fines.

Contents

List of Abbreviations

B/L	Bill of Lading
BFT	Byzantine Fault Tolerance
C/P	Charterparty
DAG	Directed Acyclic Graphs
DApp	Decentralized Application
DHT	Distributed Hash Table
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
ENS	Ethereum Name Service
ETH	Ether
IDM	Identity Management
IoT	Internet of Things
IPFS	Interplanetary File System
JWT	JSON Web Token
MitM	Man-in-the-middle
OAuth	Open Authorization
P2P	Peer-to-Peer
PoD	Proof of Delivery
POS	Proof-of-Stake
POW	Proof-of-Work
REST	Representational State Transfer
URI	Uniform Resource Identifier

List of Figures

List of Tables

Listings

1. Introduction

Distributed Ledger Technology (DLT) opens up new opportunities for business collaborations, where untrusted parties can cooperate more openly. A DLT is an ecosystem comprised of interconnected distributed systems, where data is permanently added based on a multi-party consensus mechanism. It represents a tamper-resistant distributed ledger for record-keeping. This technology serves as an ideal tool for waterway transportation, where trusted tracking of data exchanged between stakeholders is enabled.
Hinckeldeyn presents existing Blockchain-based solutions realized within the supply chain process. One such example is the management of freight carriers: a consortium of 35 german companies and research institutions, including DHL, Deutsche Bahn & Kaufland, have joined forces to manage the exchange of their freight carriers. This exchange involves the participation of various unknown parties and has high administrative expenses. The prototype, which is implemented atop the MultiChain[1] Blockchain, is used by employees at the logistics center and by administrators. The project contributed to an overall increase of efficiency and simultaneously established a standardization in the exchange of information [11].
Similarly, Roeck researches the potential use cases and benefits of the use of DLT in supply chain. He conducts a literary research and interviews DLT providers & users in the supply chain. The result of the research reveals the most important demands for this industry: *transparency, availability, authenticity* and *trust* [22]. Seeing that collaborating businesses generate large amounts of data along the supply chain process, a need for trusted data aggregation arises. The waterway transportation use case as part of the supply chain process serves as an ideal example for the scope of this thesis. The article by Müller from 2018 analyzes the latest developments of inland waterway transportation, which has been experiencing a significant decline in cargoes for the last two decades, while competing transportation means, such as freight trains and road transportation, have been seeing a two-fold increase. Waterway transportation consumes minimal resources, thus making it an eco-friendly alternative and serving as an incentive to improve upon it [16]. Inland waterway transportation still needs to catch up with digitizing their processes, as most transport orders are paper-based and sometimes even orally agreed upon. SINLOG [23], which stands for *Standardization Approach to*

[1] https://www.multichain.com/

Connect Inland Navigation to Intermodal Logistics, is a project that aims to digitize paper-based document interchanges and automate specific tasks with the intent of accelerating existing business processes. Furthermore, the use of automatic systems, like water displacement sensors and port cranes, open up a potential of integrating a machine-to-machine economy.

1.1 Problem Statement

The collaboration of external stakeholders in the supply chain makes tracing data challenging. End customers and stakeholders should be able to get insight into the origins of data contributions, for example for the processing of shipped goods. Unfortunately, most processes within the supply chain lack digitization. This increases the risk of losing paper documents and facilitates the forgery of documents, fraud or theft.
Stahlbock et al. and Goudz and Ahmad present existing Blockchain-based solutions *TradeLens & CargoX*. TradeLens was developed by IBM in cooperation with Maersk and is based on the open-source permissioned Hyperledger Fabric[2] Blockchain. Hence, it is not only operated by IBM, but is also inaccessible to non-member stakeholders. The platform does not give stakeholders direct access to the Blockchain but rather provides an API, which communicates with an off-chain service. Furthermore, the focus of TradeLens lies on container shipping, whose process rather differs from bulk-loading (e.g. vessel inspection). As opposed to TradeLens the focus of CargoX is to track & ensure the rightful ownership of goods by transferring the Bill of Lading between stakeholders. This scenario is generally futile for bulk-loading, as the ownership of goods is not transferred until they reach the customer [26, 8]. These limitations give reason for the implementation of a publicly available DLT-based solution, that is more suited for the use case of bulk-loading.

The intention is to implement a solution for the use case of inland waterway transportation with focus in bulk-loading of grain. There are five challenges, which need to be overcome in order to realize a solution. The first challenge is to figure out how to reliably and trustfully aggregate data, that originates from a multitude of external stakeholders. The second challenge deals with the issue of scalability. In general, supply chain processes generate large amounts of data. This poses an issue for current Distributed Ledger Technologies as scalability is a problem. DLT transactions with high volumes of data tend to be more expensive and use up more space in each network node, which is not sustainable long term. Therefore, the second challenge of this thesis is to design a solution for data aggregation, where high volumes of data are handled efficiently without putting a strain on the underlying DLT. The third challenge addresses the concern about data privacy in a public DLT, which is especially magnified for the automation in smart contracts, where access to raw data is essential. The fourth challenge is to design an archi-

[2]https://www.hyperledger.org/use/fabric

tecture with interoperability in mind, in order to facilitate the participation of new external stakeholders. The last challenge is targeted at the handling of the quality of grain. The moisture level of grain and hygiene of vessels contribute to its quality and thus need to be addressed such that the quality is preserved.

1.2 Objectives

Osterland and Rose present an approach that aims to address the issue of scalability by introducing an ecosystem of on- and off-chain components. The latter is intended for storing raw data and bridges the interaction of the user with the underlying Distributed Ledger Technology. The integral idea is to generate hashes from a group of raw data, store the hashes on-chain and the raw data off-chain. This way, the underlying DLT does not get cluttered with raw data, but merely with a single hash that is essentially a reference to it. Furthermore, the authors present a component for identity management, in order to properly assign user identities behind anonymous DLT accounts [17].

The primary objective of this thesis is to apply the aforementioned approach to the specific use case of inland waterway transportation. This entails the adaptation of the proposed ecosystem to the use case, where it will consist of a predefined set of stakeholders and their system environments. The solution should aim to exhibit at least three characteristics: digitization of paper documents, automation of various steps and establishment of trust between external stakeholders. In order to realize these properties, the challenges in the previous section need to be addressed. The issue of scalability becomes redundant, as it is indirectly solved by the proposed approach, e.g. with the introduction of off-chain data-storages. The remaining challenges need to be tackled with the help of a technical and literary research. Therefore the following research questions, derived from the challenges, will be addressed in the impending study:

1. What kind of data needs to be aggregated in the use case of inland waterway transportation?
2. How can data, that comes from a multitude of independent sources, be reliably and trustfully aggregated?
    - 2.1 How can data be shared efficiently and consistently between stakeholders?
    - 2.2 How can the authenticity of shared data be established?
    - 2.3 How can the integrity of shared data be verified?
3. How can the privacy of data be protected on a public DLT?
4. How can the quality of grain be controlled?
    - 4.1 Which factors affect the degradation of the quality?

4.2 What measures need to be taken in order to preserve the quality?

5. How can an interoperability within the ecosystem be accomplished?

1.3 Scope & Outline

The realization of the solution will be carried out in a smaller scale. Therefore, the scope of the use case scenario is limited to shipments between two companies using two inland ports as a loading and discharging port. The stakeholders include: trading company, shipowner company and port agencies.

The next chapter introduces the use case of inland waterway transportation in detail, where the process, the stakeholders and their tasks are discussed. The third chapter presents the State of the Art of DLT and its methodologies & standardizations. The fourth chapter reviews related work in the domain of waterway transportation and presents relevant off-chain approaches. The fifth chapter outlines the system architecture, where each system component and its connections are examined. The sixth chapter demonstrates the user journey, where each interaction is shown and its background process explained. The seventh chapter identifies evaluation criteria and analyzes the abilities and limitations of the proposed approach and its implementation. Lastly, the final chapter summarizes important findings, concludes the the thesis work and discusses the future outlook.

2. Waterway Transportation Use Case

In this chapter, an insight into the foundations of waterway transportation will be provided and their business processes, relevant for this thesis, discussed.

This specific use case of waterway transportation involves the shipment of different types of grain between inland ports. The shipment of these goods is considered as part of the supply chain, where companies process a combination of grains into ingredients intended for consumption. Considering that the supply chain presupposes the collaboration of various actors from different companies, trust and reliability become important criteria. The transportation processes in particular have many vulnerabilities. Currently, almost all maritime transportation processes lack digitization since all processes are paper-based. Therefore, these documents can either get lost or even forged. Moreover, processes are delayed due to documents not arriving timely at customs or other places. Proper provenance tracking would eliminate these issues, as suppliers would know the location, certification and ownership of the transported goods.

The paper by Philipp et al. provides a great insight into the potential use of *smart contracts* in *maritime transport* and briefly discusses the steps and actors involved in the shipment process. The authors make a distinction between the process performed by the actors working at the port, and the process performed by actors involved in the negotiation of a shipment and its execution [18].
The shipping procedure is typically preceded by a charterparty (C/P) between the trading company and the shipping company. The charterparty is a contract between these two parties negotiated by a charterer, who is either the trader itself or works for the trading company. The charterer's responsibility is to first find a shipping company, who is able to provide shipping services for their freight requirements. This particular task is delegated to a shipbroker, who is provided with freight information and investigates the market situation to find a suitable shipping company. The given information includes features such as cargo type, quantity, physical dimensions, loading & discharging ports and expected lay days & cancellation days. Typically, the shipbroker contacts multiple companies so that the charterer can negotiate a deal in the best interest of the trading company. This entire procedure is demonstrated in a simplified manner in figure 2.1. As shown in the figure, the charterer is presented with multiple offers, of which they pick the

one that appeals to them the most. This is succeeded by a discussion on the terms of their contract until both parties reach an agreement and sign a charterparty. This charterparty is a prerequisite for initiating a shipment process, as it contains significant information about the cargo and is therefore important for the use case scenario.

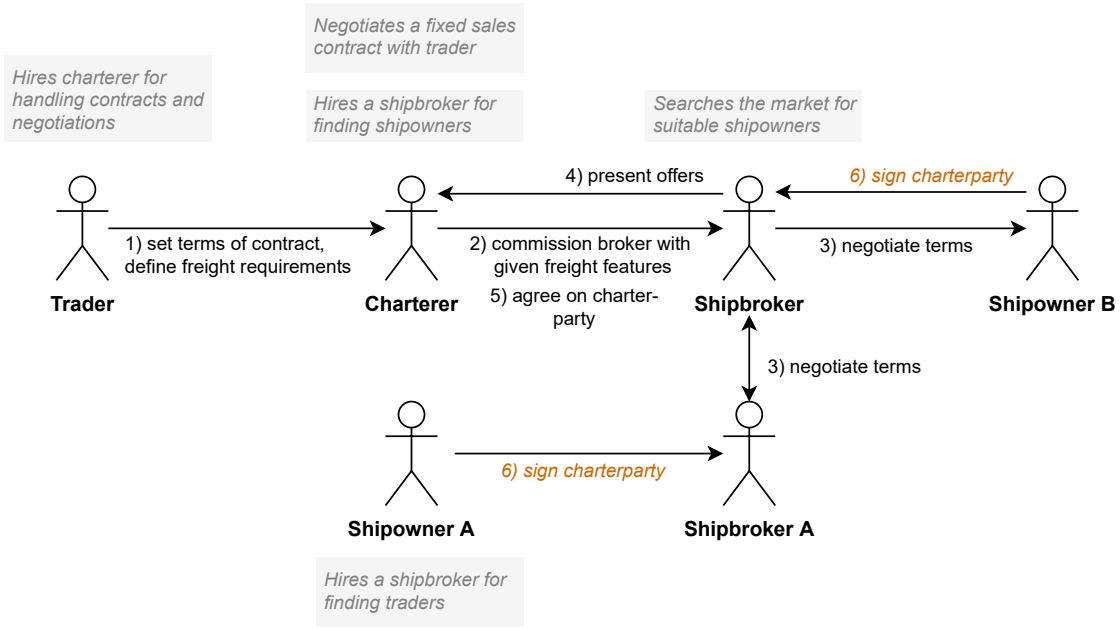

Figure 2.1: Charterparty Agreement Procedure

The Charterparty is a prerequisite for issuing any additional digital documents related to shipments. Typically, after a bulk is loaded onto the vessel, a bill, stating information about the shipment, location etc., is issued and signed. This bill is referred to as the *Bill of Lading* (B/L) and is generally issued by masters or port agents. Similarly, the receipt of shipments upon unloading the bulk is recorded as well and is referred to as the *Proof of Delivery* (PoD).
Considering that both parties cannot be physically present at the ports for auditing each shipment, they instate port agents at the loading and discharging ports. The audit involves the monitoring of loading/unloading sub-processes, vessel inspection, stevedores, previously mentioned digital documents, etc. In addition to auditing, the agents also report on the progress & status of the shipment.
The aforementioned documents describe the type of information needed for aggregating data and hence answer research question 1. To summarize, there are four document exchanges which need to be tracked in the following order: Charterparty, Certificate (for vessel inspection), Bill of Lading, and Proof of Delivery.

Table 2.1 lists the relevant actors involved in the use case and an abstract description of their tasks.

Actor	**Tasks**
Trader / Charterer	• *Agree & sign on charterparty* • Initiate a shipment process • Assign port agents
Shipowner	• *Agree & sign on charterparty*
Master	• Transport vessel • Clean vessel
Port Agent	• Issue B/L or PoD • Check B/L or Certificate • Report progress to charterer & shipowner • Inspect vessel • Monitor/measure moisture level of grain • Monitor/measure bulk weight
Customer	• Verify provenance of shipments

Table 2.1: Use Case Actors and their Tasks

The tasks *Agree & sign on charterparty* will not be fully implemented, as the charter-agreement process is less relevant for the scope of this thesis. However, the information contained in a charterparty are used as input for creating shipping processes. Therefore, a smart contract encoding the charterparty will be implemented and generated automatically with fixed parameters. Likewise, the B/L and PoD will also be implemented as smart contracts. The charterparty smart contract can be used to issue multiple B/Ls, and each B/L can be used to issue a PoD. In reality, these documents contain many legal clauses and can therefore become highly complex. However, in the interest of not violating the scope of this thesis a more simplified approach will be taken. The goal is to digitize only information that are relevant for the process and therefore legal information (i.e. terms and conditions) will not be considered.

To answer research question 4.1 one should analyse the shipping conditions and its process more in detail. The inspection of the vessel and grain quality is of high importance, given that the supplier expects the grain to be in good condition. The online article by Wong describes the transportation of wheat in detail and gives concrete criteria for evaluating its quality. The port agent needs to first check whether the shipowner is authorized to ship grain, which is typically validated with a *document of authorization*. In addition to that, the agent either asks for a cargo quality certificate, which is most likely issued by the trader, or samples the grain in order to check the quality first hand. During the sampling, the agent checks the grain for objectionable odors or insects, sieves the sample for visual inspection, and determines the grains moisture and grading factors [32]. Table 2.2 shows a classification of water content (moisture level) in wheat.

Water content	Up to 15%	15-16%	16-17%	17-18%
Designation	Dry	Medium Dry	Moist	Wet

Table 2.2: Classification of Water Content in Wheat [32]

The lower the water content, the higher the quality of grain. Nevertheless, the quality can quickly degrade due to environmental factors such as ventilation, temperature, air humidity and weather. The preferred holds temperature for preserving the grains quality is approximately 20°C and at temperatures between 20 and 30°C molds reach an optimum activity. These values can be encoded in smart contracts, in order to make the shipowner accountable for not ensuring correct environmental conditions.
After sampling, the agent carries out a vessel inspection, in which the holds is checked for potential defects such as water, mold, rust, insect infestation, sludge, chemicals, etc. If the holds is considered *'grain clean'* the agent issues an inspection certificate, forwards it to the shipowner and gives them the green light to proceed with the shipment. Once the shipment arrives, the port agent at the discharging port checks the holds condition before unloading and samples the grain. The agent possibly also issues a certificate for the holds inspection.
In conclusion, there are two types of factors which can degrade the quality of grain: environmental factors (temperature, humidity, etc.) and hygienic factors (chemicals, mold, insects, etc).

All actors and tasks presented in table 2.1 have been visualized in an activity diagram as shown in figure 2.2. Actions, which have been greyed out, are outside of the implementation scope, but were depicted in order to provide more context. Typically, the shipping process would be initiated by the customer placing an order to the trader. The charterer, who works for the trader, would then handle the charterparty agreement between the shipowner and notify them about a new shipment. Once the master arrives with the vessel at the intended port, the agent starts with the inspection of the vessel. The agent issues a certificate if the vessel is considered clean and continues with the measurement of the moisture level and weight. The captured data is then recorded in a new B/L. After the B/L is successfully issued, the master transports the goods to the discharging port. Upon arrival at the port, the agent examines the vessel and measures both the weight and moisture level. The results are recorded in a new PoD and the trader is notified about the status of the shipment. Subsequently, the trader forwards the newly received information to the customer, who is then able to verify the provenance of the provided data.

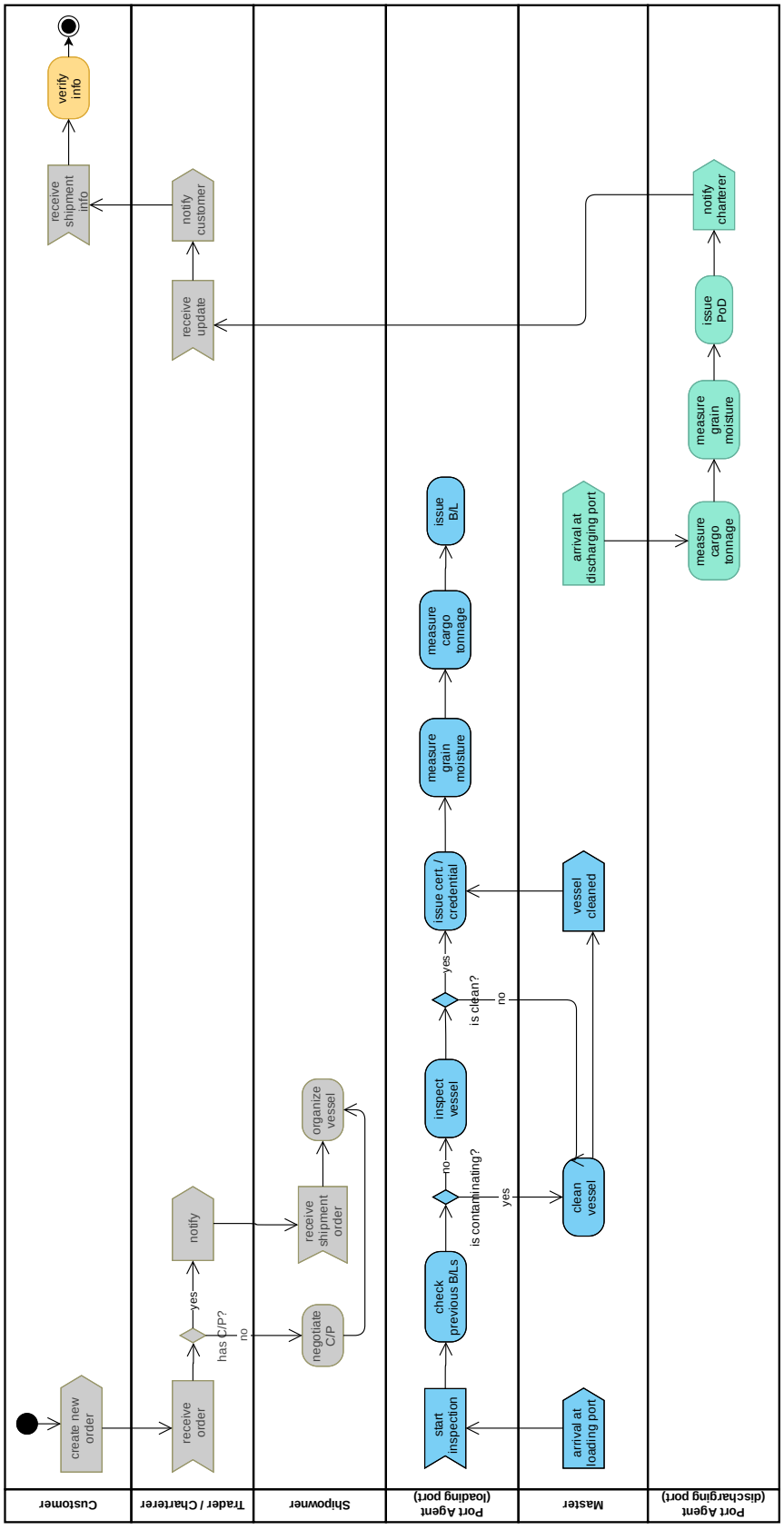

Figure 2.2: Activity Diagram of a Shipping Process

The following chapter introduces the State of the Art of Distributed Ledger Technologies. It presents a more in-depth look into the concept and inner workings of a DLT, while also examining different types of implementations and their limitations. Furthermore, it presents new standardizations for DLTs, which are important enablers for interoperability.

3. State of the Art

3.1 Distributed Ledger Technology

According to the book *Distributed Ledger Technology Systems: A Conceptual Framework* by Rauchs et al., "Distributed ledger technology (DLT) has established itself as an umbrella term to designate multi-party systems that operate in an environment with no central operator or authority, despite parties who may be unreliable or malicious ('adversarial environment')" (p. 12) [19]. In other words, it is regarded as an autonomous consensus-driven multi-party system, wherein members need to reach an agreement over data and its validity. The differences of a DLT compared to distributed databases lie in its design capabilities, that support data and their integrity in an adversarial environment. The authors present the following properties that a DLT system needs to adhere to:

1. **Shared record-keeping:** multiple parties should have the possibility to create, maintain and update shared ledgers (authoritative records)
2. **Multi-party consensus:** all parties have to agree on a set of records
3. **Independent validation:** each party should have the possibility to verify the state and integrity of transactions
4. **Tamper evidence:** each party should be able to detect modifications on data
5. **Tamper resistance:** it should be difficult for a single party to change the transaction history

The authors dive deeper into the meaning of *shared ledgers* and present the key concepts which form its building blocks. Assets, be it in economic or non-economic nature, are transferred using *transactions*, which need to be validated by the nodes of the network. A set of validated but unconfirmed transactions are initially kept in a *log* of a node, awaiting to be subject to consensus rules by other nodes of the network. Once these have been confirmed by the network, they are assembled into a *record* at the consenting nodes. Each node of the network owns a *journal*, containing a set of records, which may or may not contain all the same records. A *ledger* is defined as the collection of authoritative records held by a considerable

proportion of the network nodes at all times. Hence, the likelihood of these records being erased or changed are incredibly low. Figure 3.1 illustrates the key concepts in form of a pyramid.

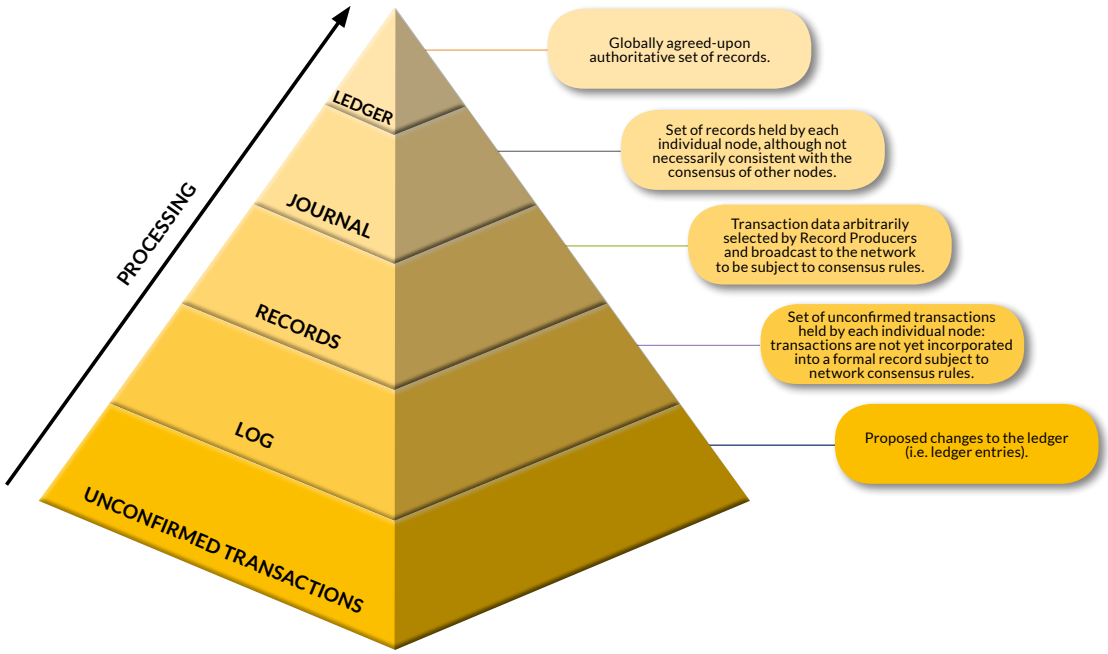

Figure 3.1: From Transactions to Records [19]

3.1.1 Merkle Tree

Concerns about the lack of scalability and privacy in DLTs like Blockchain make it challenging to fully encode business processes in smart contracts. Therefore, a mechanism for anonymizing & aggregating raw data and verifying its integrity is essential. Bitcoin and Ethereum use *Merkle Trees* to verify the integrity of their blocks. This particular algorithm can also be used in different scenarios, one of which will be explained in chapter 6.2. What makes merkle trees particularly special is their ability to circumvent scalability issues. Vitalik Buterin, the founder of the Ethereum foundation, explains the workings of a merkle tree in his blog [5]. A merkle tree is constructed using a series of raw data, which essentially represent the leaves of the tree. Each leaf is hashed in pairs with its neighbor and its result is rehashed with the hash of the neighboring node until a single hash, the merkle root, is obtained. The result is a binary tree as depicted in the figure 3.2.
A piece of information can be verified without the knowledge of all the raw data. The verifier is required to have knowledge about the merkle root, which will later be used for comparison. The figure shows a scenario, in which the green node represents the piece of information available to be verified. In order to reconstruct the tree, the verifier only needs to know the nodes highlighted in yellow. Once the tree is reconstructed, the verifier can compare the resulting root with the merkle

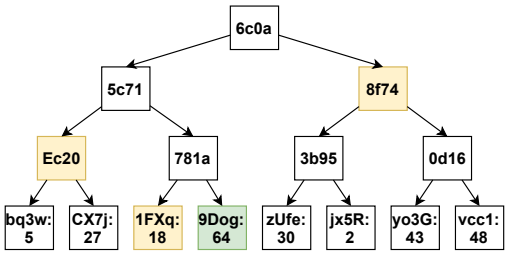

Figure 3.2: Merkle Tree [5]

root that was already known.
Off-chain data aggregation can be accomplished by storing the merkle root on the Blockchain and storing the proof, i.e. the complete tree, off-chain. This way the aggregation scheme is not only tamper-resistant but also privacy conserving, scalable and accessible at all times.

3.1.2 Smart Contract

According to Rauchs et al., smart contracts are 'Programmatically-executed Transactions (PETs)', that reside on the DLT and are executed upon being triggered by a message. The behaviour of a smart contract is defined and agreed upon by all affected and untrusted parties, hence the name 'Contract'. What makes them special is the lack of intervention from 3rd parties. Typical use cases of smart contracts are for the replacement of fiduciary services such as custody or escrow [19].
Wang et al. discuss the architecture and trends of smart contracts. Popular DLT platforms, that support smart contracts are Ethereum and Hyperledger Fabric. The authors suggest that smart contracts possess the following properties: autonomy, self-sufficiency and decentralization. The first property, autonomy, indicates that a smart contract becomes detached from the initiating agent after being deployed and thus can act autonomously from there on. Self-sufficiency describes the ability of a smart contract to marshal its own resources, for example raising funds by offering services. The authors also discuss recent challenges and vulnerabilities of smart contracts, one of which is called the reentrancy attack. It exploits the fallback transaction function, which makes recursive calls for withdrawing money from a contract and thus re-enters the contract multiple times. Another issue is the transaction-ordering dependence (TOD), where the order of multiple consecutively executed transactions invoking the same contract become unreliable, since miners can process transactions in a different order. Furthermore, the lack of trustworthy data feeds can also be problematic, when for example an oracle is used to query the current price of virtual assets [30].
Figure 3.3 shows the operating mechanisms of smart contracts in a Blockchain. The in-going arcs represent actions performed by the deploying agent, i.e. user. The user can define conditions & the corresponding actions and trigger certain events by externally calling functions. The out-going arcs represent the behaviour of the contract based on the condition it fulfills. The smart contract itself adds a

new transaction to the block whenever its state is updated by an event [30].

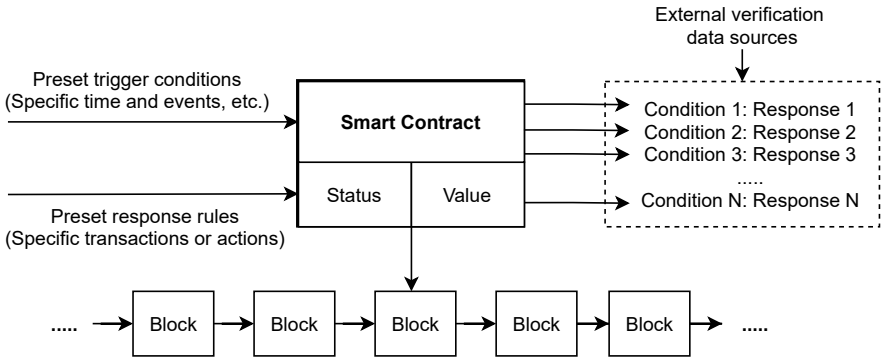

Figure 3.3: The Operating Mechanisms of a smart contract [30]

The paper "From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild" by Akhtar reviews types of existing DLT implementations and discusses their advantages & limitations [2]. These will be presented in the following sections.

3.1.3 Blockchain

The most commonly used and known DLT implementation is *Blockchain*. As its name already implicates, transaction data are stored in form of blocks, which are chained using cryptographic hash algorithms. Each block contains the hash of the previous block and each node in the network has a copy of this chain. Therefore making changes to historic data will break the chain and expose the tampering. Blocks are the equivalent to *records* presented in figure 3.1 and the entire Blockchain itself is regarded as a *ledger*. The two major Blockchain technologies, Bitcoin[1] and Ethereum[2], use the *proof-of-work* (POW) consensus algorithm for adding blocks to the chain. The POW algorithm requires every member, so called *miners*, to perform a complex mathematical calculation. The member who solves it first gets to add the new block to the chain and is in turn rewarded with cryptocurrency. These calculations require the use of special hardware and not only consume significant power but also take longer to complete, which is why Ethereum plans to switch to the *proof-of-stake* (POS) consensus algorithm. In this algorithm, each member needs to stake a minimum amount of cryptocurrency, i.e. Ether (ETH), to become eligible for *mining*. The *mining* process involves the creation or validation of blocks, where the algorithm randomly selects a miner. The miner, who is selected creates a new block, while the other miners validate blocks. Unlike Bitcoin, Ethereum is also able to make use of *smart contracts* and therefore not only establishes a platform for trustful transaction of assets, but also for trustful collaboration.

[1] https://bitcoin.org/en/
[2] https://ethereum.org/en/

3.1.4 Sidechain

According to Akhtar, "Sidechain combines two different Blockchain architectures to remove existing shortcomings of Blockchain in terms of performance, privacy and security" [2]. The paper by Back et al. introduces the underlying workings of the proposed technology of sidechain. It exists alongside the main Blockchain and is described as a sub-network, where requests (i.e. sidechain transactions) are managed locally. The transfer of digital assets from the main chain to the local chain and vice-versa is done via the so called *two-way peg* mechanism. This mechanism works as follows: a user transfers coins (from the main chain) to a special output in the main chain. The transacted coins are locked until it can be unlocked by the sidechain using a proof of possession. The coins can be unlocked after the confirmation period and finally transferred to the sidechain. The purpose of the confirmation period is to prevent denial of service attacks. Upon transfer, the coins cannot be spent until the contest period, which intends to prevent double spending, expires. Given that sidechains are not fully decentralized like typical Blockchains, they are more scalable and are able to provide confidentiality. Therefore, they present an ideal environment for businesses who want to establish a trustful platform for collaboration (e.g. within the supply chain) [3].

3.1.5 Tangle

Tangle is the ledger technology of IOTA[3], a DLT platform that is focused on the integration of micro-transactions in the Internet of Things (IoT). Tangle is based on *Directed Acyclic Graphs* (DAG), where vertices/nodes represent *sites* (i.e. transactions) and edges the *transaction approvals*. Unlike Blockchain, this technology does not require expensive consensus mechanisms and thus does not charge transaction fees. Whenever a member issues a new transaction (i.e. site), he/she has to select two existing sites and validate them. The validation is done via a less complex POW algorithm called *hashcard* and works well with IoT devices that have low computing power. This ecosystem is more scalable, considering that transactions can be validated in parallel. It is also noteworthy that tangles underlying key-generation algorithm is quantum resistant, thus also a long-term solution [2].

3.1.6 Hashgraph

Similarly, hashgraph also uses DAG as the underlying ledger technology. While hashgraph is public and accessible to everyone, its is only partially decentralized. The reason for this is that only selected entities (e.g. institutions, businesses) operate the platform and participate on decisions regarding protocol implementations and fees. Transactions are propagated to the network using the *gossip-about-gossip* protocol. In this protocol members either create a new event (i.e. transaction) or randomly pick another member and 'gossip' about the creation of a new event.

[3]https://www.iota.org/

These events contain the hashes of the previous event and another user's event. The incorporation of both hashes enables the tracking of transaction history, which is also used to build the DAG of events. Tamper resistance is ensured with the use of digital signatures. The limited number of validator members and the underlying ledger technology (DAG) make hashgraph faster and more scalable [2].

3.1.7 Comparison of DLTs

Akhtar defines a set of comparison criteria in his paper, in order to analyze the differences between the presented DLT implementations. Table 3.1 juxtaposes the evaluation each DLT.

Criteria	**Blockchain**	**Sidechain**	**Tangle**	**Hashgraph**
Architecture	Linked list blocks	Multiple linked lists	DAG	DAG
Transaction	Grouped in block	Grouped in block	Separate entity	Grouped in event
Consensus	POW (SHA-256)	POW (Ethhash)	POW (hashcard)	Virtual voting
Copyright	Open source	Open source	Open source	Patented
Latency	A few minutes	A few minutes	A few seconds	less than 10 seconds
BFT	No	No	No	Yes
Privacy	Low	High	Low	Low
Fee	Yes	Yes	No	No
Throughput	5-20 tps	Limited by main chain	500-800 tps	100,000 tps
Security	High	High	High	High
Fairness	No	No	Not yet	Yes
Cost	High	High	Low	Low
Maturity	Many implementations	Experimental	Experimental	Experimental
Setting	Public	Private & public	Private	Private
Competition	Yes	Yes	No	No
Scalable	Limited	Yes	Yes	Yes
Platform	Ethereum, Bitcoin, ...	Monax[4]	IOTA	Hedera[5]

Table 3.1: Comparison Based on Evaluation Criteria [2]

Architecture represents the underlying datastructure used for storing and arranging transactions; *Copyright* represents the rights of use; *Latency* represents the time needed to validate a transaction; *BFT* stands for *Byzantine Fault Tolerant* and describes systems that tolerate failures belonging to the Byzantine Generals

Problems as explained by Abraham et al. in *Revisiting Fast Practical Byzantine Fault Tolerance*; *Throughput* represents the rate at which transactions are processed per second (tps); *Fairness* represents fairness in the processing order of transactions; *Cost* describes the cost needed for participating in the system and *Setting* describes whether the DLT is public or private [2].

3.1.8 Choice of DLT

Table 3.1 shows the limited capabilities of different distributed ledger technologies. This section deals with the reasoning of the DLT choice by weighing the advantages & disadvantages of the above mentioned technologies. Criteria such as *Throughput, Setting, Scalability, Latency, Maturity, Privacy and Fee* play a significant role in determining the ideal DLT for our use case scenario. Moreover, the prospect of smart contracts is a decisive factor for the integration of business processes with DLT. While IOTA and Monax have limited support for smart contract development, Ethereum and Hedera provide a wide range of tools and documentation for implementing and even automating business processes in form of smart contracts. From a feasibility standpoint, Monax and IOTA are less ideal choices, given that they have few documentation and are intended primarily for different use cases, e.g. IOTAs focus is in integrating crypto-transactions using IoT devices.
Hedera exceptionally outperforms Ethereum in terms of *Throughput, Fee* and *Latency*. However, it resides in a private setting and is therefore not fully autonomous, as it is operated by certain companies & organizations. This becomes an obstacle in establishing full trust among companies who wish to rely on a DLT. Furthermore, there is no guarantee that the permissioned DLT will exist forever, or that it will not be acquired by a company with malicious intentions. Currently, the Ethereum Foundation is in the process of migrating to a newer version, which will use the *Proof-of-Stake* consensus mechanism along with other changes. The new version[6] is expected to enhance the performance in terms of *Latency, Cost* and *Throughput*. With these changes in mind, transaction fees and delays become problems of the past and make Ethereum therefore an ideal system for the use of smart contracts.

3.2 Ethereum Name Service

The Ethereum Name Service[7] (ENS) is a resource naming system based on the Ethereum Blockchain. It is the Blockchain equivalent of a DNS, where machine-readable identifiers such as Ethereum addresses are mapped to human-readable names. The underlying architecture of ENS differs from DNS, but it nevertheless uses the same dot-separated hierarchical naming system. Thus in ENS, the owner of a domain automatically owns and has control over all subdomains. There are two top-level domains '.eth' and '.test' owned by smart contracts called *registrars*. These registrars determine the rules for allocating new subdomains, which can

[6] https://ethereum.org/en/eth2/vision/
[7] https://docs.ens.domains/

be acquired by anyone complying with these rules. Figure 3.4 demonstrates an example for an `ENS Registry`.

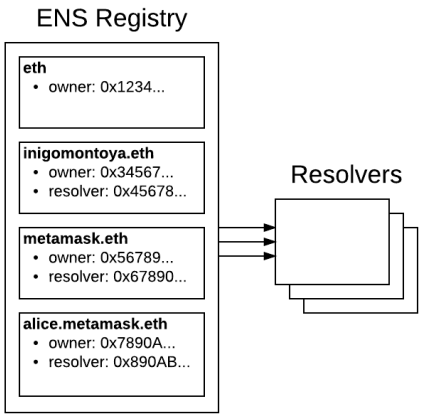

Figure 3.4: ENS Architecture[7]

It is a smart contract containing a list of domains and their subdomains. Each allocated (sub-)domain specifies three important information: the owner of the domain, the resolver and the caching time-to-live. The `Resolver` is a smart contract intended for mapping a human-readable domain name to an Ethereum address, either a smart contract or an account. The `Resolver` smart contract is provided by the owner of the domain, who has control over which address owns which subdomain.

Figure 3.5[7] shows an example usage of an ENS domain. The `User Code`, e.g. front-end, queries the `ENS Registry` for the domain 'foo.eth' and receives the smart contract address of the `Resolver`. Subsequently, the user code queries the *owner* of the domain and accordingly receives an Ethereum address.

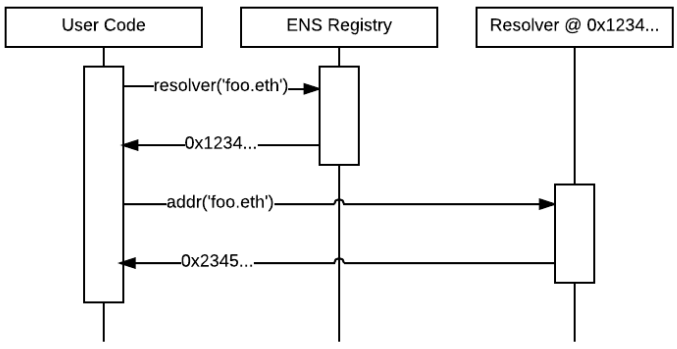

Figure 3.5: Resolving an ENS Domain[7]

3.3 Decentralized Identifier

One of the core concepts of a Distributed Ledger Technology is the ability for users to join the ecosystem anonymously. Considering that DLTs, like Blockchain, are built on cryptographic principles, users who join the ecosystem interact with it using cryptographic keys. However, verifying user identities behind random cryptographic keys is significantly challenging. Therefore a standardized approach for resolving user identities behind randomized identifiers, like public keys, is imperative.

A decentralized identifier is a standardized protocol specified by the W3C foundation intended for the authentication of digital identities without the involvement of 3rd parties. The specification states that "A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID. [...] DIDs are URIs that associate a DID subject with a DID document allowing trustable interactions associated with that subject" [20]. A DID document contains information about cryptographic keys in use, their metadata (e.g. algorithm) and service endpoints which application can interact with. The ownership of a DID can inherently be authenticated with the cryptographic keys defined in the DID document. The uses of these documents are not solely limited to authentication schemes and will be addressed in chapter 5.5. URIs (Uniform Resource Identifiers) serve as unique identifiers for locating specific information associated with the identifier. Figure 3.6 presents the structure of a DID URI consisting of three parts. A DID document residing in the Ethereum Blockchain would look as follows `did:eth:0x5c563c922849cd8172b1a2461eedf6389bcb056a`. The last part represents the Ethereum address (public key) of the DID owner.

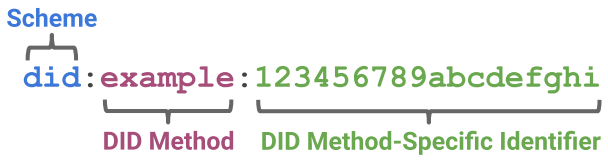

Figure 3.6: DID URI Scheme [20]

Listing 3.1 shows an example document that corresponds to a DID-URI. The `@context` refers to resources defining the schema of a DID document and `id` is the unique identifier of this document. All public keys used for authentication are listed in `authentication`, where the `type, controller` and `key` is also specified. The `service` defines endpoints by `type` and is intended for providing data from external resources about the owner of this DID document. Consider a scenario similar to *Open Authorization* (OAuth), where a user uses their DID-URI to login to a website. The resolved DID document itself does not provide any personal information, but the system can use the service endpoint to query information about the user.

Listing 3.1: DID Document Example [20]

```
{
  @context: [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  id: "did:example:123456789abcdefghi",
  authentication: [{
    id: "did:example:123456789abcdefghi#keys-1",
    type: "Ed25519VerificationKey2020",
    controller: "did:example:123456789abcdefghi",
    publicKeyMultibase:
    "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }],
  service: [{
    id:"did:example:123#linked-domain",
    type: "LinkedDomains",
    serviceEndpoint: "https://bar.example.com"
  }]
}
```

3.4 Verifiable Credential

Conventionally, a physical credential discloses various personal information such as the identity of the subject (e.g. photo, name), the authority that issued the credential (e.g. city government), the type of credential (e.g. passport, driving license), attributes asserted by the issuing authority (e.g. nationality), evidence on how the credential was derived and constraints of the credential (e.g. expiration date, terms of use). All of these properties of a typical credential can be represented in digital form with additional measures, such as digital signatures, for making them more tamper-resistant and verifiable [25].
The structure and content of a credential is shown in figure 3.7. The metadata is comprised of information about the issuer, the expiry date, a representative image, public key etc. The *Claim(s)* are assertions made by the issuer regarding the subject, e.g. a proof of enrolment at a university. The credential can be verified using the *Proof(s)*, which are typically digitally signed by the issuer. The structure of a verifible credential in JSON format is shown in listing A1.

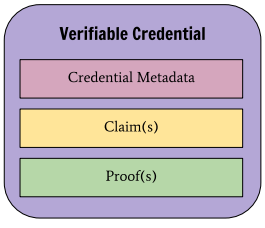

Figure 3.7: Credential [25]

The relationship of the roles involved in the use of Verifiable Credentials is depicted in figure 3.8. The *Holder* is an entity, who possesses multiple Verifiable Credentials and generates verifiable presentations. Example holders are students,

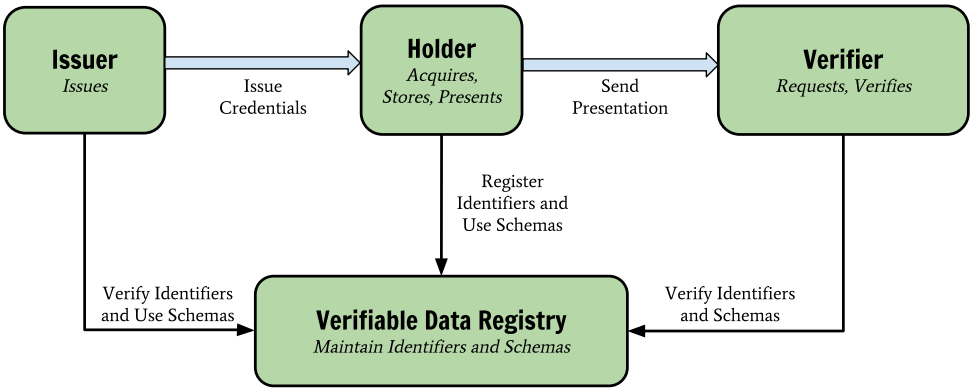

Figure 3.8: Ecosystem Overview [25]

employees or customers. Generally, *subject*s of the credentials are holders. An *Issuer* is an entity, who creates new Verifiable Credentials for claims asserted about subjects. Such an entity is typically a corporation, organization, government, trade association or individual. The *Verifier* is an entity, who is interested in checking the authenticity of claims by digitally verifying their Verifiable Credentials. A *Verifiable Data Registry* is generally a decentralized system/storage, which intends to mediate the creation of Verifiable Credentials and stores information such as identifiers, schemata, keys of issuers or revocations.

Figure 3.9 shows an information graph of a Verifiable Credential issued for university alumni. The color scheme reflects the structure of a credential as presented in figure 3.7. The components in pink represent the properties of `Credential 123`. The subject of the credential is `Pat`, asserted to be an alumni of the university hence both are highlighted in yellow. The proof, highlighted in green, not only consists of a digital signature, but also of its properties, such as the signature algorithm (type) that was used to sign the data, the nonce, the signer (creator), etc. Proofs can appear in two forms: *external* or *embedded* form. The former is a proof wrapped into a *JSON Web Token* (JWT) and the latter is a proof JSON object within the Verifiable Credential (see Appendix A1). Generally, Verifiable Credentials are used jointly with Distributed Ledger Technologies. Entities, such as issuer or subject, can be referenced using DID-URIs, as presented in the previous section. A library for Verifiable Credentials using DID and JWT is available for the Ethereum Blockchain[8].

[8]https://github.com/decentralized-identity/did-jwt-vc

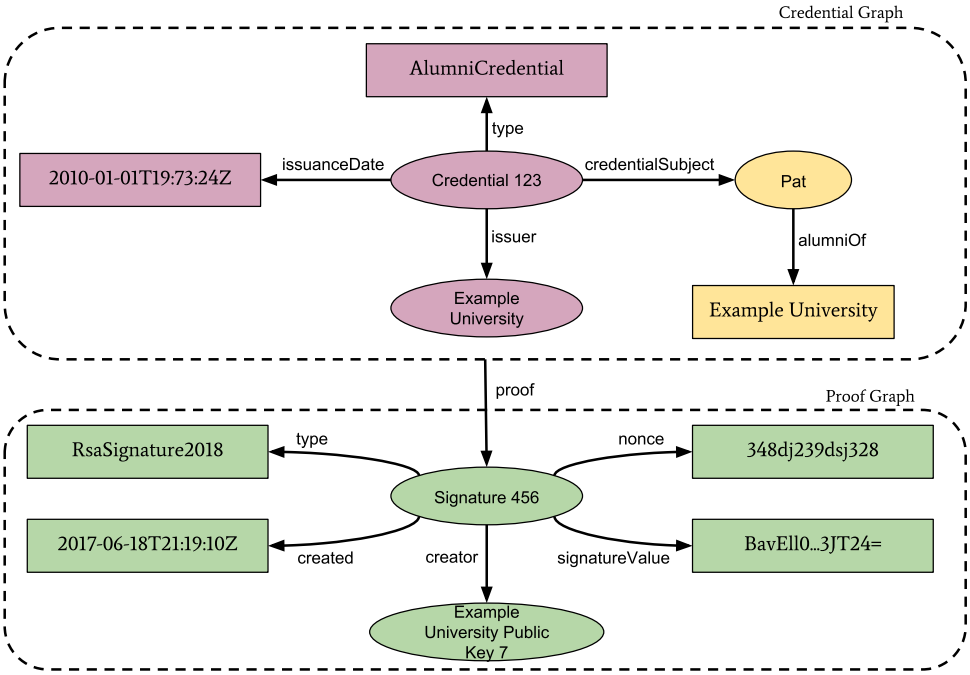

Figure 3.9: Credential Graph for the Use Case of Alumni [25]

3.5 JSON Web Token

According to RFC 7519 a "JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted" [12]. The thesis focuses in signed tokens, as the primary use case of JWTs is the authentication & authorization of users and servers. The structure of a JWT typically has the format

`xxxx.yyyy.zzzz`

The first part `x` represents the header, `y` the payload and `z` the signature. A header contains information about the signature algorithm and the resulting type:

Listing 3.2: JWT Header [12]

```
{
  alg: "HS256",
  typ: "JWT",
}
```

The payload can contain any kinds of claims about the subject. An example is shown in the listing below. The `sub` property defines a unique identifier of the subject, which needs to be recognized by the audience (i.e. verifier). The payload claims that this user, `John Doe`, is an administrator.

Listing 3.3: JWT Payload [12]

```
{
  sub: "1234567890",
  name: "John␣Doe",
  admin: true
}
```

The signature is generated using the algorithm specified in the header, in this case the `Base64-URL-encoder`, and is used to encode the header & payload. The following listing shows the pseudo-code for encoding both JSON objects into a JWT. The last argument in the function, `secret`, expects the private key. The result is the concatenation of the Base64-URl-encoded header, payload and signature, as shown respectively in lines 3, 4 and 6.

Listing 3.4: JWT Encoding Example

```
1 HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(
     payload), secret)
2
3 >eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
4 .eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly
5  9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
6 .dBjftJeZ4CVP-mB92K27uhbUJU1p1r\_wW1gFWFOEjXk
```

Figure 3.10 illustrates a typical usage example of JWT between a user and a web application. The user obtains the token by first authenticating itself to the server. The `login` function represents a `HTTP GET` request containing the users username and password. The server responds with a new session token, which is then used in all consecutive requests to the server. This is accomplished by using the `Authorization` HTTP header with parameter `Bearer JWT`, where `JWT` is a variable.

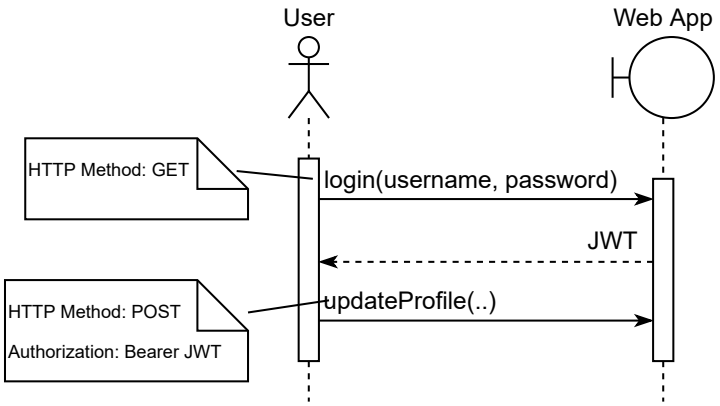

Figure 3.10: JWT Usage Example

3.6 Summary

Distributed Ledger Technology is a system for distributed record-keeping of data, which have been agreed upon using a multi-party consensus mechanism. New records remain in the system forever and cannot be changed. Smart contracts are programs, which are defined and agreed upon by all involved parties. Its advantage is the establishment of trust between parties without the involvement of a 3rd party. Blockchain technology, a concrete realization of DLT, uses merkle trees for cryptographically verifying existing records. Other types of presented DLTs are: Hashgraph, Sidechain and Tangle. A decision for Blockchain (Ethereum) as the choice of DLT was made based on a comparison of important criteria. Useful standardizations and methodologies such as Decentralized Identifier, Verifiable Credential and JSON Web Token, were also presented.

The subsequent chapter reviews related work in the domain of maritime transportation and off-chain approaches.

4. Related Work

The following sections discuss current advances of Blockchain-based solutions in the field of waterway transportation and present approaches of off-chain solutions and authentication in other domains.

4.1 Waterway Transportation

The transport and logistics branches are considered as one of the least digitally developed industries. In recent years however more research has been invested in Blockchain-based solutions, some of which have been already adapted by startups and major companies. The book by Stahlbock et al. presents potential applications of use cases in maritime logistics and introduces some of the projects that have been developed. The authors argue that only actors who are involved in the supply chain should be part of the Blockchain and therefore propose the use of a permissioned Blockchain. The main focus lies in the digitization of the paper-based bill of lading (B/L), which has to comply to certain rules such as the Hague, Hague-Visby and Hamburg rules. These are intended for making sure that an electronic bill of lading (e-B/L) is legally binding.
In figure 4.1 the authors demonstrate a use case, in which the arrival of shipping containers is handled using smart contracts. Actors, who work at the port can be automatically tasked with unloading the container while other actors in the supply chain can be notified about its current state. This will prevent payments from being delayed and save additional costs. On the other hand, the mismanagement of the agreed upon temperature and moisture can be automatically fined, as these values are monitored electronically by the smart contract [26].

The issue of provenance tracking of goods is also addressed with example use cases involving the shipment of fruits and vegetables. Not only are the health aspects of the goods of relevance, but also whether their branding, e.g. "bio" or "FairTrade", is legitimate. An Antwerp-based startup called T-Mining has developed a solution that focuses particularly on provenance tracking [24]. Their pilot use case centers around shipment of fruits from New Zealand to Belgium. The goal is to employ digital and Blockchain-based health certificates of plants. The scenario is represented as follows: before the cargo is exported from New Zealand,

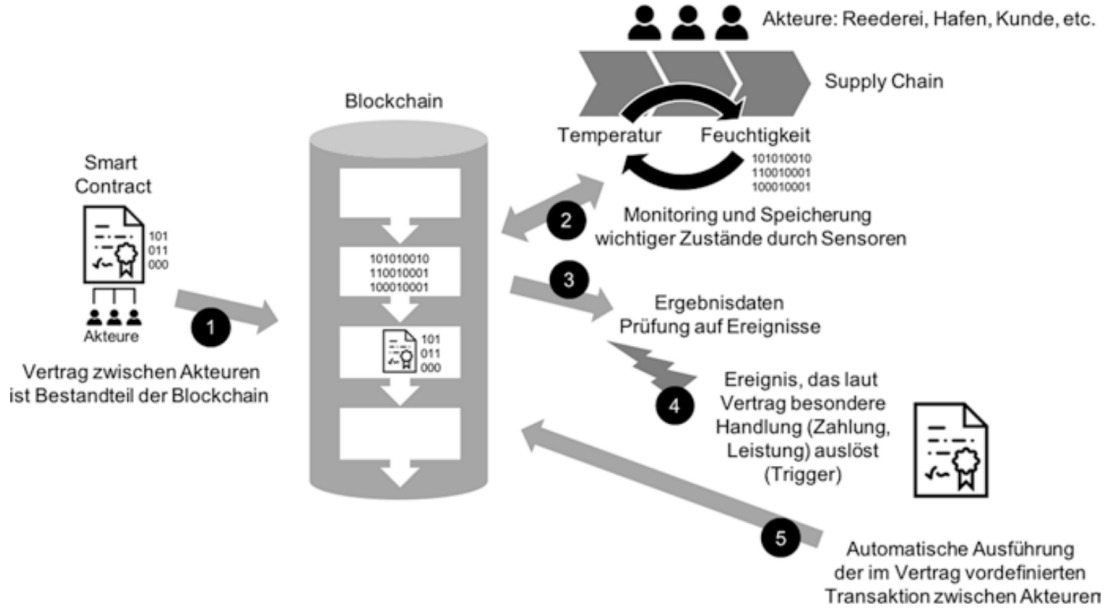

Figure 4.1: Example for Smart Contract with the Monitoring of Temperature and Moisture of Containers [26]

its certificate is verified and sent to the Belgian authorities for validation and if everything goes well the cargo can be released for import.
Stahlbock et al. rationalize that the implementation of the proposed solution is not entirely realistic. Seeing that the e-B/L only conforms to certain regulations, companies located in countries that do not enforce these rules cannot be held liable for any damage or disagreement, thus the e-B/L would be useless. Furthermore, there is a distrust towards a permissioned Blockchain, as it needs to be operated by a company or central authority. Privacy of data is also of concern since a permissioned Blockchain allows every member access to it and therefore needs to be encrypted, thus making its use more inconvenient. In general, the use of Blockchain technology comes with many limitations, such as the irreversibility of faulty smart contracts and scalability. The feasibility of a Blockchain-based solution is also questioned on the grounds that the validation of blocks would not only require enormous computing power, but also require every cooperating partner to have enough internet bandwidth to transfer data in real-time. The average bandwidth in african countries is around 5 mbit per second, which is the reason why current Blockchain-based solutions are only deployed in countries that have more modern infrastructures (e.g. IBM & Maersk [27] deployed in Rotterdam and Newark). The authors of the article expect approaches in the near future to be hybrid solutions, i.e. Blockchain with an external data-storage, while purely Blockchain-based solutions could be possible in 10 to 15 years [26].

Goudz and Ahmad analyze potential use cases for a Blockchain-based solution and investigate current problems in maritime transport. The authors claim that 15% of the costs generated in the maritime industry originates from the documen-

tation of papers. The majority of the cost is caused by delays in the transport chain since documents need to be mailed to various actors in the supply chain. Seeing that only original documents are legally required, there can only be one document owner at a time. And throughout this process, there is a chance of documents getting lost or even forged. When it comes to evaluating the quality and legitimacy of transported goods provenance tracking takes an important role, especially in food transport. It is critical to monitor the temperature of containers and the quality of foods and in case the food gets spoiled on the way the receiver should be notified in time. Furthermore, the monitoring of temperature helps evaluate whether the quality of food may have degraded during the shipping process [8].
Another issue the authors address is the inability to locate the whereabouts of containers and their contents. Moreover, 20% of international shipments and 40% of national shipments comprise of empty containers. Each shipping company tends to use their containers only, thus resulting in empty containers being returned due to lower amount of goods being transported on the way back. The cost of transporting empty containers is as much as transporting full containers, which is why the authors suggest that containers be shared with other shipping companies located on the same port. While many ships return back with empty containers, some other ships are in need of containers and end up not transporting goods that are waiting to be transported. To highlight this problem: costs are not only generated due to empty containers being transported, but also generated due to other ships not being able to transport all of their goods. To expand on the idea of container sharing, the authors propose the application of container asset management, wherein unused containers can be rented out or resold to other companies. This approach however requires the life-cycle of the containers in question to be tracked [8].
Another concern that is discussed is the 'no-show' of containers resulting in the so-called 'rolling'. This means that exporters do not show up on reserved spots and the ports end up losing money since only vacated spots are being paid. The authors argue that 10% of the time these shipping companies experience a loss in profit. In order to circumvent this problems shipping companies overbook the reservations, which results in exporters having to wait until a spot is free. This current system costs the industry around 23 million dollars every year [8].

A startup based in Slovenia called CargoX has been developing Blockchain-based solutions for the maritime industry. The bluepaper "Reshaping the Future of Global Trade with World's First Blockchain-based Bill of Lading - Business Overview and Technology Bluepaper" introduces the workings of CargoX and how it helps solve problems mentioned in the aforementioned paragraphs [21]. CargoX's solution centers around the digitization of the B/L and the processes it is involved in. While it is built on top of the Ethereum Blockchain it nevertheless lives in its own ecosystem. The primary goal is to integrate the lawful ownership of digital documents on a platform that enables the exchange of B/L. The legitimacy of documents is ensured with the help of cryptographic functions, wherein the carrier, for example, signs a document with a hash value. This hash value

is persisted in the Blockchain to make sure that it cannot be modified while the actual data is persisted in a decentralized database, which makes the entire ecosystem more scalable. The hashes on the Blockchain are then integrated into smart contracts, which are responsible for verifying signatures. With the use of decentralized databases data cannot get lost and is more secure against outages. While a B/L cannot be modified, users can nonetheless attach additions to documents, and its ownership can be proven with the help of Ethereum addresses, which are asymmetric cryptographic encryption keys. The paper demonstrates the following scenario for authenticating ownership of documents: an importer wants to prove the carrier that he is the legitimate owner of the B/L. The carrier generates a hash value and sends it to the importer, who has to sign it with the private key of his Ethereum address. The importer now releases his signature and public key, which is verified by the carrier and the smart contract. This entire procedure is automated by the ecosystem. The transfer of ownership is performed in a similar manner: the smart contract checks the existence of the receivers address and the authenticity of the sender. In order to administrate B/Ls one has to use CXO tokens, which assigns members rights to execute certain actions on the platform. Some of these actions are: getting access to the system, administering transaction costs, monitoring completion of smart contracts and making payments. Figure 4.2 illustrates how various types of users can interact with the ecosystem. CargoX developed a decentralized application (DApp), which enables users to access the Blockchain-based platform and perform certain tasks. A unique feature that the

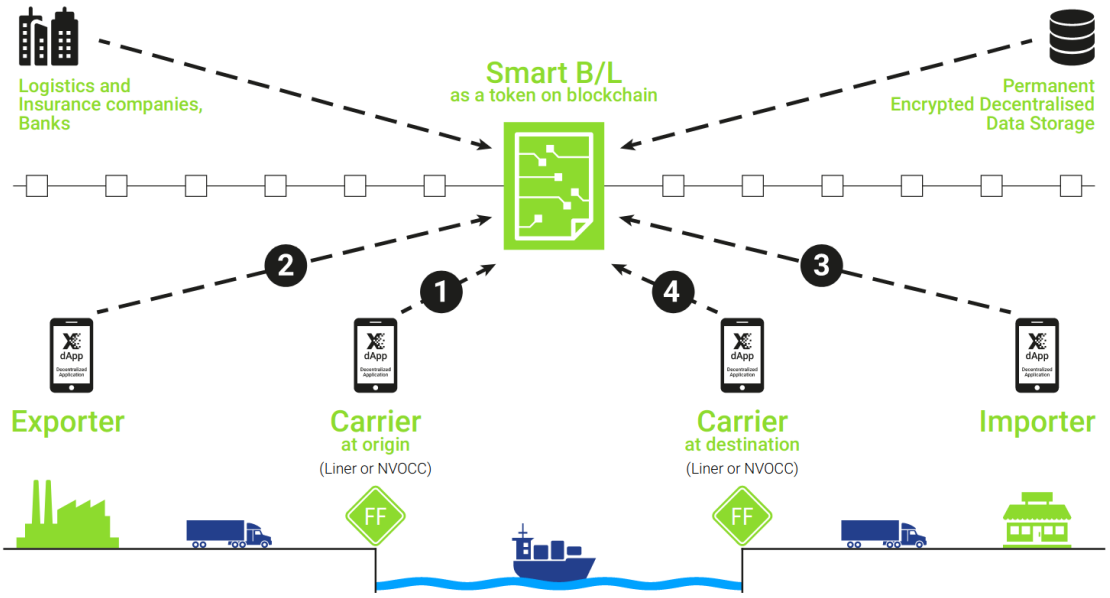

Figure 4.2: Overview of CargoX [21]

platform offers is the use of CXO tokens as replacement of documents like B/L. Smart B/Ls can be split into these tokens and can either be forwarded to or traded with other users of the supply chain. To paint a better picture of the process: the carrier can use the DApp to create a smart B/L, sign it and convert it into a token.

The token will be forwarded to the sender, who can use the DApp to transfer the ownership of the B/L to the receiver [21].

Table 4.1 shows a comparison between the papers presented in this section. In general, both papers address the problems mentioned in section 1.1 and propose ideas to digitize and improve current processes. Stahlbock et al. also discuss the limitations of Blockchain as a technology and take a critical stance towards realizing a purely Blockchain-based solution. Neither paper criticizes the privacy concerns associated with the use of permissioned & unpermissioned Blockchains.

Criteria	**[26]**	**[8]**
1. Addresses limitations of Blockchain technology		
1.1 Scalability	✓	✓
1.2 Confidentiality	✗	✓
2. Discusses types of Blockchain implementations		
2.1 Public / permissionless	✓	✓
2.2 Private / permissioned	✓	✓
2.3 Consortium	✗	✓
3. Addresses problems of paper-based processes		
3.1 Cost	✓	✓
3.2 Susceptibility of errors	✓	✓
3.3 Risk of fraud	✓	✓
3.4 Delays	✓	✓
4. Introduces own ideas for improving processes		
4.1 Incorporating IoT	✓	✓
4.2 Digitization of documents using smart contracts (e.g. B/L)	✓	✓
4.3 Automation of tasks	✓	✓
4.4 Standardization of legal documents	✓	✗
4.5 Provenance tracking	✓	✓
5. Introduces other potential use cases of Blockchain in maritime industry		
5.1 Container Management	✓	✓
5.2 Cyber Security	✓	✗
5.3 Rating of business partners (similar to DB Schenkers *VeChainThor* [29])	✓	✗
6. Presents existing solutions		
6.1 T-Mining	✓	✗
6.2 TradeLens	✓	✓
6.3 CargoX	✗	✓
7. Addresses the conflicts for adopting Blockchain		
7.1 Rivalry between suppliers	✓	✗
7.2 Infrastructure requirements	✓	✗
7.3 Cost of creating and operating Blockchain solution	✓	✗
7.4 Skepticism towards new technologies	✓	✗
7.5 Network requirements	✓	✗
7.6 Potential exposure of sensitive data & business secrets	✗	✓

Table 4.1: Comparison of Presented Research Papers

4.2 Off-Chain Approaches

The paper "Distributed Off-Chain Storage of Patient Diagnostic Reports in Healthcare System Using IPFS and Blockchain" by Kumar et al. presents an off-chain Blockchain implementation approach in the use case of healthcare systems. They suggest that when the need arises, patients medical information should be accessible by physicians in different hospitals. It is crucial however, that information is not only kept private, but is also immutable. Blockchains characteristics make it an ideal solution where privacy, immutability, integrity and consistency is ensured. Taking into account the volume of medical data that needs to be persisted, the use of Blockchain as data-storage is not sustainable. Therefore, the authors offer the use of a decentralized peer-to-peer (P2P) database. One such database is IPFS [4], Interplanetary File System, which stores the hash of each data in a distributed hash table (DHT) [31] and uses a version-control history to remove duplicate files. In addition, it stores highly requested data locally, so it can be accessed faster. It offers useful functionalities like high throughput, security through hashing and concurrency. At this point, the Blockchain is only used to store the hash generated by IPFS, which later can be used to look up the corresponding data. In order to limit access to the ecosystem, a consortium Blockchain will be employed. It is a combination of a public and private Blockchain, where the Blockchain is still decentralized but operated by participating hospitals. With the Proof-of-Identity based access regulation the privacy of data can be preserved. The authors propose a framework consisting of three modules: data upload, mining process and data storage. A web user interface (DApp), which is part of the framework, is used to access the ecosystem. Each health care provider is required to register in the consortium Blockchain using an app and obtain a Proof-of-Identity. Through the app the healthcare provider can upload patient diagnostic, which is processed in form of a Blockchain transaction and therefore mined using the Proof-of-Work approach. The miner verifies the transaction with its peers and persists the data in the IPFS, while also storing its newly generated hash in the Blockchain [13].
Hepp et al. also present numerous data storage systems that can be utilized alongside Blockchain. Most implementations of DHTs struggle with the exchange of data between users of the same network. Seeing that the data is owned by one particular user and therefore encrypted, it is useless to other users in its encrypted form. Therefore if a user wants to share their personal data, they would have to first download it, encrypt it and then forward it. However BlockDS [7], another variant of DHT, combats this problem by integrating access rights. The owner of the data has primary access rights and is able to allow access to other users as well. The encrypted data is assigned a set of static keywords for the purpose of searching, since encrypted data is not readable. There is a potential risk of a user, who was granted access by the data owner, copying and distributed the data to the public [10]. There are also distributed databases like EthDrive and BigchainDB [14], that integrate a querying language while also using DHTs and P2P nodes.

Although the problem of Blockchain scalability can be combatted by the use

of DHT implementations, the exchange of data between smart contracts and data storages is yet to be discussed. The paper "Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World" presents the concept of oracles, which represent an interface between smart contracts and external data storages. The implementation of oracles is regarded as crucial, since the decentralized characteristics of a Blockchain and external data storages is of importance. Therefore, oracles should be resistant to failures and should preserve privacy and transparency. This would mean that they should also be decentralized. In table 4.2 the definition of oracles is further abstracted into different categories consisting of direction (i.e. inbound, outbound) and initiator (i.e. pull, push).

	Pull	**Push**
Inbound	The *on-chain* component *requests* the off-chain state from an *off-chain* component	The *off-chain* component *sends* the off-chain state to the *on-chain* component
Outbound	The *off-chain* component *retrieves* the on-chain state from an *on-chain* component	The *on-chain* component *sends* the off-chain state to an *off-chain* component

Table 4.2: Overview of Oracle Types [15]

The authors analyze the communication pattern of each oracle type and present them in form of example use cases. The following figure demonstrates how information is passed between the components within a pull-based inbound oracle.

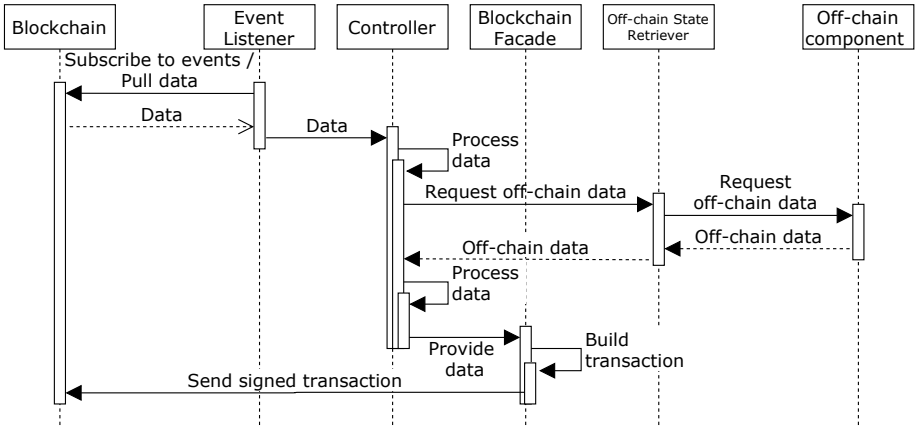

Figure 4.3: Workflow of a Pull-based Inbound Oracle [15]

This type of oracle intends to address the following problem: a smart contract needs to retrieve data that is located outside of the Blockchain, but cannot access it due to Blockchains sandbox-like architecture. Generally, data is being pushed to smart contracts via distributed apps, but smart contracts themselves cannot initiate interaction. The only way for them to interact is by notifying apps whenever an event occurs. The oracle uses this particular feature to strategically interact with

the outside world. Its implementation consists of two types of elements: internal elements that live within the Blockchain and external elements that live outside the Blockchain. The elements that live inside the Blockchain can in practice be represented by a single oracle smart contract, which utilizes the *Event Listener* to listen for events from smart contracts, that need to query data. The *Controller* is responsible for forwarding data to the external element *Off-Chain State Retriever*, which interacts with the *Off-chain component* to retrieve data. Once the *Controller* obtains the data from the *Retriever*, it can use the *Blockchain Facade* to communicate with the smart contract, that triggered the event in the first place. The authors question the stability of the proposed idea in terms of network connection. One of the drawbacks of this approach is the response time being dependent on the underlying Blockchain network thus creating the potential for bottlenecks. Furthermore, network congestion can cause delays or lose connection while retrieving data outside of the Blockchain network [15].

The development of off-chain implementations require the application of an authentication scheme, which is presented by Zhang et al. in the domain of social networks. They propose a self-sovereign mechanism that does not rely on a central authority, but rather primarily on the Blockchain. The prospective framework consists of several parties: (1) social network apps, (2) users, (3) Blockchain and (3) an off-chain storage [33]. Figure 4.4 shows an overview of the proposed architecture.

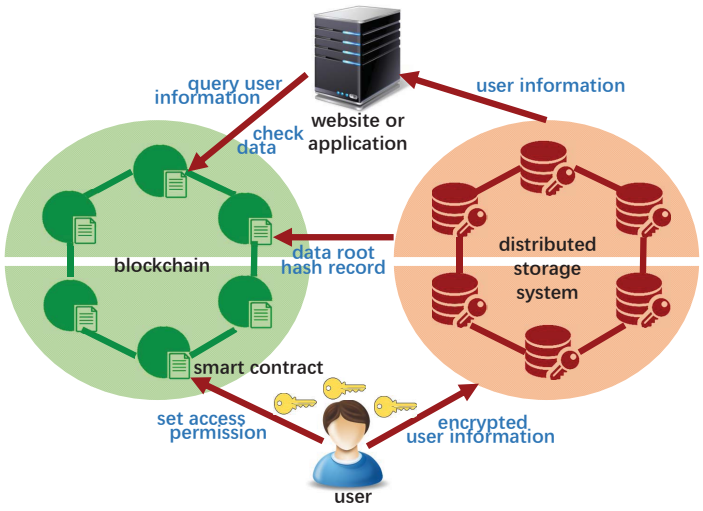

Figure 4.4: Overview of the Distributed Authentication Scheme [33]

The general idea is described as follows: a user encrypts their personal information and saves it into a DHT. The hash, that is generated by the DHT, as well as the users public key is linked to a smart contract, that was created by the user itself. This smart contract is used for authentication by social networks, wherein the smart contract decides which personal information to release. This entire challenge-response protocol is similar to OAuth [9], where the user logs in to the social network using his private key. Figure 4.4 illustrates an overview of all

parties communicating using the proposed authentication scheme.

4.3 Conclusion

The literature review shows that there are existing Blockchain-based solutions for maritime transportation, namely TradeLens, CargoX and T-Mining. These solutions either use permissioned or consortium Blockchains and are targeted at international container shipping. Twenhöven and Petersen conduct a survey about the benefits and impact of Blockchain in logistics. When asked the question *'Which are challenges for Blockchain adoption?'* 56% answered with *Operation and maintenance of system*, 52% answered with *Transparency of Business Secrets* and 47% answered with *Storage of personal Information.* These concerns are the main reasons why many companies have not joined Blockchain-based solutions. Hence, existing platforms (e.g. T-Mining, Tradelens) are struggling to find more cooperating businesses. With this knowledge in mind, one should aim at a solution, which is self-sovereign, accessible to any interested stakeholder, confidential and standardized. This would not only facilitate maintenance but also give the control of confidential data to the stakeholders who own them.
Stahlbock et al. present a realistic use case of smart contracts in waterway transportation. The authors propose the monitoring of IoT data (temperature & moisture) using smart contracts and automation of transactions based on external events. Furthermore, the authors argue that the issue of scalability could be tackled by implementing a hybrid solution, which consists of an on- and off-chain system. Implementing the previously mentioned smart contract example together with a hybrid solution would be perfectly applicable to the aggregation of shipment data and the provenance tracking of grain.
However, the realization of a hybrid solution requires a communication protocol between smart contracts and the 'outside world'. Mühlberger et al. present and discuss multi-purpose oracle patterns for the provisioning of external data. The *pull-based* pattern is of more relevance for this thesis, considering that the majority of data should be stored off-chain and supplied to the smart contract when the need arises. The authors also examine possible bottlenecks and delays due to network congestion and overall slow transaction throughput. This is a factor, which could affect the overall performance of processes and thus needs to be considered for the evaluation of the realized solution.
The approach by Kumar et al. presents a hybrid Blockchain solution for healthcare, where a consortium Blockchain is used by participating hospitals and data is stored off-chain on a distributed DHT-based database (IPFS). Data is by default cryptographically hashed and can only be looked up by its hash value. The integration of DHT into Blockchain is simple and can be achieved by simply storing the hash value. This method has two advantages: (1) a large dataset is only referenced by a single hash and (2) the raw data is immutable and thus cannot be manipulated.
Zhang et al. propose a Blockchain-based authentication scheme for social media applications. Users have full control over the data they share, by defining access permissions in a smart contract. Users store and encrypt their private information

on a public distributed storage system and use its hash to reference to it from the smart contract. This solution resembles the OAuth protocol, with the exception that it is fully self-sovereign and gives users full control over their data. However, this scheme does not guarantee that the data is authentic, meaning that the user could claim to be someone else in order to remain anonymous. This scenario would be tolerable for social media applications, but not for commercial ones.

5. Conceptual Design and Implementation

The impending chapter presents the conceptual design of the system architecture and the implementation of its individual components (ecosystem). The aim is to present a solution which addresses the research questions 2, 3, 4.2 and 5. The final section depicts the data-flow, in order to provide a clear overview of how data is processed by the components of the ecosystem throughout a shipment process.

5.1 System Architecture

The architecture to be presented was adapted from the approach by Osterland and Rose [17]. The objective of this thesis is to integrate trusted data aggregation to an ecosystem designed for the business process presented in figure 2.2. This includes the digitization of artifacts (e.g. charterparty, B/L) and its encoding into smart contracts.
The authors propose a hybrid DLT-based solution consisting of both on-chain and off-chain components. The main idea is to store raw data off-chain in order to handle scalability issues. The following figure shows the architecture proposed by Osterland and Rose.

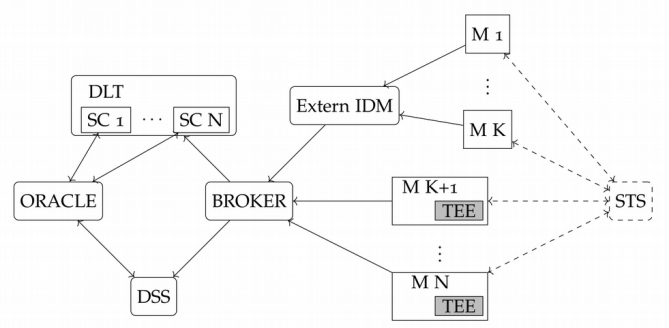

Figure 5.1: First Version of the Architecture [17]

It consists of 5 types of components: the Distributed Ledger Technology (DLT) and its smart contracts (SC), the oracle, the broker, the data-storage (DSS) and

the external Identity Management (IDM). As already discussed in section 4.2, the oracle intends to bridge the autonomous communication between smart sontracts and the 'outside world', i.e. servers or databases. The broker is solely intended for supplying new information to the smart contracts and the respective database. Hence, any data that is provided needs to be checked for authenticity. The external IDM is intended to handle user authentication and verification, which uses the STS component for generating random nonces or timestamps when authenticating users. The intention of the STS is to prevent old requests of being replayed by malicious users and thus adds an additional layer of trust. This approach does not dictate which server environment or database architecture to use, neither does it mandate whether components should be centralized or decentralized. The challenge of this thesis is to contribute to and adapt this architecture, such that

- stakeholders do not expose confidential data,
- stakeholders have control over their own data,
- external stakeholder are able to exchange data with each other,
- any data that was provided by an untrustworthy stakeholder is authenticated and comes from a reliable source.

Figure 5.2 shows the adapted version of the architecture, which includes the involved stakeholders.

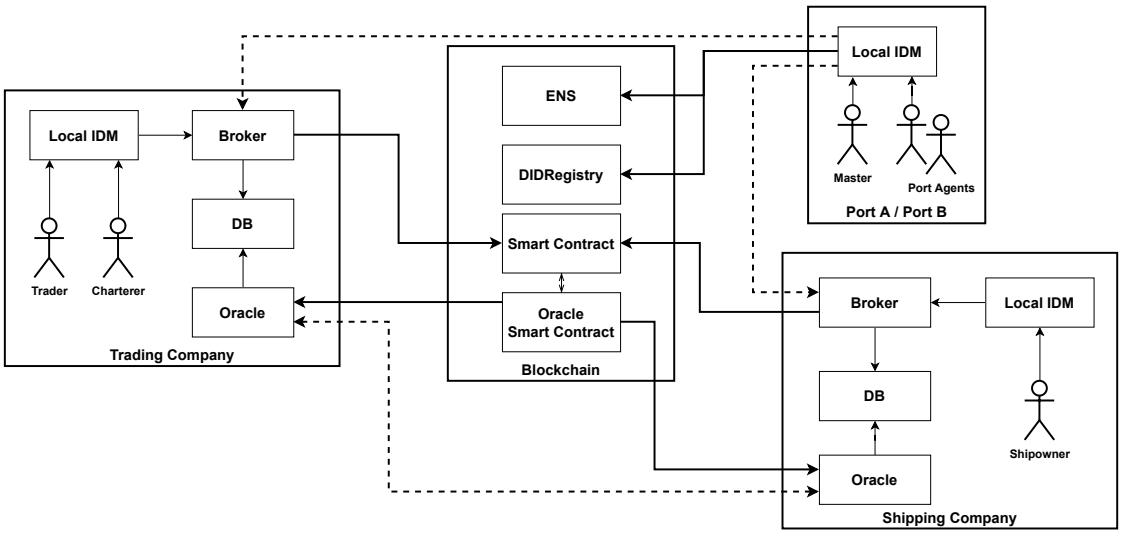

Figure 5.2: Refined Architecture

The overall architecture consists of semi-decentralized oracles and brokers. As can be seen, companies operate within their own environments and thus have control over their own servers and databases. In theory, there could be many trading and shipping companies existing in this ecosystem.
Seeing that users in the supply chain are working at different locations, it is not only important to ensure authentication but also security against cyber-attacks

such as man-in-the-middle (MitM). Ideally, users should be authenticated in local networks, which is why the responsibility of authentication should not fall unto the broker, but instead should be done by external Identity Management (IDM) servers. In our use case, these servers would be located at companies (e.g. supplier, shipowner) and ports. The external IDM is distributed across many local networks and intends to authorize users in a network that is remote to the broker. IDMs, that are not located in the same network as the broker, need to be trusted (or white-listed) by the broker, so that they can authorize access to users.
The trading and shipping company delegate their rights to users and entities using the *DIDRegistry*. Moreover, they register their service endpoints (broker and oracle) so that authorized users can interact with them without having to know about them beforehand. Users can obtain these information by resolving the company's DID-URI, but memorizing the Ethereum address of the company is bothersome and complicated. Therefore, an *Ethereum Name Service* (ENS) contract is used in order to register an alias (i.e. similar to DNS) for addresses, which are simple and human readable. These aliases can be provided to external *IDM*s, which then can locate and resolve further information about the company's service endpoints.
The authentication is performed using a challenge-response protocol, where users prove the ownership of their keys by signing randomly generated nonces. Upon successful authentication, users are issued an expirable session token, with which they can access the broker and make requests to the smart contract. The broker itself is a web-application which uses the Ethereum client in order to interact with the Blockchain.
Information between the trading and shipping company is automatically exchanged using oracles, which listen to events from the oracle smart contract. The only purpose of the oracle is to react to smart contract events, either by responding to the smart contract or by requesting data from another oracle. The broker on the other hand is only intended for adding new data to the smart contract and database.
Consider a scenario, where the charterer deploys and signs a charterparty. The shipowner does not know the contract address yet, and therefore its oracle cannot listen to events tied to a particular charterparty. In order to notify all partners about new contracts, a deployment event is emitted from the oracle containing information about the event parameters, including the deploying account. Using this information, the shipowner can resolve the did document of the deploying account and obtain the service endpoints of the corresponding oracle. Once retrieved, the shipowner's oracle can authenticate itself to the deployers oracle and make a request, e.g. private information about the charterparty. This way, sensitive information is kept off-chain while update events are pushed on-chain.

The proposed architecture in 5.2 gives each company the freedom to decide which underlying architectures and technologies to use: centralized/decentralized servers and underlying frameworks. Nevertheless, it imposes restrictions by enforcing the use of a custom protocol for authentication and oracle-to-oracle communication.
The realization of the complete ecosystem involved the application of various tools

and frameworks. Smart contracts were developed using the solidity programming language and with the help of *Truffle*[1], a framework for compiling and testing contracts. The deployment was performed on a local Ethereum node simulated by the *Ganache*[2] Blockchain. The *web3js*[3] API was incorporated into web apps, in order to allow users to communicate with the Blockchain. The front-end of these web applications have been developed using the ReactJs[4] UI framework. And all servers operated off the chain were implemented using the NodeJs[5] framework with MongoDB[6] as the underlying storage system. The implementation can be found in the Github repository[7]. The following sections describe the implementation details of the smart contracts and off-chain components.

5.2 Smart Contracts

Figure 5.3 depicts the class structure of the smart contracts. Trusted data aggregation only takes place in digitized documents, which have been encoded into smart contracts. These documents are `Charterparty, BillOfLading` and `ProofOfDelivery`. Each charterparty can have multiple shipments and therefore also multiple Bills of Lading. Similarly, each shipment needs to have a Proof of Delivery. The oracle smart contract is used to broadcast events, which is why `Charterparty, BillOfLading` and `ProofOfDelivery` have a reference object. These documents need to be deployed for each new shipment, hence a factory pattern was implemented. The factory itself creates (i.e. deploys) an empty instance of objects, which need to be initialized after creation using the `init()` method. The application of a factory pattern not only facilitates the deployment of contracts, it also provides the possibility to track created objects and, most importantly, it saves gas. Both `Factory` and `DelegatableFactory` utilize the `Clone` library, in order to avoid deploying the same bytecode again and thus reducing gas consumption.
The `Delegatable` contract utilizes `ENS` & `DIDRegistry` and is intended for representing an entity, e.g. company, who can

- delegate their rights to their employees,
- register their oracle and broker service-endpoints,
- register their own ENS-domain,
- create sub-domains for their employees, and
- authorize employees to make payments on behalf of the entity.

[1] https://www.trufflesuite.com/docs/truffle/overview
[2] https://www.trufflesuite.com/docs/ganache/overview
[3] https://web3js.readthedocs.io/
[4] https://reactjs.org/
[5] https://nodejs.org/
[6] https://www.mongodb.com/
[7] https://github.com/Merve40/master-thesis

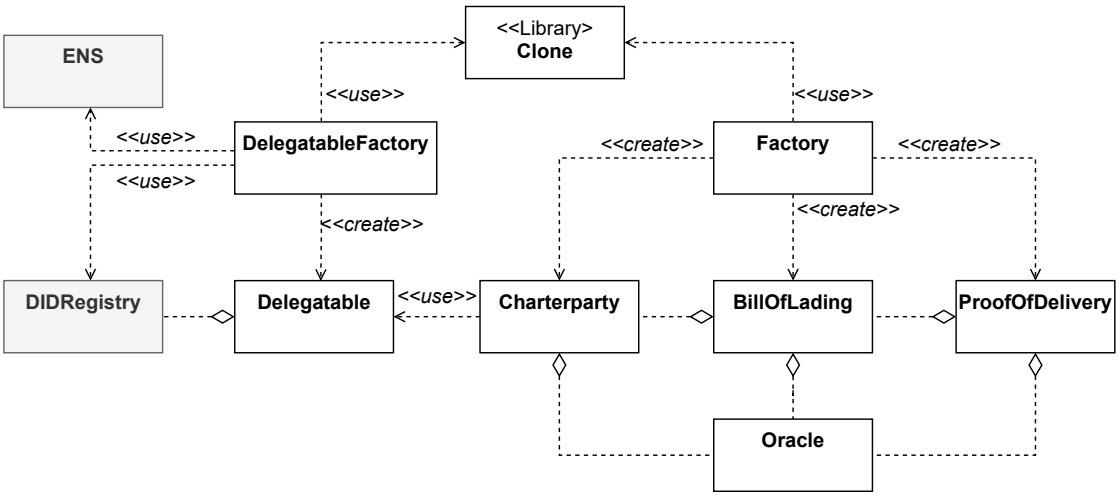

Figure 5.3: Class Structure of Smart Contracts

Using this approach, access to certain contracts and their functionalities can only be restricted to delegates, which adds an additional layer of trust and safety.
In our use case scenario, the trading and shipping company have their own `Delegatable` contracts and appoint delegates, who only use their Ethereum addresses to interact with the Blockchain. Delegates are granted rights according to their roles: the trading company grants rights to *charterer* and *agents* at both ports, while the shipping company grants rights to *shipowner, master* and *agent* at the discharging port.

Figure 5.4 shows the properties of the smart contracts relevant for the data aggregation. `Charterparty` has a reference to the `Delegatable` of the shipowner and charterer, and to both appointed port agents. The contract contains three important information required by `BillOfLading` and `ProofOfDelivery`:

- `fineFee` defines the fine amount which is paid by the shipowner if a condition was not met during the shipment process,
- `laytime` defines the maximum time allowed for the cargo to be shipped,
- `max_moisture_level` determines the maximum allowed percentage of the grains moisture, which is influenced by the environmental conditions and is therefore measured at each port

The `submitSignature()` method allows both parties (charterer & shipowner) to cryptographically sign the contract for agreeing to the terms. `BillOfLading` has a reference to the charterparty and defines the time of loading (`time_loaded`), the weight of the cargo (`cargo_quantity`) and the measured moisture level (`cargo_moisture_level`). Moreover, it defines a `deposit()` function, which when called transfers ether to the contract. It is intended for the shipowner, who needs to deposit the fine amount and the port dues (i.e. payments of the port agents) in advance. The total amount is later transferred to the `ProofOfDelivery` via the

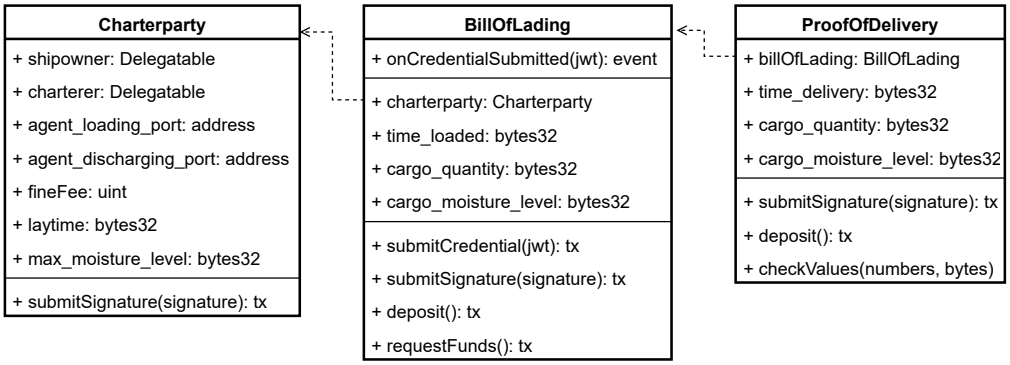

Figure 5.4: Class Properties

`requestFunds()` function. `ProofOfDelivery` also captures the cargo information and the time of delivery (`time_delivery`). Once captured, it transfers the port dues to the agent and queries the oracle for raw data. The raw data is then used for checking whether the values meet the required conditions. In case these conditions are not met, the contract transfers the fine fee to the charterer on its own accord. With this procedure, unmet conditions can be captured in the Blockchain and compensations, such as fines and payments, be automated.

5.3 Identity Management

Access control and authentication is mainly handled by the Identity Management (IDM) entity. It consists of decentralized servers deployed at several different locations, and disconnected from each other. These servers are used in two environments: internal networks and external networks.

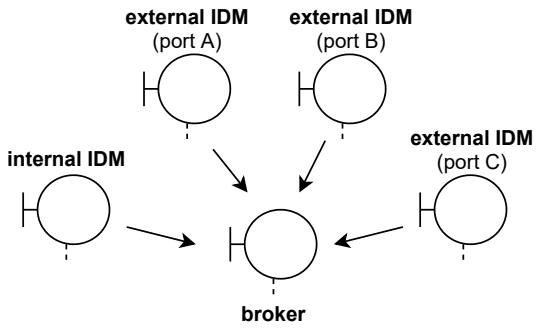

Figure 5.5: Overview of Broker-IDM Connections

Internal servers are located within the same network as the broker and are therefore trusted by the broker by default. In contrast, external servers authenticate users within their own network, that is remote to the broker. Accordingly, external servers need to be trusted by the broker beforehand and frequently authenticated. The external IDM can communicate with the broker using session tokens, which are issued upon successful authentication (i.e. via a challenge-response protocol).

Figure 5.5 shows the overview of IDM servers communicating with a single broker. Taking our use case into account, a user can access the broker from any port that has a local IDM server deployed.
Figure 5.6 demonstrates the authentication scheme of a user via an external IDM server.

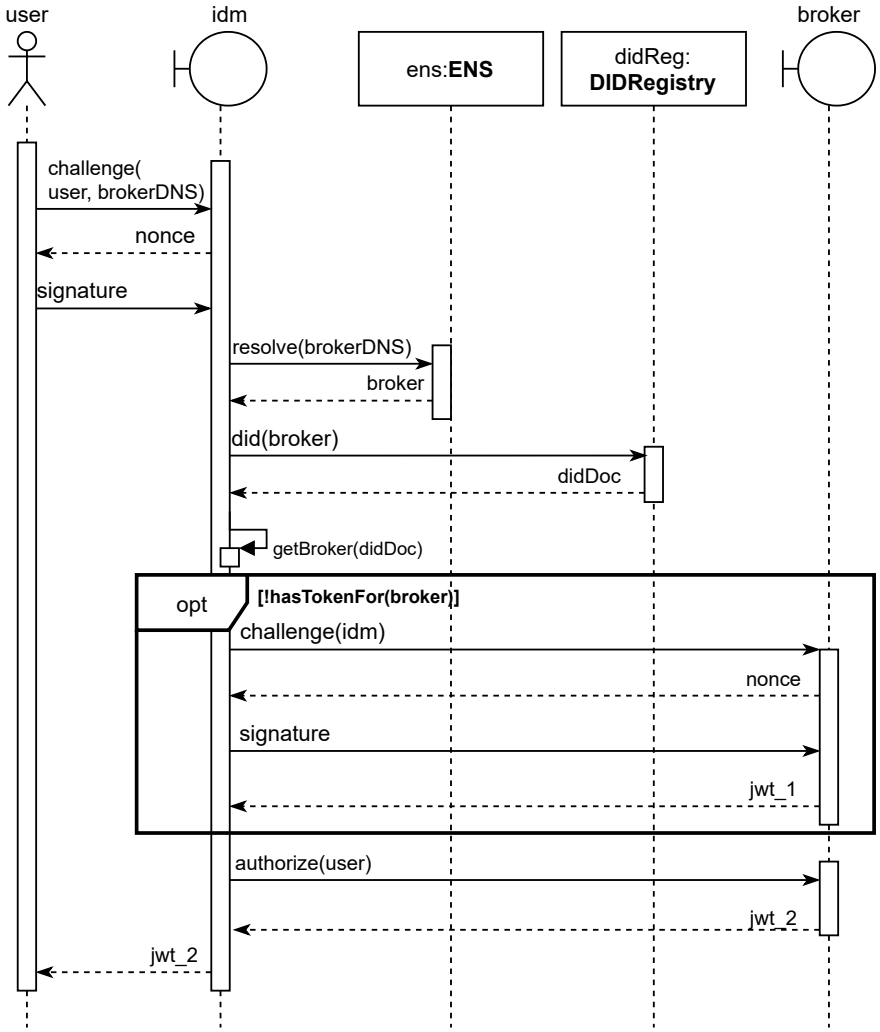

Figure 5.6: Authentication with External IDM

The authentication is initiated with a challenge-response protocol where the user requests a challenge (i.e. nonce) from the IDM server and signs the nonce with their private key. The IDM server recovers the public key from the signature and thus verifies the ownership of the public key. The user provides the `brokerDNS` as an additional parameter, in order to let the IDM know which broker to connect to. Upon successful authentication, the IDM resolves the `brokerDNS` using the `ENS` contract, retrieves the did-document from the `DIDRegistry` contract and fetches the service endpoint of the broker. The IDM checks whether there is a session token between itself and the broker server and if this is not the case it authenticates itself using the challenge-response protocol. Lastly, the IDM requests a session token

from the broker on behalf of the user and forwards the token back to the user.

5.4 Broker

The implementation of the broker component is represented in the class structure in figure 5.7. The core functionalities of a broker are implemented in a separate class `BrokerCore`, which is used by concrete implementations `BrokerCharterer` and `BrokerShipowner`.

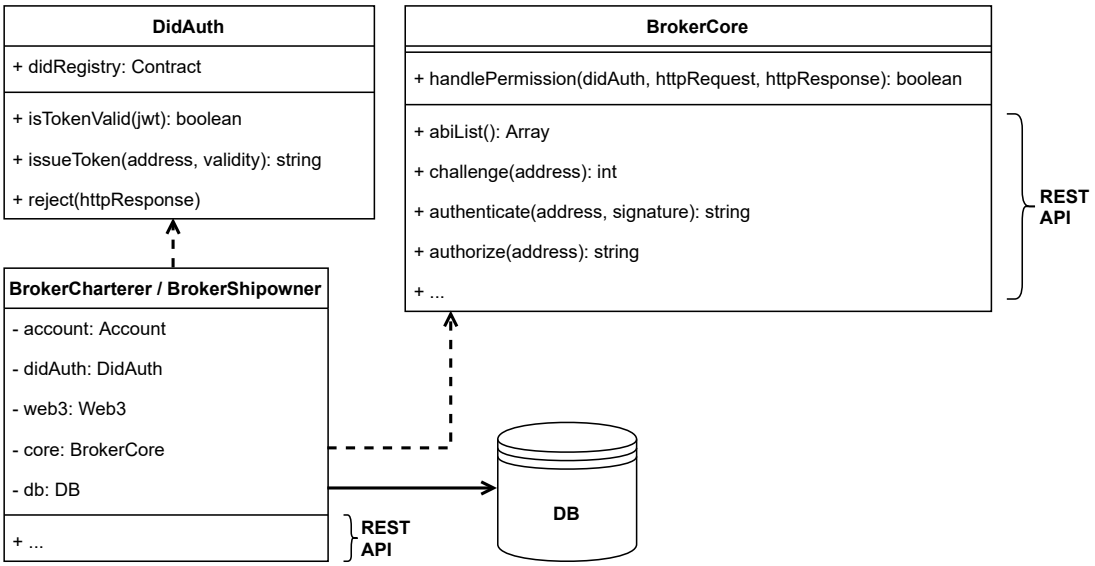

Figure 5.7: Structure of the Broker Implementation

The authentication between an IDM and broker is enabled by REST endpoints `GET /challenge` and `GET /authenticate`, which are illustrated as functions in the class diagram. Similarly, the authorization of the IDM is granted via the `GET /authorize` endpoint. Both `authenticate` and `authorize` produce a new *JSON Web Token*, which permits the user to access the brokers REST API. This session token is supplied in the HTTP header of each HTTP request and is validated with the `handlePermission()` function. Its parameters include an instance of `DidAuth, httpRequest` and `httpResponse`. The second argument is used for fetching the *JWT* token from the HTTP headers, while the third argument is used for rejecting (i.e. returning a server error) in case of a missing or invalid token. The `DidAuth` class provides functions for handling authentication based on DID documents. The `issueToken()` function adds a new delegate and issues an expirable token, which can later be checked for validity with `isTokenValid()`. The `reject()` function returns a server error with a message informing the user of the failed authentication. The concrete broker implementations `BrokerCharterer` & `BrokerShipowner` have a dedicated ethereum account, which is used for issuing and signing tokens. The same account is the owner of the corresponding `Delegatable` contract of the company and is also used for the oracle component. Each broker

implementation has a reference to their own database and implementations of their respectable REST APIs for accessing and inserting data.

5.5 Oracle

The oracle component consists of two parts: the smart contract and the corresponding server. Figure 5.8 shows the class diagram containing fields, public functions and events. The oracle smart contract primarily serves two purposes: one being the notification of new information and the other being the request of raw data. New information is pushed from `Charterparty, BillOfLading` and `ProofOfDelivery`, with each oracle server having the possibility to filter certain events by parameters with an `indexed` attribute. This allows an oracle to discard events intended for other oracles.

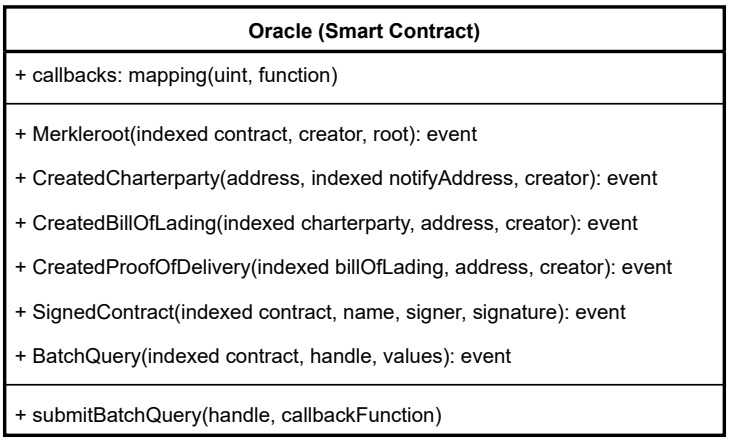

Figure 5.8: Oracle Contract Implementation

Contract instantiation events are emitted in order to notify relevant parties of the contract address, which is used for filtering new events. Consider a scenario where charterer C deploys a `Charterparty` for shipowner S: S would not be able to listen to new events related to the charterparty, for example creating a `BillOfLading`, unless S was informed about the contract address beforehand. Therefore, whenever the `CreatedCharterparty` event is emitted, the `indexed notifyParty` parameter representing the address of S, is also supplied. This method allows for the automation and acceleration of certain information exchanges.
Contracts are considered valid once they were signed by relevant parties. In order to notify all interested parties about the validity of a contract, a `SignedContract` event is emitted. By default, unsigned contracts cannot be used to perform any further actions, such as creating a new `BillOfLading` for a `Charterparty`.

As mentioned before, the second purpose of the oracle is to request certain data off the chain. This scenario will be discussed in the impending section 6.2 *User Interaction*.

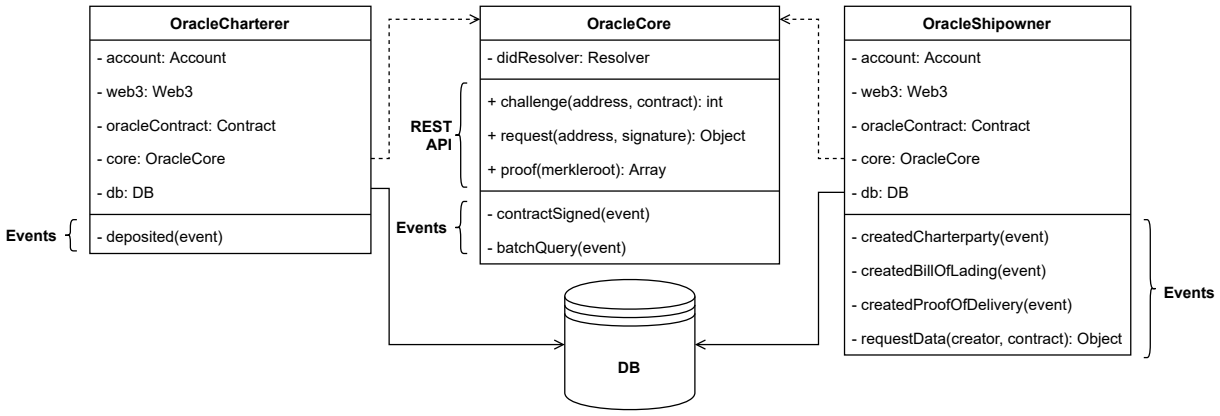

Figure 5.9: Structure of the Oracle Implementation

The server architecture of the oracle is depicted in figure 5.9. Similar to the broker architecture, the oracle utilizes core functionalities from the `OracleCore` class and accesses the database to query data. The concrete implementation of the oracle depends on the companys role. As the charterer is financially responsible for the cargo, the port agents issue and sign documents on behalf of the charterer and therefore use the charterers broker. All information supplied to the broker are saved automatically to the database. Thus, the `OracleCharterer` only listens to `deposit` events. However, the shipowner needs to keep copies of the issued documents and as a result uses the oracle smart contract to listen to deployment events. The smart contracts themselves do not reveal any relevant information, which is why the oracle needs to fetch the corresponding raw data from an off-chain database. As shown in figure 5.8, deployment events contain information about the `creator`, i.e. account/contract address, that emitted the event. This information is used for locating the oracle server, that keeps a copy of the raw data. The `OracleCore` class defines a protocol for authenticating & communicating with other oracles and is illustrated in figure 5.10. It depicts a scenario where a `BillOfLading` is being deployed. The oracle smart contract emits the event `CreatedBillOfLading`, which contains the address of the corresponding `charterparty`, the address of the newly deployed `billOfLading` and the address of the `creator`. The oracle server of the shipowner company, which listens to this particular event, looks up the DID-document of the `creator` in order to retrieve the service endpoint of the oracle server. This is followed by a *challenge-response* authentication, where the shipowners oracle needs to prove the ownership of their public key and the charterers oracle checks whether the oracle is whitelisted. Subsequently, the shipowners oracle makes a call to `GET /request`, represented as `request(..)`, with the address of the `billOfLading` as parameter. The charterers server then responds with the raw data of the `BillOfLading`.

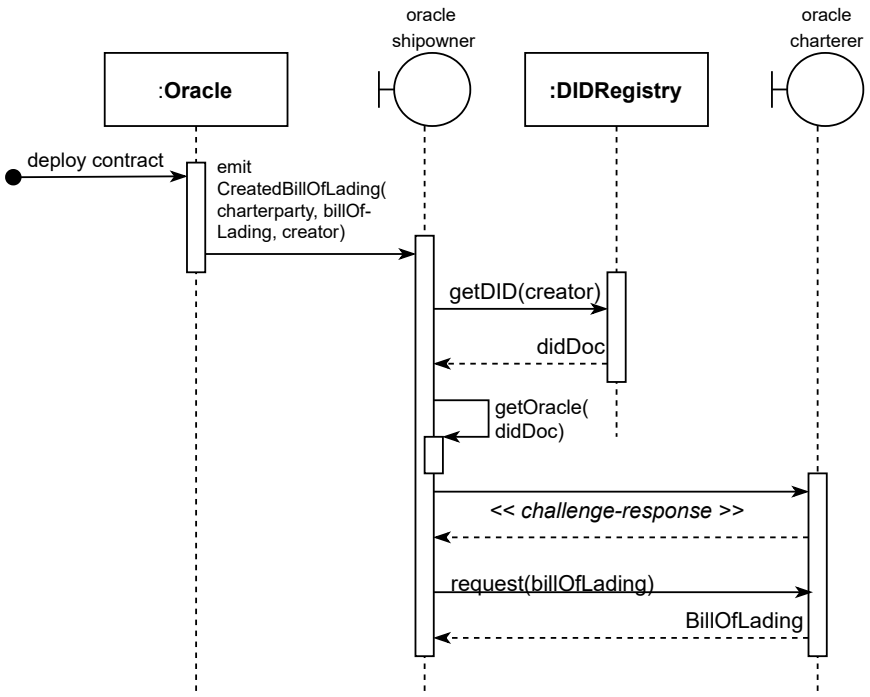

Figure 5.10: Oracle-to-Oracle Communication

5.6 Data-Flow Process

This section intends to tie the information presented in the preceding sections of this chapter together into a series of data-flow processes. Its purpose is to demonstrate how data is created, modified and transferred throughout the process. Figure 5.11 illustrates a chain of processes, starting with the Charterparty agreement between stakeholders `Charterer` and `Shipowner` and ending with the verification of the delivery data by the `Customer`. The diagram is not only able to illustrate how data moves throughout the process but also the user interactions enabling them. Figure 5.11 shows all possible user interactions with labeled arrows going from users to high-level processes. The `Blockchain` is the common entity between all processes and represents the aggregation of all operations to the smart contracts `Charterparty, BillOfLading, ProofOfDelivery` and `Oracle`. The figure only illustrates an abstracted form of data flow and is hence called *Level 0*. Each high-level process can thus be iteratively decomposed into multiple sub-processes until a detailed flow of data can be achieved. A next level decomposition of high-level processes 2.0-4.0 is depicted in Appendix B.

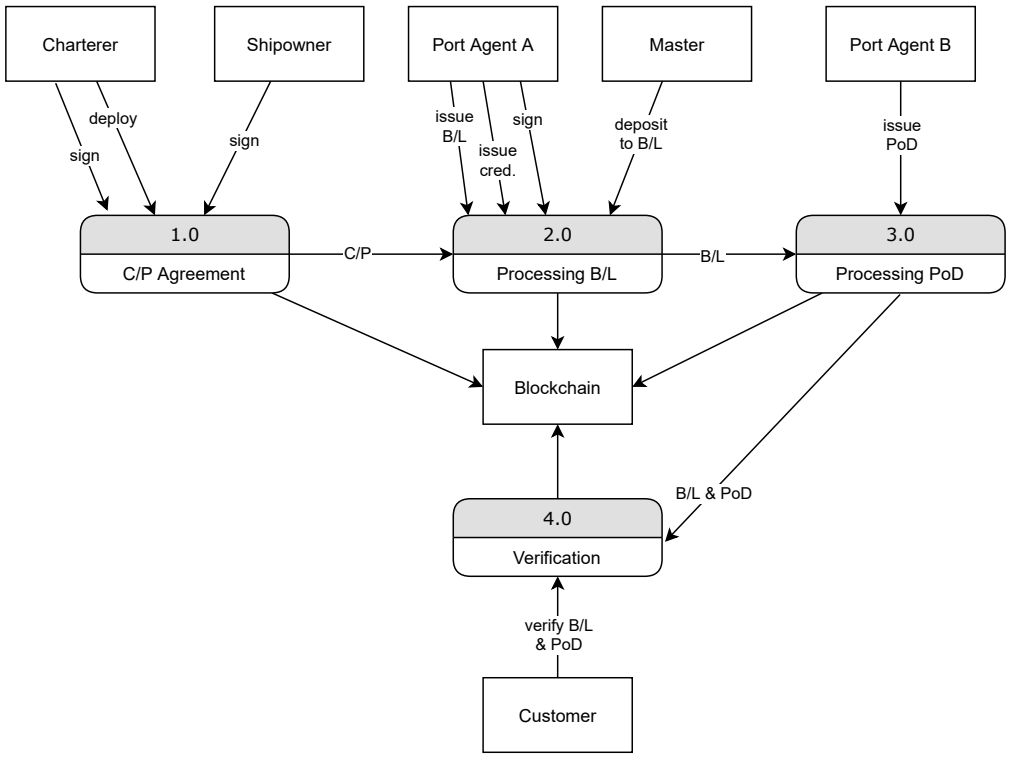

Figure 5.11: Data Flow Diagram - Level 0

Level 1 diagram of `Processing B/L` is illustrated in B1 and is by far the most complicated process. It has a total of 12 steps, some of which are triggered externally, i.e. by users, and others by internal entities such as oracle servers. Consequently, internal steps like oracle-to-oracle communications and database operations are made visible. Contrary to the `Oracle` entities, broker operations are implicitly handled by users themselves and are therefore not illustrated. Data flow to and from the `Blockchain` follows a certain pattern, where incoming arrows are new data inserted to smart contracts and outgoing arrows events emitted from smart contracts. This indicates that the `Blockchain` entity itself does not transform any data, but rather forwards it to other entities.

6. User Journey

This chapter illustrates the interaction of the stakeholders with the respective brokers. The first section presents the use case scenarios of each actor and the environment that they interact with. The second section shows the users front-end interaction and the process behind it.

6.1 Use Case Scenarios

Figure 6.1 shows use three case diagrams, each with their respective environment. The `Charterer` and `Shipowner` are responsible for making a charterparty (C/P) contract, in order officially agree to the terms of their cooperation. The `Charterer` is responsible for deploying the smart contract, but both parties need to digitally sign it in order to make it official. The charterparty must contain information about the maximum allowed moisture level, the maximum allowed laytime and the fine fee. This information is essential for performing automated checks on certain conditions, such as moisture level, laytime and weight.

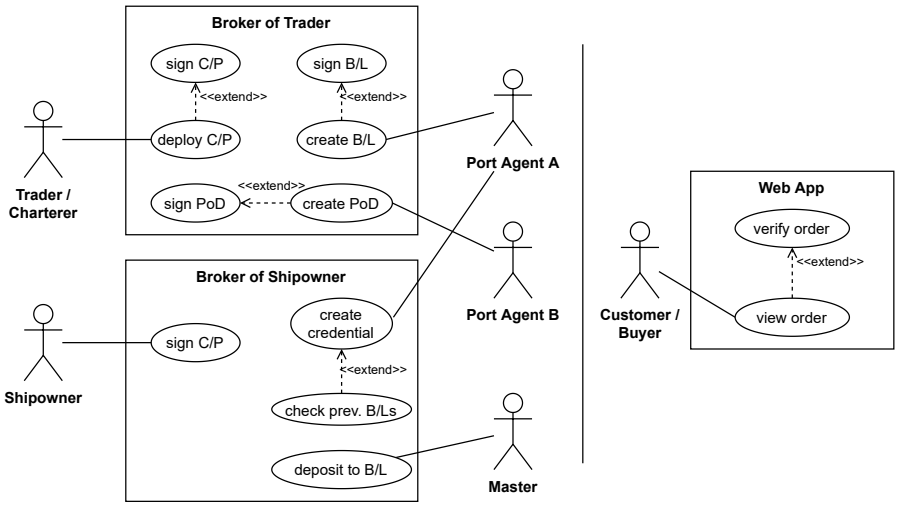

Figure 6.1: Use Case Diagrams

As the `Port Agents` act on behalf of the `Charterer`, they mostly interact with the `Charterers` system. These interactions include the issuance of the `BillOfLading`

and `ProofOfDelivery`. However, `Port Agent A` needs to access the `Shipowner`s system in order to review previous Bills of a vessel that they are inspecting. This is intended for determining whether the vessel should be cleaned based on the previous loads, i.e. dangerous chemicals. A successful review and inspection is followed by the issuance of a Verifiable Credential in order to certify the vessel as 'clean'.
The `Master` works for the `Shipowner` and hence uses the corresponding broker/front-end to interact with the system. Given the circumstance, that the `Shipowner` itself is not present at the port, the payment of the deposit needs to be delegated to the `Master`. The deposit intends to (1) pay `Port Agent`s for their work and (2) ensure that for any violation of the agreement, e.g. moisture level, the `Charterer` is compensated. Once the deposit is made the `Port Agent A` can proceed with signing the `BillOfLading`. This deposit strategy is aimed at maintaining the quality of the grain and thus addresses the research question 4.2.
`Port Agent B` can now proceed with issuing a `ProofOfDelivery` upon finishing the inspection of the vessel at the discharging port. The `Charterer` notifies the `Customer` of the delivery details, upon which the `Customer` can view the delivery on their system and verify them.
Figure 6.2 illustrates an overview of all tasks within a complete process. The tasks have been logically ordered and separated into sub-processes *C/P Agreement, Inspection & Loading, Unloading* and *Verification*. The charterparty defines cost agreements and constraints for the moisture level & laytime. The Bill of Lading contains information about the moisture level, weight and date at the time of loading. And Similarly, the Proof of Delivery contains the same information at the time of unloading.

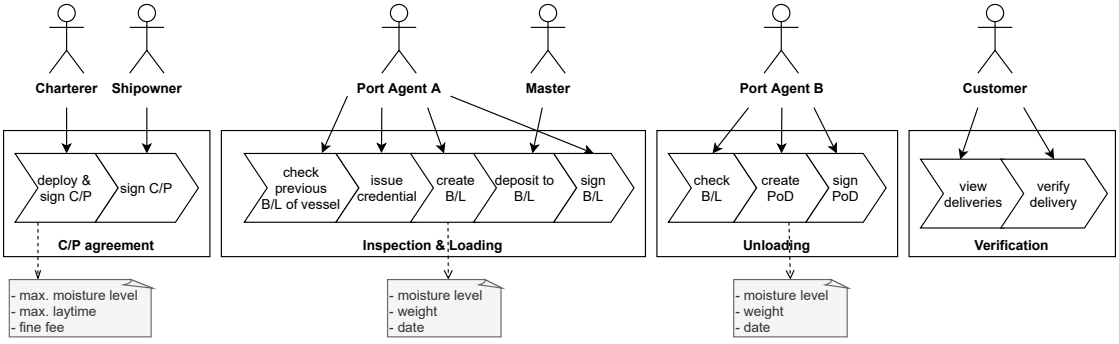

Figure 6.2: Use Case Process

6.2 User Interaction

The following illustrates a scenario for creating a shipment for an existing Charterparty agreement. As already shown in figure 6.2 the port agent at the loading port (*Port Agent A*) initiates the process by inspecting the vessel condition followed by the issuance of a credential. In theory, the user is located at the port and uses the external IDM server to access the respective broker front-end. The screenshot below shows the portal front-end, where the user uses their Ethereum private key

and the domain name of the broker to login. The server-side authentication scheme has been discussed in section 5.3.

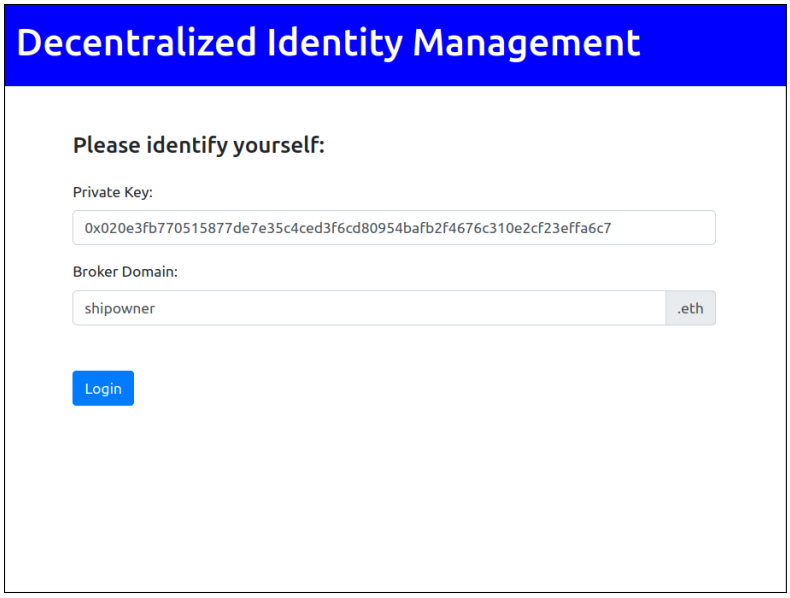

Figure 6.3: IDM Portal

The agent needs to issue the credential on behalf of the shipowner, hence they use the corresponding broker interface 'shipowner.eth' to view previous shipments (i.e. Bills of Lading). The intention is to check whether these shipments contained chemicals or other elements that could potentially taint the grain. Figure 6.4 shows the forwarded landing page, where the agent selects the vessel to issue a credential for.

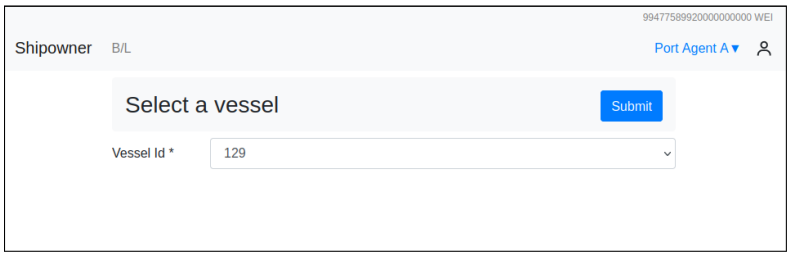

Figure 6.4: Selecting a Vessel for Inspection

After inspecting the corresponding Bills of Lading and deciding that the vessel is clean, the agent proceeds with issuing a Verifiable Credential as shown in figure 6.5. The *subject* refers to the DID-URI of the ships Master and the condition of the vessel is explicitly stated as 'Clean'.

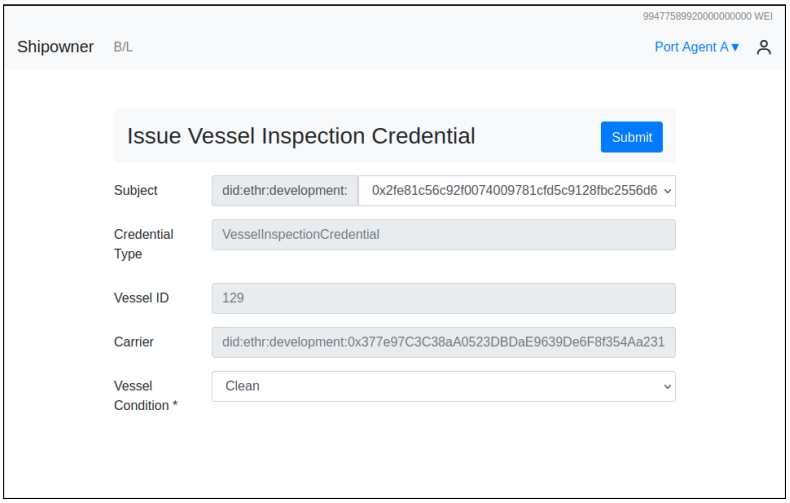

Figure 6.5: Issuing a Verifiable Credential

Now the agent uses the IDM portal to log into the charterers broker interface, which forwards to the landing page (see Figure 6.6) showing an unprocessed order. The order contains information about the corresponding smart contract of the Charterparty, date, the customer, description of the bulk-load, the expected moisture level, the weight and further information about the nutrition.

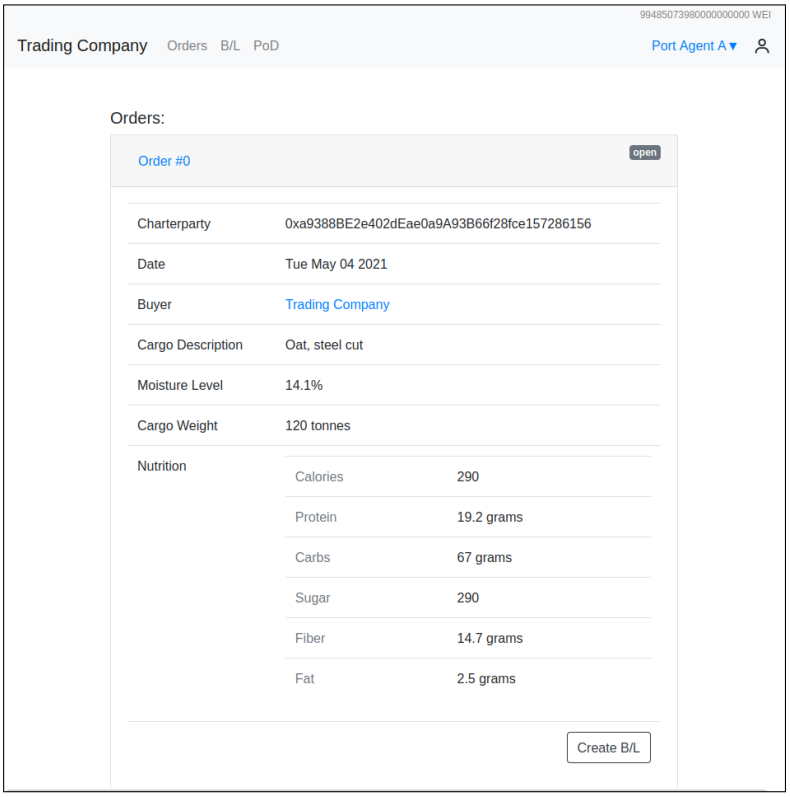

Figure 6.6: Landing Page of Trading Company

The port agent can create a shipment for this order by clicking ‘Create B/L’

at the bottom. The state of the order changes from *open* to *issued*, in order to prevent issuing multiple Bills of Lading for a single shipment. Once a shipment is complete, i.e. reaches the customer, the state changes to *closed*.

Figure 6.7 shows a form for issuing a Bill of Lading. Theoretically, the port agent should either carry out or oversee the measurement of the weight and moisture level on-site. Input fields which have been greyed out cannot be changed and are intended to provide more context to the user. For the remaining fields, the port agent provides information about the vessel, the port location, the measured weight and measured moisture level.

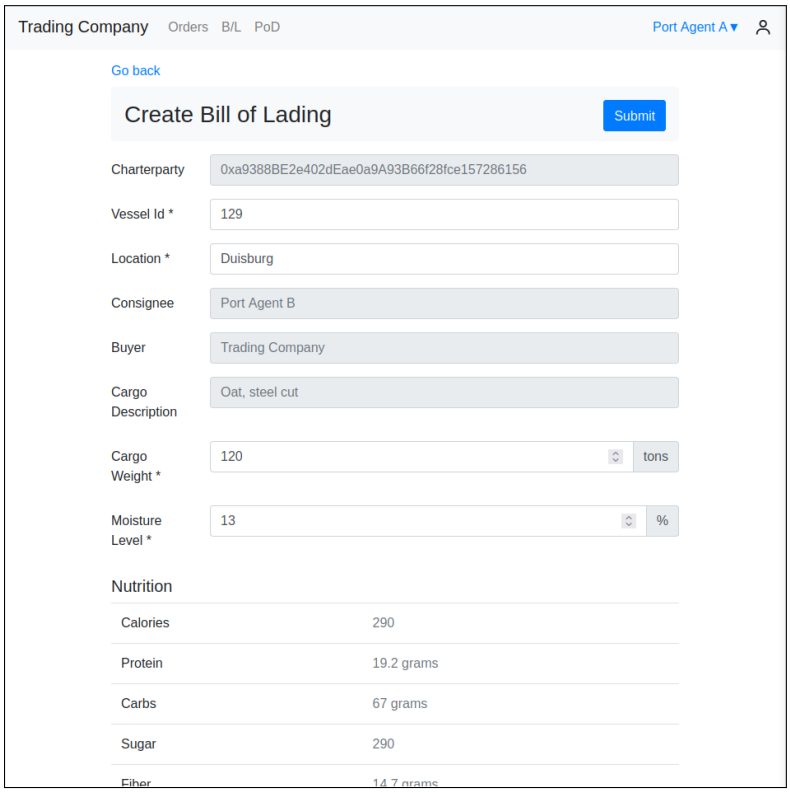

Figure 6.7: Create Bill of Lading

The aggregation for this Bill of Lading is illustrated in figure 6.8. The client uses the library, `Lib`, to generate a merkle tree from the information contained in the `BL` object. Afterwards, the client deploys an empty `BillOfLading` smart contract, which is then initialized using the `init()` method. The arguments of this method are: the contract address of the `Charterparty`, root of the merkle tree, the hash of the date, weight and moisture level. Finally, the agent sends the newly generated Bill of Lading and the merkle tree to the broker server, who saves them to the database.

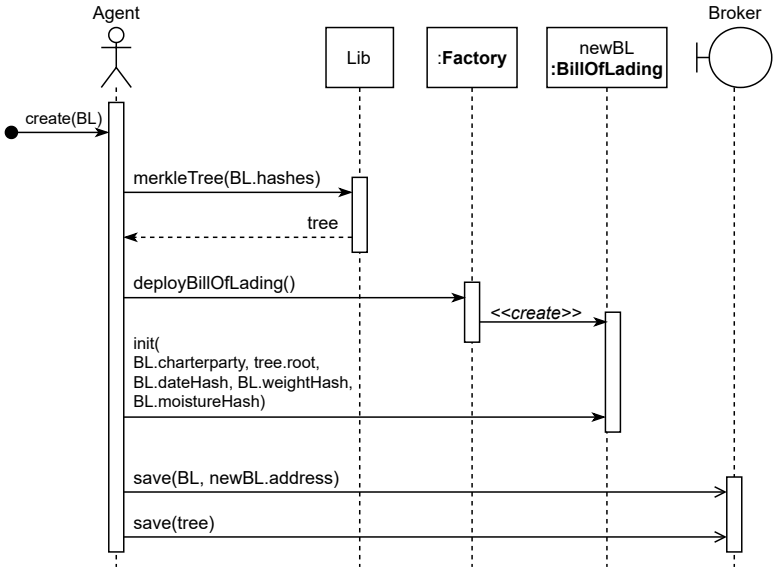

Figure 6.8: Data Aggregation of Bill of Lading

Now the master needs to transfer a deposit on behalf of the shipowner, which consists of two payments: port dues + fine fee. The former is intended for automatically paying port agents for their work and the latter is for compensating potential damages to the cargo. The master uses the IDM portal to login to the shipowners broker and opens the Bill of Lading, which is in a *pending* state. Figure 6.10 displays the view after clicking 'Confirm & Deposit'.

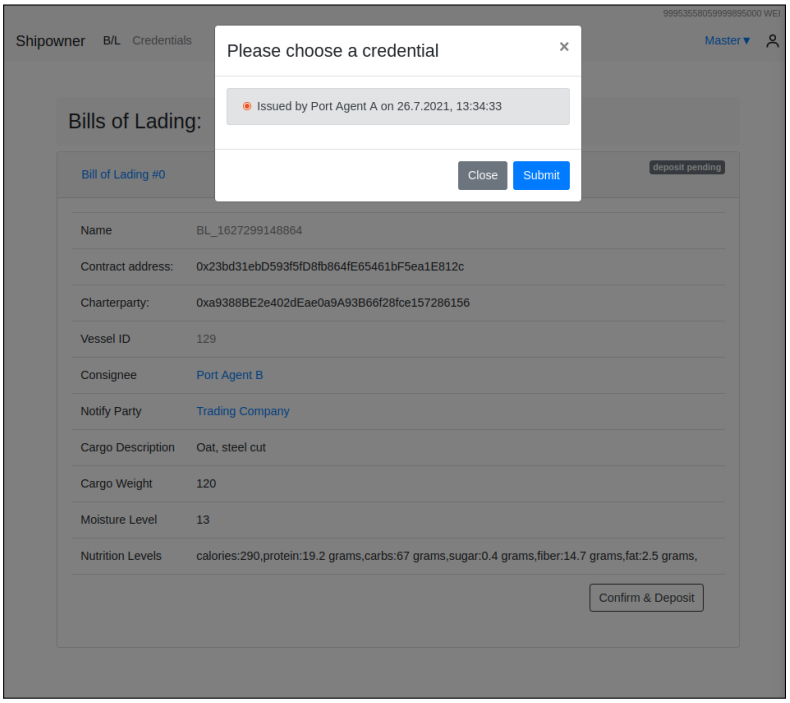

Figure 6.9: Depositing to Bill of Lading

The credential, which was previously issued and assigned to the master is automatically displayed. Upon clicking 'Submit', the deposit is transferred to the `BillOfLading` smart contract.
The following sequence diagram shows the interaction with the underlying smart contracts. The master submits the Verifiable Credential in JWT format to the `BillOfLading`. The shipowner has previously transferred Ether to its `Delegatable` contract, in order to allow authorized users to make payments on behalf of the shipowner. Thus, the master calls `payContractFee(bl)`, which takes the contract address of the `BillOfLading` as argument. The `Delegatable` fetches the fine amount, `fineFee`, determined by the charterers `Delegatable` and calls `deposit(fineFee)` in order to transfer the fine to the `BillOfLading`.

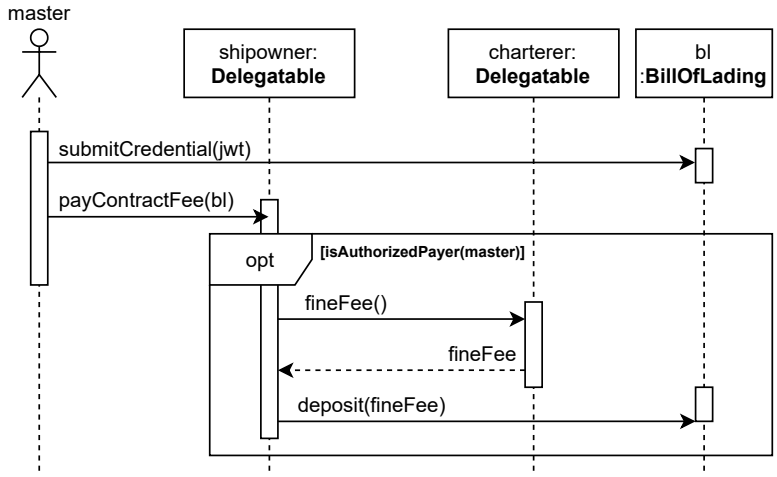

Figure 6.10: Master Interacting with the Smart Contract

Once the deposit is completed, the port agent is notified about the changes on their front-end. Figure 6.11 shows the updated view of the Bill of Lading with status 'deposit paid'. The credential, submitted by the master, is fetched from the Blockchain and when clicked resolves to a Verifiable Credential (see Appendix C2). The agent can now cryptographically sign the contract with their Ethereum private key by clicking the 'Sign contract' button.

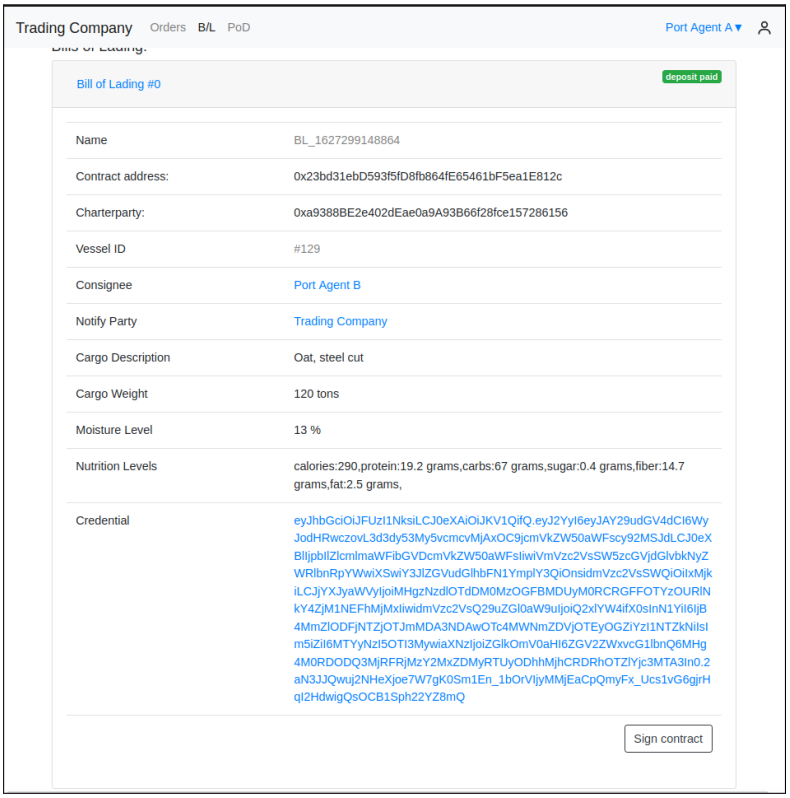

Figure 6.11: Submitting a Signature to the Bill of Lading

The sequence diagram shown in 6.12 demonstrates the interaction with the smart contracts in the background. After calling `submitSignature(signature)`, the contract checks whether the deposit was already paid and proceeds to fetch the payment amount (port due). The `amount` is transferred to the port agent invoking this contract, i.e. *Port Agent A*. Finally, an event `SignedContract` is emitted, in order to inform all interested stakeholders about the changes. This event is listened to by the shipowners *Oracle* server, which updates its own data.

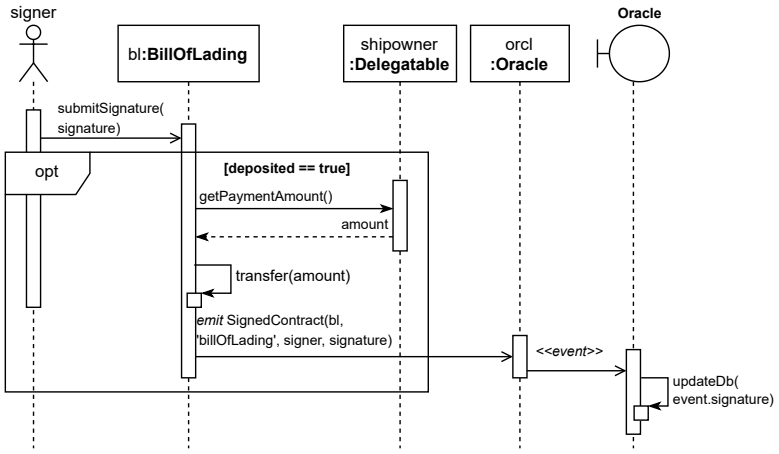

Figure 6.12: Agent Signing the Smart Contract

With the submission of the signature, the loading process of the cargo is officially completed. The cargo can now be transported to the discharging port, where the on-site agent, *Port Agent B*, will be waiting.
Upon arrival of the vessel, the agent carries out or oversees the measurement of weight & moisture level and inspects the grain. The agent uses the IDM portal to login to the charterers broker, opens the Bill of Lading of this cargo and clicks on the 'Issue Proof of Delivery' button (see Appendix C1). Figure 6.13 shows the form for issuing a Proof of Delivery. The agent inputs the time and day of delivery, the measured weight & moisture level and the condition of the cargo.

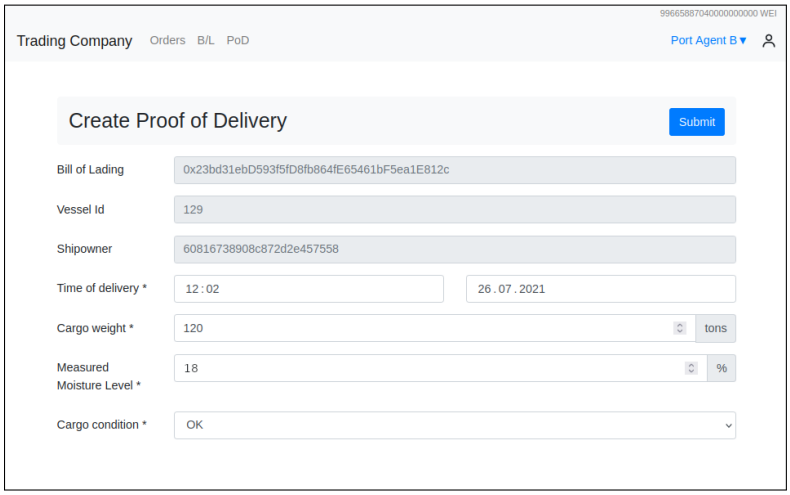

Figure 6.13: Issuing a Proof of Delivery

The client performs a data aggregation for the Proof of Delivery, similarly to the previously shown aggregation method in figure 6.8. As shown in the next figure 6.14, the first step upon deploying and initializing the state of the contract is to transfer the funds from the corresponding `BillOfLading` to the `ProofOfDelivery`. Once transferred, the contract initiates an automated criteria check of the supplied variables: `cargo_moisture_level`, `cargo_quantity`, and `time_delivery`. The automated check is implemented in a callback method, which is supplied together with the hashes to the `makeBatchQuery` function. The oracle generates a unique number for each batch query and temporarily stores the number with the corresponding `callback` in a `mapping` (as defined in figure 5.8). Then it emits an event with the generated unique number, the contract address and an array of the hashes to be queried. The oracle servers listening to query events for this particular contract check their database for the hashes listed in the array and if found respond with the raw data by calling the `submitBatchQuery` function. The oracle smart contract retrieves the callback function from the `mapping` by the unique number and invokes it with the supplied raw data. Finally, the callback function performs three consecutive checks, where for each satisfied condition a part of the deposit is refunded to the *shipowner*. In case a condition is not satisfied a `warn` event is emitted, in order to record the negative outcome. At the end of the function the remaining funds are transferred to the *charterer*.

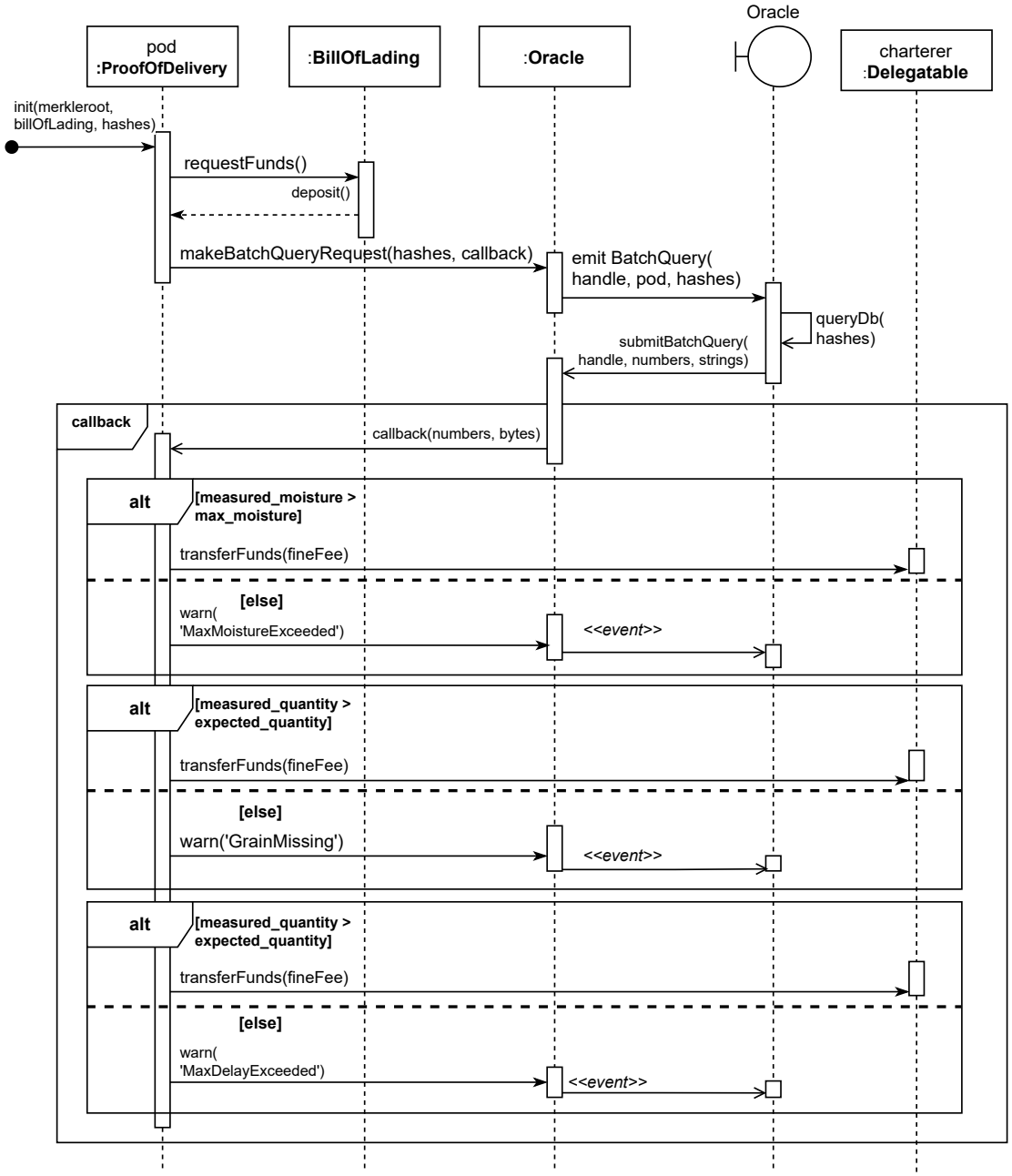

Figure 6.14: Oracle Requesting Data Off the Chain

The following figure 6.15 shows the view of a Proof of Delivery, which was signed by the agent.

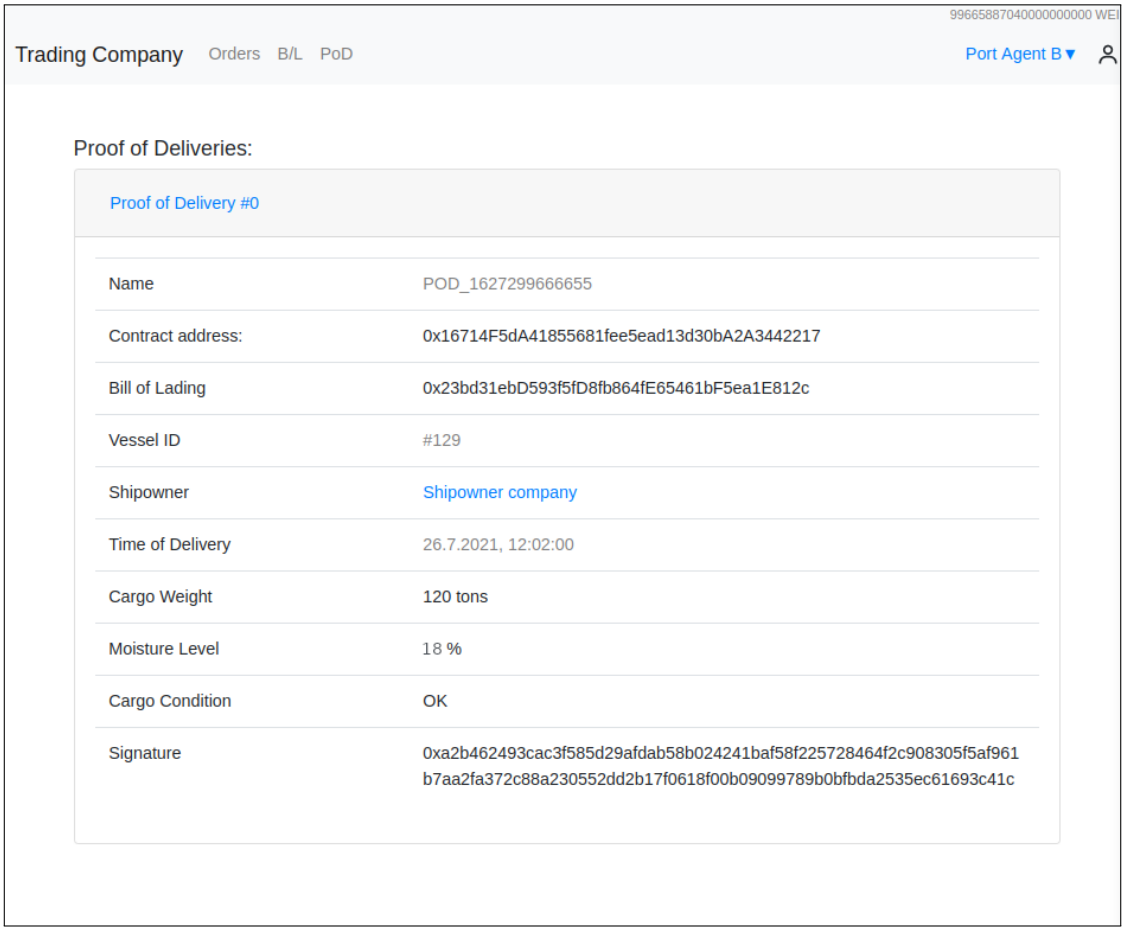

Figure 6.15: View of Signed Proof of Delivery

The signing process on smart contract level is shown in figure 6.16. Upon submitting the signature to `ProofOfDelivery`, it first emits a `SignedContract` event, in order to notify interested stakeholders about the state of the contract. It then queries the port due, `amount`, from the `Delegatable` contract and transfers the payment to the signer, i.e. *Port Agent B*. And finally, it checks whether the balance of the contract is greater than 0, meaning that the deposit was not fully paid. If the balance is greater, it refunds the deposit, or the remainder of the deposit, back to the shipowners `Delegatable contract`.

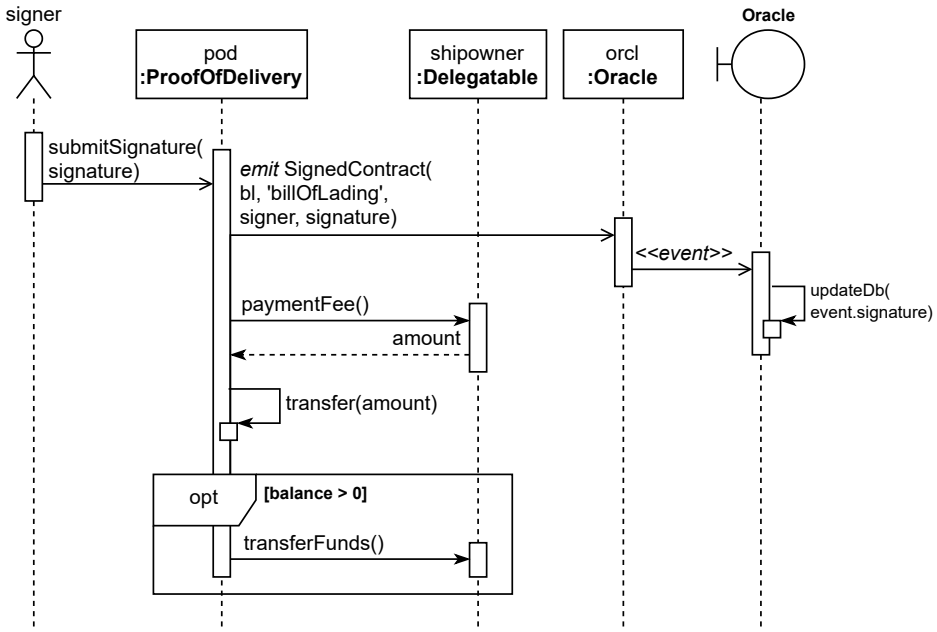

Figure 6.16: Agent Signing Proof of Delivery Contract

The *Inspection & Loading* and *Unloading* sub-processes are hereby completed. The raw data of the Bill of Lading and Proof of Delivery is automatically sent to the customers server and contains the delivery data shown below.

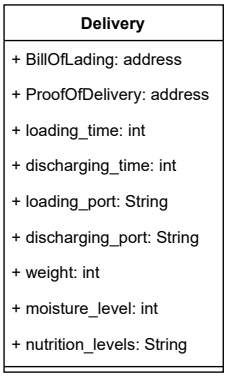

Figure 6.17: Delivery Datastructure

The following figure shows the details of a delivery, which has not been verified yet. It shows the contract addresses of the `BillOfLading` and `ProofOfDelivery`. Furthermore, there are details about the departure/arrival time, weight, moisture level and nutrition of the cargo. Any potential violations during the shipment can be discovered by querying past `warn` events. As shown in the figure, the moisture level is annotated with an exclamation mark stating 'Quality violation: moisture level is too high'. This indicates that the recorded moisture is higher than it was initially agreed on the Charterparty.

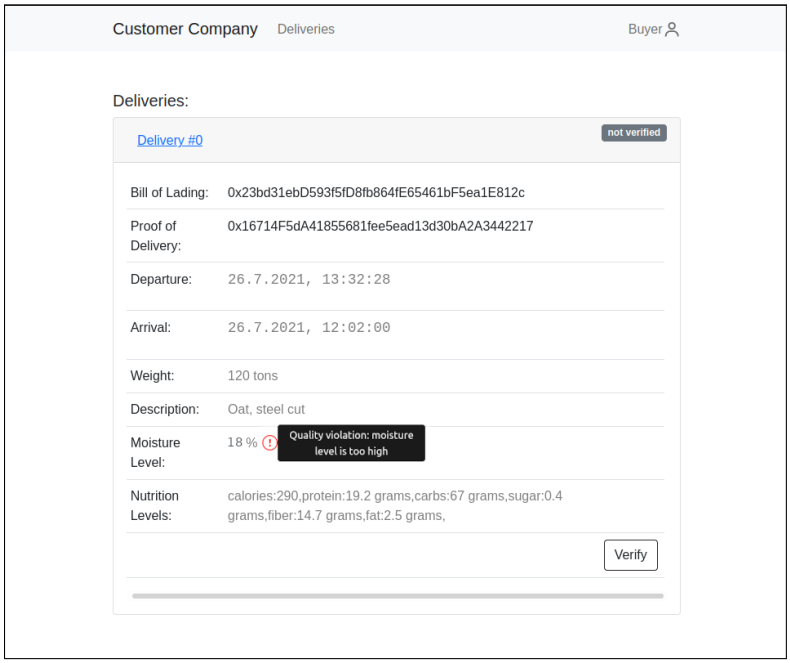

Figure 6.18: View of Deliveries

In order to verify the delivery, the customer clicks the 'Verify' button, which displays the modal shown in figure 6.19.

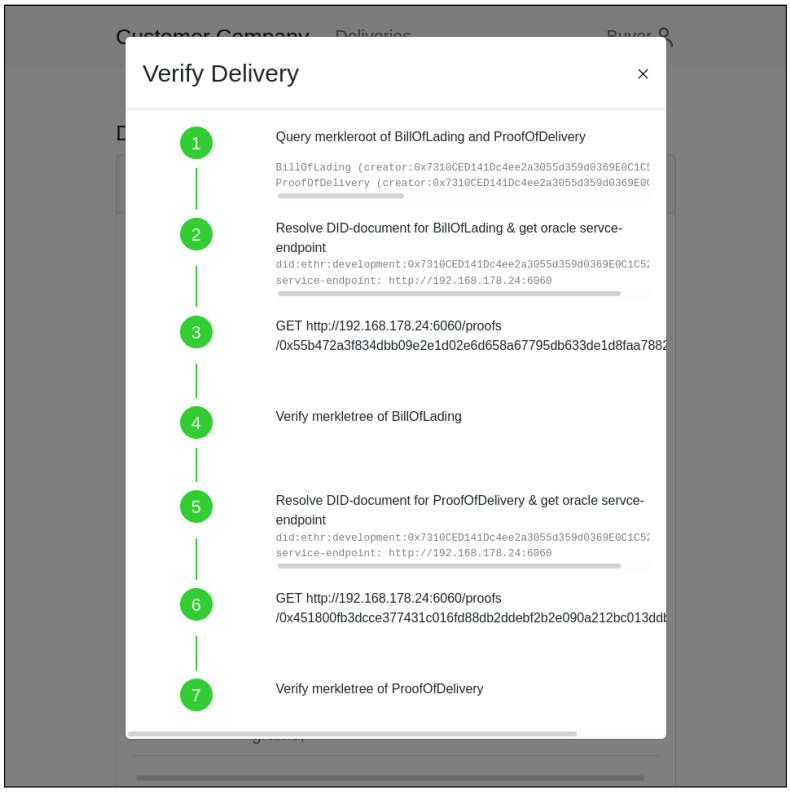

Figure 6.19: Verification of Delivery

The modal shows each verification step, which are in total comprised of the individual verification of the Bill of Lading and the Proof of Delivery. The customer can access the hashed information and merkleroot using the contract addresses provided in the delivery. The merkleroot of the Bill of Lading consists of `loading_time, loading_port, nutrition_levels` and other data, whereas the Proof of Delivery consists of `discharging_time, discharging_port, moisture_level, weight` and other data.
The sequence diagram in 6.20 shows the process of verifying the merkleroot of a `contract`, which is an indexed parameter of the smart contract event defined in the oracle class diagram in 5.8. The customer first filters for past `Merkleroot` events by the given `contract` parameter from the Blockchain. The returned `event` object contains the account/contract address of the `creator`. Subsequently, the customer retrieves the DID-document of the `creator` by querying the `DIDRegistry` and fetches the service-endpoint of the oracle. Next, the customer retrieves the proof (i.e. Merkletree) of the merkleroot from the oracle. Finally the customer verifies the merkleroot with the retrieved proof and raw data. This procedure is applied both to `BillOfLading` and `ProofOfDelivery`.

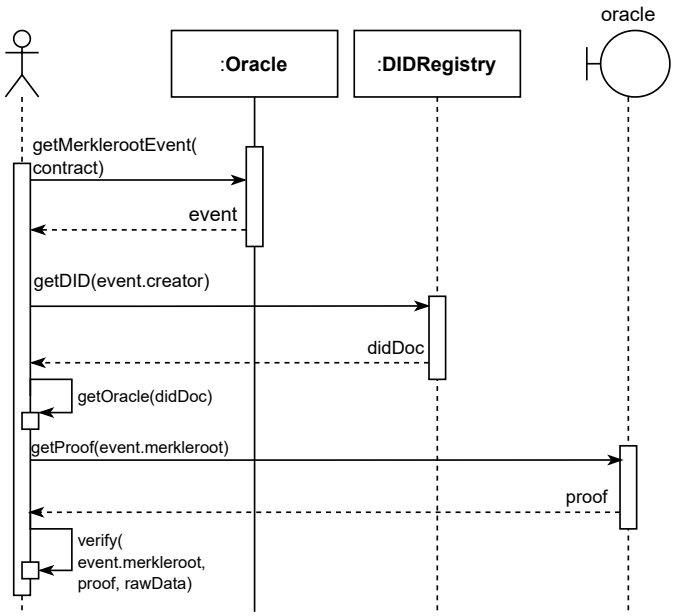

Figure 6.20: Verification of Smart Contract

The presented interaction process involves both port agents, the master and the customer. It consists of 7 tasks, starting from the issuance of a Verifiable Credential up to the verification of a delivery. The data aggregation is conducted using merkle trees and their roots, which are stored on-chain. The merkle tree is later used by the customer to verify the provenance of their shipment. With the help of the oracle-to-oracle communication protocol, external stakeholders can exchange and synchronize their data. Moreover, the payment of penalties and port dues are automated using smart contracts and off-chain oracle requests.

7. Evaluation

This chapter deals with the evaluation of the proposed approach and consequently the implemented prototype. The evaluation process is subdivided into two parts: a quantitative analysis and a qualitative analysis.

7.1 Quantitative Analysis

The quantitative analysis is intended for measuring the performance and cost of transactions. The performance metric measures the transaction latency, i.e. the time it takes to mine a transaction, while the second metric measures the transaction cost. As both metrics cannot be measured in an artificial setting, an estimation approach will be applied. Transactions on the Ethereum Blockchain are mined based on their gas price, which indirectly affects the mining delay. The gas price fluctuates based on the network congestion, which is why the latency of a transaction is of a higher importance than its cost. For this reason, the latency estimation will be analyzed first and its resulting gas price estimations will be used for estimating the cost.

7.1.1 Latency Estimation

Zhang et al. evaluate the end-to-end latency of transactions on the Ethereum Blockchain. The authors analyze the time it takes for a transaction to be mined after it was just submitted to the mempool. New transactions are flagged by Ethereum as `pending` and are immediately broadcasted to clients listening for new pending events. The authors use the block-explorer *Etherscan*[1] to acquire a sample of new pending transactions and use the `web3js`[2] library to later check whether the transactions have been mined and when they were mined. The latency is determined by the time a transaction was submitted and by the time it was mined [34]. The authors use samples from May 2019, which are considered outdated as the user-base and the node size of the Ethereum Blockchain have grown immensely over the past couple of months (since January 2021). Therefore, the sampling approach presented in the paper was used to acquire new data using the `web3js` library. The

[1] https://etherscan.io
[2] https://web3js.readthedocs.io/

sampling itself consisted of two parts: (1) acquiring new pending transaction and (2) checking whether pending transactions have been mined. The second part was performed a couple of hours later, in order to include longer pending transactions. Seeing that the Ethereum Blockchain is very volatile, during certain times of the day the network is highly congested. For this reason, the sampling was performed twice: once where the network was highly congested and once where it was normal. By design, the end-to-end latency of transactions depend on the gas price that was set by the sender. Miners are interested in mining transactions with higher gas prices, since their rewards are higher. Hence, the end-to-end delay was analyzed based on the gas price of the transactions.

Figure 7.1 shows the result of the sample, which consists of roughly 3,500 transactions. The x-axis depicts the gas price in Gwei and the y-axis the transaction latency in seconds. The dots represent transactions and the colors represent sample groups, where one group was sampled when the network was congested (red) and the other when the network was regular (blue). The maximum delay recorded was 67,054 seconds (ca. 18 hours) and the maximum gas price was 1,706 Gwei.

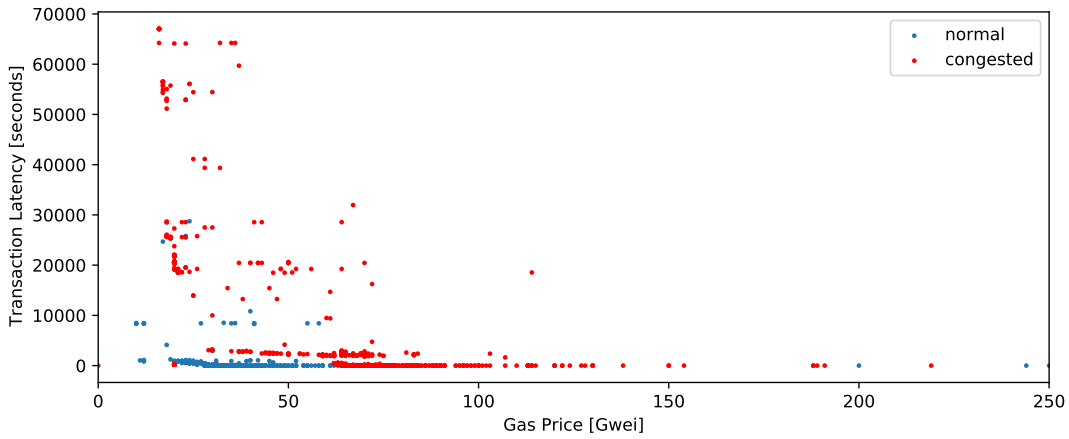

Figure 7.1: Transaction Delay over Gas Price

The following figure shows a zoomed-in view of the plot, where differences between the sample groups are more transparent. The sample contains transactions with gas prices higher than 10. This is due to transactions with low gas price remaining in the pending state for days. *Etherscan* shows transactions with gas price lower than 1 being in the pending state for more than 6 days.

The blue sample shows transactions with gas price higher than 35 being mined almost immediately (i.e. within 5-15 seconds). Transactions with gas prices between 20 and 28 are mined within 500-1000 seconds (8-16 minutes). The red sample has significantly higher gas prices and latency. As the network is more congested, i.e. more transactions are being submitted, users have to compete with other transactions and therefore set a higher gas price. Figure 7.1 shows transactions starting with gas price 20 having a latency higher than 20,000 seconds (ca. 5.5 hours). In

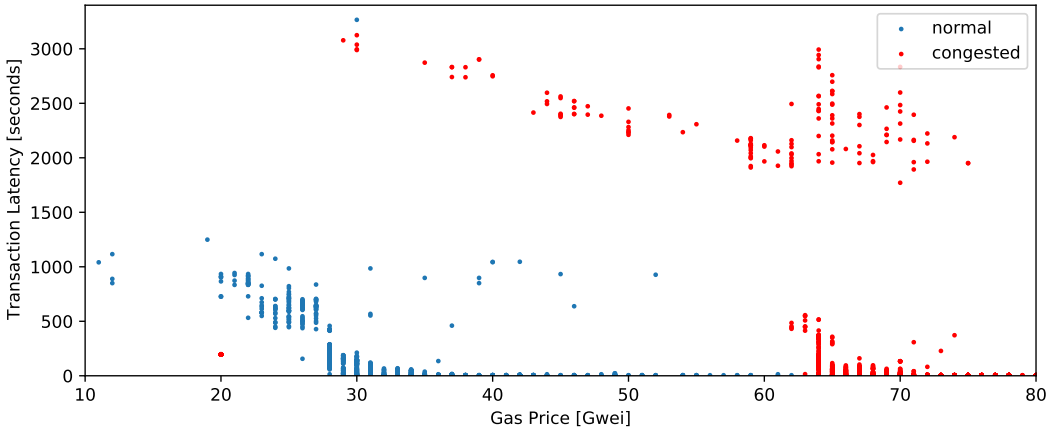

Figure 7.2: Transaction Delay over Gas Price (Zoomed In)

comparison, transactions with a gas price higher than 30 have a chance of being mined roughly within 3000 seconds (50 minutes). However, in order for transactions to be mined within a couple of minutes a gas price higher than 62 has to be set.

The following table shows a comparison of both samples, with the categorization of mining speed based on gas price. The gas price in the congested network is three times higher for slow and normal mining and at least twice as high for fast mining.

network	**slow** ($> 1h$)	**normal** ($< 15min$)	**fast** ($< 60s$)
normal	$<$ 10 Gwei	$>$ 20 Gwei	$>$ 28 Gwei
congested	$<$ 30 Gwei	$>$ 62 Gwei	$>$ 70 Gwei

Table 7.1: Gas Price Estimation

In order to ensure timely mining for the user activities presented in section 6.1, it is imperative to pick a safe gas price. Therefore, a price of at least **65 Gwei** would be suitable under most circumstances. Fortunately, Ethereum 2.0 (*ETH2*) will have a significantly faster transaction throughput, which is expected to lower the gas price. The block explorer of the Beacon chain[3], a fork of Ethereum 2.0, shows gas prices around **20 Gwei**. It can be assumed that the gas price will be around the same range of the regular Ethereum network, when it is not congested.

7.1.2 Cost Estimation

For most companies cost is a determining factor when adopting a new solution. The digitization of business processes consumes a lot of costs, some of which originates from the maintenance and operation of servers, devices or transactions. The increasing popularity of the Ethereum Blockchain has been accompanied by increasing transaction costs as a consequence. This becomes a great concern for

[3]https://beaconscan.com

complex smart contract transactions. In order to estimate the cost of the implementation, an analysis of the most used functions was performed and its results summarized in table 7.2. For reference, the standard gas consumption for a transaction of ether is around **21,000**.
The costs for the charterparty agreement and DID-operations were excluded as they occur less frequently. The cost estimation of transactions depends on the gas price and the gas consumption of the transaction in question. For this reason, the consumption for each function was measured with the help of the `web3js` API. The factory functions `createBillOfLading()` and `createProofOfDelivery()` deploy new contracts to the Blockchain and hence have almost the same amount of consumption. The deployment of smart contracts are generally considered expensive, which is why the `Factory` contract uses a cloning library to reuse the bytecode of the smart contract and initialize it with a new state. The deployment of a `BillOfLading` without the `Factory` consumes around 1,580,156 of gas, which is at least 14 times higher than the gas consumed with the help of the `Factory`. The function `submitCredential()` has the lowest gas consumption, as it only emits an event without changing the state of the smart contract. In contrast, `ProofOfDelivery.init()` has the highest consumption, as it performs more complex operations.

Function	**Actor**	**Gas**
Factory.*createBillOfLading()*	Port Agent A	109,938
BillOfLading.*init()*	Port Agent A	284,462
BillOfLading.*submitSignature()*	Port Agent A	144,533
BillOfLading.*submitCredential()*	Port Agent A	24,826
Delegatable.*payContractFee()*	Master	85,038
Factory.*createProofOfDelivery()*	Port Agent B	109,829
ProofOfDelivery.*init()*	Port Agent B	428,466
ProofOfDelivery.*submitSignature()*	Port Agent B	159,622
Oracle.*submitBatchQuery()*	Oracle (charterer)	83,480

Table 7.2: Gas Consumption Estimation of Functions

The following table presents the total gas cost for each function with respect to the type of Ethereum version used. The total gas is computed by multiplying the gas consumption (see table 7.2) and the gas price. As mentioned before, the gas price for Ethereum 1.0 (ETH1) is set to *65 Gwei* and for Ethereum 2.0 (ETH2) to *20 Gwei*. The gas costs were converted to Euro, in order to interpret the prices better. The current rate of 1 ETH is €2192.68 (as at August 2021).

Function	**ETH1** (Gwei)	**ETH2** (Gwei)	€ ETH1 / ETH2
Factory.*createBillOfLading()*	7,145,970	2,198,760	€15.67 / €4.82
BillOfLading.*init()*	18,490,030	5,689,240	€40.54 / €12.47
BillOfLading.*submitSignature()*	9,394,645	2,890,660	€20.6 / €6.34
BillOfLading.*submitCredential()*	1613690	496520	€3.54 / €1.09
Delegatable.*payContractFee()*	5,527,470	1,700,760	€12.12 / €3,73
Factory.*createProofOfDelivery()*	7,138,885	2,196,580	€15.65 / €4.82
ProofOfDelivery.*init()*	27,850,290	8,569,320	€61.07 / €18.79
ProofOfDelivery.*submitSignature()*	10,375,430	3,192,440	€22.75 / €7.00
Oracle.*submitBatchQuery()*	5,426,200	1,669,600	€11.9 / €3.66

Table 7.3: Cost Estimation of Functions (Aug. 2021)

The cost of transactions for both Ethereum versions are extremely high. The execution of a business process in ETH1 would cost ca. €200 and in ETH2 ca. €60. The reason behind such high gas consumption is the size-limitation of smart contracts. According to EIP-170[4] (Ethereum Improvement Proposal) the limitation was introduced in order to prevent denial-of-service attacks. The proposal specifies that a contracts bytecode cannot be bigger than 24.576 kb. The bytecode increases if a contract has a state variable referencing another contract instance. In order to reduce the bytecode size contracts are instead referenced as `address`. Unfortunately, the drawback of this approach is that the resolving of the address during runtime is expensive. The following table shows the bytecode sizes of the smart contracts. The `BillOfLading` contract references `Charterparty` by its address and similarly the `ProofOfDelivery` references `BillOfLading` by its address. For this reason, the gas consumption is significantly increased compared to `Delegatable` or `Oracle` (see table 7.2).

Contract	**Byte Code Size**
Charterparty	11,153 kb
BillOfLading	14,209 kb
ProofOfDelivery	15,527 kb
Oracle	4,885 kb
Delegatable	11,641 kb

Table 7.4: Bytecode-Sizes of Contracts

7.2 Qualitative Analysis

This section will present and analyse the qualitative criteria of the implemented approach. The first section will reflect on the goals and problems presented in the introduction and evaluate how they were addressed. The second section will discuss the benefits and limitations of the resulting prototype.

[4]https://eips.ethereum.org/EIPS/eip-170

7.2.1 Reflection on Goals and Problem Statement

The objective of this section is to reflect on the initially defined goals and problems and evaluate whether they have been addressed and fulfilled throughout the thesis work. As stated in the first chapter, the thesis will address two problem areas: **A.** the challenges posed in the domain of waterway transportation with the specific use case of grain transport and **B.** the limitations of on-chain Blockchain solutions. The following presents each criteria and how they were addressed:

A.1 Digitization
Another concern of the current business process in waterway transportation is the exchange of paper-based documents. Paper documents can not only be easily forged by malicious actors, but can also get lost during the shipment. Moreover, sending documents to customs takes time and often causes delay in the overall business process. All these issues were resolved by digitizing relevant paper documents into smart contracts, which cannot be forged and will always be available.

A.2 Trust
The most important issue in the supply chain is the trust between cooperating companies and actors. In the particular case of waterway transportation, the certification and origin of goods is the primary focus. Certificates are used for vessel inspections, in order to guarantee the customer that their shipment is free of any harmful chemicals and substances. These certificates can only be trusted by customers, if they are able to verify the identity of the issuer and the certificate itself. This is accomplished with the use of decentralized identifiers and Verifiable Credentials, which allow Blockchain-based authentication and verification of users and certificates without involving a 3rd party. The authenticity and provenance of the shipment data can be checked using signatures and merkle trees. The smart contracts were designed in such a way that only certain actors have the authority to issue and sign digital documents. This implicitly provides an additional layer of trust and authenticity to the data on the smart contract, which contains the merkle root and hence can be verified off-chain.

A.3 Automation
The digitization opens up opportunities for automating trivial tasks, one of which is automatically compensating port agents for their work. However, one use case specific objective was to ensure the quality of the goods (grain) during shipment. The quality indicator for the grain is its moisture level, which can easily be influenced by environmental conditions during shipment. In order to incentivize all parties to enforce optimal conditions, a deposit strategy was incorporated into the smart contracts. As the master is responsible for the shipment on board, the shipowner company, who employs the master, needs to deposit a fee. At the end of the shipment process, the conditions will be automatically evaluated by the smart contract and the fee transferred back to the shipowner company if all conditions are met. This

strategy adds an additional degree of security for the trading company, as they take the financial burden of the cargo up until it reaches the customer.

B.1 Scalability
One of the few limitations of current Blockchain technologies is their lack of scalability, both in terms of throughput and memory. Supply chain processes generate a tremendous amount of data and it is one of the reasons why businesses hesitate to adopt DLT based solutions. The system architecture presented in chapter 5.1 was designed to address scalability by introducing a decentralized off-chain storage component, which interacts with the Blockchain via oracles. However, merely storing data to off-chain storages does not guarantee the integrity of data like Blockchain does. Hence, a data aggregation scheme was employed, where raw data was assembled into a merkle tree and its merkle root stored on all smart contracts. The root itself is a hash of type `bytes32`, which takes the minimum possible amount of data storage in a smart contract.

B.2 Privacy
The second biggest concern for businesses is the privacy of data. The Blockchain is by design accessible to everyone in the world, which poses a risk for exposing sensitive user data and company secrets. As mentioned in the previous paragraph, data is aggregated using merkle trees into a single cryptographic hash, which cannot be reverse engineered without knowing the raw data. Other sensitive data on the Blockchain is also stored in form of hashes. This strategy allows companies to make use of the public Blockchain and its consensus mechanism without having to reveal sensitive information.

7.2.2 Discussion of the Implemented Approach

This section intends to further discuss and analyse the benefits and limitations of the implemented approach.

While the approach aims to give companies the freedom to choose their underlying systems (e.g. Database Server or Server Framework), it nevertheless puts restrictions on inter-server communications. Both the IDM and oracle implementations adhere to a predefined protocol, in order to exchange information between servers. Any company who intends to join the ecosystem is forced to comply with the predetermined protocol. Moreover, all companies need to agree to a unified format for the *Charterparty, Bill Of Lading* and *Proof of Delivery*, in order to use the same smart contracts. Fortunately, there are many standardizations for these documents with particular focus on grain shipment available. The implemented smart contracts used standardizations defined by *BIMCO*, an international shipping association[5].

[5]https://www.bimco.org/contracts-and-clauses/bimco-contracts/graincon

Another administrative hurdle that might deter companies from adopting this approach is the obligation of all third party actors to participate in this ecosystem. This means, that the port agency needs to assume the role of a trusted certification authority, in order to issue Verifiable Credentials to their port agents. Moreover, ports need to deploy a local IDM server, which can only be accessed by the local network. This requires companies to know IDM servers beforehand, so that they can white-list them.

Aside from administrative hurdles, there are also technical limitations posed by this approach. The availability of data at all times cannot be guaranteed. Consider the circumstance, where a port agent issues a Proof of Delivery, but the charterer's oracle server is down. This would mean that the smart contract would be in an incomplete state for an indefinite amount of time and hence cannot be signed. The process could potentially remain unfinished forever. There are many other scenarios where the absence of a server would pose a problem. Other limitations are a result of the underlying Distributed Ledger Technology. One possible concern is the immutability of the smart contract code. A contract cannot be changed afterwards and needs to be replaced by a new one. However, there are ways to make contracts upgradeable using libraries by OpenZeppelin[6]. This solution only overrides old function calls with new functions having the same signature, but does not allow adding new function signatures. Furthermore, upgraded contracts are resolved during runtime, which makes all function calls on this contract expensive. Another limitation that makes function calls expensive is the size restriction of smart contracts. This means that implementing more complex contracts is either very difficult or simply not possible. And in order to reduce bytecode size, one would have to make more expensive function calls during runtime.

There are also some potential vulnerabilities of the implemented prototype. All test scenarios were carried out in an artificial setting, meaning that transactions were performed on a local Blockchain node simulating the Ethereum network. As a result, all transactions were executed immediately when in reality this is not the case. Whenever the user uses the broker to submit data to a smart contract, the broker first performs the transaction and *then* persists the data to the database. There is a chance that a transaction takes a long time to finish, and in that time the user might lose the connection to the server and after reconnecting the transaction would disappear. In this case the data would not be persisted to the database, forcing the user to repeat the transaction and hence pay the transaction fee twice. While the implemented smart contracts do not store raw data, they still do expose information about the actors. If the identity behind Ethereum addresses are known, meta information about a shipment can be reconstructed: e.g. which trader works with which shipowner, which port agents were hired and where the port agents work.
One might also argue that losing the private key of an Ethereum account could pose a risk to the business process. Fortunately, the standardizations for DIDs

[6]https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable

and Verifiable Credentials allow them to be revoked at a later time. This would allow users to use a new account and prevent malicious actors of exploiting stolen accounts to get access to certain data and smart contract operations.

By design, the system architecture enhances the efficiency of business processes, as the servers are able interact with each other. Both the charterer and shipowner have to track and save the state of digitized documents, such as the Bill of Lading and Proof of Delivery, but have their separate servers. In order to synchronize changes to documents, the oracles listen to Blockchain events and contact other oracle servers to retrieve information. This step not only automates data exchange, but also eliminates potential user errors. Consequently, port agents do not have to supply data twice and companies get updates as soon as possible, thus saving time.
The implementation also targets the security of users by instating authentication and verification methods. This issue is mainly handled by the IDM server, which allows users located at ports to be authenticated in a local network. The use of local networks intends to prevent Man-in-the-middle attacks, thus adding an extra layer of security. Moreover, users receive expirable session tokens, which ensures that they cannot be reused by malicious actors at a later time. The authentication scheme is also applied for oracle-to-oracle communication, which allows compromised servers to be blocked.

7.3 Summary

The first carried out evaluation dealt with the performance analysis of transactions, where latency and cost played a significant role. This was followed by a reflection of the addressment of problem statements and the justification on the fulfillment of goals. The resulting analysis shows that all initially defined objectives and problems were handled accordingly.
Subsequently, the implemented prototype was further discussed, where benefits and limitations were examined. To summarize the discussion, it was disclosed that the adoption of the approach comes with restrictions. Any company, who wants to join the ecosystem, has a limited degree of freedom, due to the compliance with software protocols and the recognition of uniform contract formats. Furthermore, the maintenance can be demanding, considering that a lot of Ethereum accounts, certificates and IDM servers have to be managed.

The overall results from the quantitative and qualitative analysis were aggregated and summarized in table 7.5. The criteria were classified into three categories: *performance, efficiency* and *security*. The second column indicates the degree of fulfillment of the corresponding criterion and consists of three levels: *fulfilled* (✓), *partially fulfilled* (○) and *not fulfilled* (✗).

The performance category includes metrics and quality indicators for time and memory. Availability is also listed under performance as it influences the com-

Criteria	
Performance	
Latency	✗
Scalability	✓
Availability	○
Efficiency	
Digitization	✓
Automation	✓
Cost	✗
Security	
Authentication & Verification	✓
Privacy	○

Table 7.5: Degree of Fulfillment of Evaluation Criteria

pletion of a task negatively if servers are not available. While the availability of the underlying Distributed Ledger Technology is guaranteed, the availability of off-chain servers cannot be made certain. The efficiency category is comprised of digitization, automation and cost. The cost criterion is considered as not fulfilled as the cost of transactions are extremely high. The last category, security, includes the authentication of users and verification of data, as well as the privacy of data. Privacy is interpreted as partially fulfilled, considering that certain meta information about smart contracts and user transactions can be used for profiling.

8. Conclusion

The final chapter of this thesis aims to summarize important findings and the proposed approach described in previous chapters. It will also discuss the addressment of research questions, reflect on important design decisions and introduce potential future work.

The primary challenge of this thesis work was to establish trusted data exchange in the supply chain. End customers and stakeholders should be able to get insight into the origins of data contributions, for example for the processing of shipped goods. Thus, the aggregation of data originating from various trusted sources became a crucial goal. Osterland and Rose propose a DLT-based approach, which addresses the issue of scalability by introducing a hybrid ecosystem consisting of on- and off-chain components. The objective of this thesis was to adapt, contribute to and finally apply this approach to the use case of inland waterway transportation.
The adapted approach introduced in chapter 5 is intended for the Ethereum Blockchain as the underlying DLT and stores smart contracts representing the *Charterparty, Bill of Lading* and *Proof of Delivery*. The `ENS` smart contract is used for resolving Ethereum addresses of human readable domains, while `DIDRegistry` is used to register service endpoints and assign delegacy rights to other Ethereum users. The `Oracle` smart contract is intended for communicating with the 'outside world' via Blockchain events. Hence, there are various off-chain oracle servers listening and reacting to events. As opposed to the oracle, the broker is mainly intended for submitting new data to the Blockchain. However, access to the broker component needs to be permitted to authorized users, which are authenticated by IDM servers beforehand. In order to avoid potential MitM attacks, IDM servers are accessed through the local network. The implemented authentication protocol reliably authenticates and forwards users to their respective brokers. Each company operates their own broker, oracle and database servers, in order to preserve the privacy of confidential data. The oracle-to-oracle communication protocol allows external stakeholders to systematically and trustfully exchange data.

The literature review in chapter 4 shows that there are existing Blockchain-based solutions for the supply chain. *Tradelens* uses a consortium Blockchain and offers a platform & APIs for its users to interact with the Blockchain. It is

operated by IBM and thus is not a self-sovereign ecosystem. Furthermore, the focus of Tradelens is mainly container shipment, which differs from bulk-load shipments. Similary, *CargoX* presents a Blockchain solution for tracking and trasferring the ownership of legal documents. However, it does not include the digitization of these documents or important tasks, such issuing a Proof of Delivery or certificate. The quantitative evaluation in chapter 7.1 shows that the Ethereum Blockchain is very volatile when it comes to the transaction-submission rate. At times, the network load is congested and the time or day cannot be predicted. The congestion impacts the overall transaction latency, given that the gas price increases and transactions with the highest prices are prioritized. In order to ensure timely mining, a high gas price needs to be used at all times, which leads to extremely high transaction costs. Nevertheless, transactions on the Ethereum Blockchain are not carried out near real-time, which is why latency is inevitable even if it is displeasing. Throughout the realization process of the prototype, it was discovered that the implementation of complex smart contracts is challenging, as the size of a contract is limited. The referencing of smart contracts in other contracts readily causes an increase in the overall bytecode-size. In order to be able to deploy complex contracts, a trade-off between contract size and gas consumption had to be made.

8.1 Research Questions

In order to evaluate whether the goals and problems were properly addressed, an assessment of the research questions will be conducted. The following examines each individual question and explains how it was addressed.

1. What kind of data needs to be aggregated in the use case of inland waterway transportation?

In order to answer this question, an online research about the business process was conducted and its result respectively reported in chapter 2. The process and the information passed in-between activities is laid out in figure 2.2. During the inland shipment process, the goods are handed over to the shipowner and then back to the port. The handover is documented with the *Bill of Lading*, while its return is documented with the *Proof of Delivery*. In addition to these documents, a *Certificate* claiming that the vessel in use is clean is also issued.

2. How can data, that comes from a multitude of independent sources, be reliably and trustfully aggregated?

2.1 How can data be shared efficiently and consistently between stakeholders?

The data exchange between external stakeholders is managed using the oracle-to-oracle communication protocol explained in chapter 5.5. The exchange is automatically executed after certain Blockchain events occur, which elimi-

nates the need to manually send emails and thus increases *efficiency*. Adhering to the predefined protocol and datastructure instates *consistency*.

2.2 How can the authenticity of shared data be established?

Access to the Blockchain should only be performed using the broker interface. It is intended to be accessed either from the local network (e.g. work place), or remotely using the Identity Management (IDM) server. The authentication protocol is strictly adhered by all broker systems, hence it is safe to assume that any data contribution was authenticated. The protocol is executed in 3 steps (see figure 5.5): (1) the local IDM server verifies the ownership of the users Ethereum account using a challenge-response protocol, (2) server forwards the Ethereum account to the corresponding broker, who checks the `DIDRegistry` for delegacy rights and (3) checks whether the user is saved in the database.

2.3 How can the integrity of shared data be verified?

The integrity of data can be verified using merkle trees. As explained in section 3.1.1, data is hashed & aggregated into a single hash, i.e. the merkle root. All shipment related smart contracts (Charterparty, B/L, etc.) store a single hash representing the root of the aggregated data. The advantage of merkle trees is that partially available data can be collectively verified using the corresponding tree. The trees are stored off-chain and can be queried at all times. This way, any stakeholder can first query the merkle tree, check whether its merkle root is the same as in the smart contract and then proceed with verifying the raw data using the tree.

3. How can the privacy of data be protected on a public DLT?

The use of merkle trees serve multiple purposes, one of which is the protection of data confidentiality by producing cryptographic hashes. As only the root (hash) is stored on the Blockchain, the raw data cannot be recovered.

4. How can the quality of grain be controlled?

4.1 Which factors affect the degradation of the quality?

The research conducted in chapter 2 reveals, that environmental conditions during the shipment, such as temperature or humidity, severely affect the moisture content in grain. Furthermore, if previous shipments in the same vessel contained chemicals or left dirt, it would taint the grain and thus make it worthless.

4.2 What measures need to be taken in order to preserve the quality?

In order to motivate the shipowner to comply with environmental and hygiene rules, a deposit strategy was developed. The shipowner needs to deposit a fee for potential damages before the goods are shipped. Once the shipment

arrives at the discharging port, it is examined by the port agent. If there were no damages, the deposit is automatically refunded.

5. How can an interoperability within the ecosystem be accomplished?

While the use of underlying server or database architectures do not have an impact on the ecosystem, it would nevertheless not function properly, if off-chain components would not be able to communicate with each other. Therefore, a set of data structures (e.g. smart contracts), and protocols (e.g. authentication & oracle-to-oracle communication) were developed. Any company, who intends to join the ecosystem is compelled to adhere to these restrictions.

8.2 Reflection

This section intends to reflect on the design & implementation decisions made throughout this thesis work and discuss possible improvements.
The public Ethereum Blockchain was selected as the underlying Distributed Ledger Technology, as it allows the incorporation of smart contracts, is very well documented and offers many tools & resources. However, the use of a public DLT comes with manifold limitations, some of which are the remarkably high transaction cost and latency. These factors will most likely deter businesses from adapting DLT-based solutions. Another limitation is the fact that transactions are public on the Ethereum Blockchain. The research paper by Béres et al. describes how public transactions can be used for profiling and thus deanonymizing Ethereum users [6]. The same technique could potentially be used to investigate which businesses in the supply chain collaborate with each other. One possibility to solve these limitations would be to use a consortium Blockchain, which are still public but only used for the supply chain. One such Blockchain is Quorum[1], which is a fork of the Ethereum Blockchain. It offers the possibility to make transactions or smart contracts private, thus eliminating the risk of profiling. Furthermore, the gas price can either be reduced or removed completely, thus cutting down costs significantly.

In order to protect the privacy of confidential data, it was decided to let businesses operate their own broker, oracle and database servers. This way, the company has control over access permissions. The use of the underlying server or database architecture is left to the company, in order to allow them to use technologies that they are familiar with or even incorporate existing servers/databases. Unfortunately, with this approach there is a possibility of certain servers not being available at times, e.g. due to outages. This could introduce unwanted delays in the shipping process.

The ecosystem requires that each port operates their own IDM server, in order to add an additional layer of security by allowing users to authenticate in the local

[1] https://consensys.net/quorum/

network. Port agents need to interact with multiple different brokers and the use of a single login portal would facilitate the access to all of them. Moreover, the use of an IDM implicitly guarantees all stakeholders that users have been properly authenticated.

One aspect that could be improved upon is the authentication protocol. The current implementation requires all users to be registered to brokers before they can interact with them. In reality though, it is not efficient since new actors join the shipment process all the time. A possible solution would be to rely on port agencies as certification authorities, who issue certificates to their agents. These certificates could be stored on a publicly distributed database such as swarm[2], and a reference to it could be made in the DID document of the agent. This way, businesses would not have to register external stakeholders, but instead verify the certificates referenced from their DID.
The incorporation of the *Ethereum Name Service, Decentralized Identifier, Verifiable Credential* and *JWT* was intentional, since these protocols are standardized and continuously improved upon. In the future, it would be easier to incorporate new elements into the ecosystem.

8.3 Future Work

The architecture design and implementation proposed in this thesis serves as a good starting point for bulk-loading. However, there are also possibilities for integrating IoT devices into the business process. For example, vessels could be equipped with temperature & humidity sensors that automatically supply data to a broker. Thus the shipping process could be accelerated by replacing manual tasks. Moreover, the port itself could equip the port crane and other devices with sensors.

Another possible extension would be to expand the proposed approach to include the complete supply chain rather than waterway transportation. For example, a farmer could use the ecosystem to provide information about the origin (location) of their goods. Additionally, organizations like *Fair Trade* could issue certificates (Verifiable Credentials) to farmers, who ethically produce their goods. This would essentially guarantee the authenticity of claims to all stakeholders in the supply chain process.

The current ecosystem is not self-sufficient as it relies on the availability of all off-chain components. Hence, the objective would be to re-engineer the architecture towards a more self-sovereign one. A starting point could be to store merkle trees on a publicly distributed database (e.g. swarm). Furthermore, public information related to Ethereum accounts could also be stored publicly in order to avoid registering users. Additionally, the public database could be used for storing confidential data in encrypted form, which would still protect it from the public.

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

References

[1] Abraham, Ittai et al. *Revisiting Fast Practical Byzantine Fault Tolerance*. 2017. arXiv: 1712.01367 [cs.DC].

[2] Akhtar, Z. "From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild". In: *2019 International Conference on Electrical, Electronics and Computer Engineering (UPCON)*. 2019, pp. 1–6. DOI: 10.1109/UPCON47278.2019.8980029.

[3] Back, Adam et al. "Enabling blockchain innovations with pegged sidechains". In: (2014). URL: http://kevinriggen.com/files/sidechains.pdf (visited on 03/28/2021).

[4] Benet, Juan. "IPFS - Content Addressed, Versioned, P2P File System(DRAFT 3)". In: (2014). URL: https://raw.githubusercontent.com/ipfs/papers/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf (visited on 03/11/2021).

[5] Buterin, Vitalik. "Merkling in Ethereum". In: (2015). URL: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/ (visited on 06/09/2021).

[6] Béres, Ferenc et al. *Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users*. 2020. arXiv: 2005.14051 [cs.CR].

[7] Do, H. G. and Ng, W. K. "Blockchain-Based System for Secure Data Storage with Private Keyword Search". In: *2017 IEEE World Congress on Services (SERVICES)*. 2017, pp. 90–93. DOI: 10.1109/SERVICES.2017.23.

[8] Goudz, Alexander and Ahmad, Inas. *Mögliche Anwendungsbereiche der Blockchain-Technologie im maritimen Transport*. 2020. DOI: 10.17185/duepublico/71592. URL: https://duepublico2.uni-due.de/receive/duepublico_mods_00071592.

[9] Hardt, Dick et al. *The OAuth 2.0 authorization framework*. 2012.

[10] Hepp, Thomas et al. "On-chain vs. off-chain storage for supply- and blockchain integration". In: *it - Information Technology* 60.5-6 (2018), pp. 283–291. DOI: doi:10.1515/itit-2018-0019.

[11] Hinckeldeyn, Johannes. "Blockchain-Technologie in der Supply Chain". In: *Einführung und Anwendungsbeispiele (essentials)* 5 (2019), p. 2019. DOI: 10.1007/978-3-658-26440-6.

[12] Jones, M. et al. "JSON Web Token (JWT)". In: (2015). ISSN: 2070-1721. URL: https://datatracker.ietf.org/doc/html/rfc7519 (visited on 07/27/2021).

[13] Kumar, R. et al. "Distributed Off-Chain Storage of Patient Diagnostic Reports in Healthcare System Using IPFS and Blockchain". In: *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*. 2020, pp. 1–5. DOI: 10.1109/COMSNETS48256.2020.9027313.

[14] McConaghy, Trent et al. "Bigchaindb: a scalable blockchain database". In: *white paper, BigChainDB* (2016).

[15] Mühlberger, Roman et al. "Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World". In: *Business Process Management: Blockchain and Robotic Process Automation Forum*. Ed. by Aleksandre Asatiani et al. Cham: Springer International Publishing, 2020, pp. 35–51. ISBN: 978-3-030-58779-6. DOI: 10.1007/978-3-030-58779-6_3.

[16] Müller, Hans-Martin. "Binnenschifffahrt". ger. In: *Handwörterbuch der Stadt- und Raumentwicklung*. urn:nbn:de:0156-5599228. ARL - Akademie für Raumforschung und Landesplanung, 2018, pp. 243–251. ISBN: 978-3-88838-559-9. URL: http://hdl.handle.net/10419/225684.

[17] Osterland, Thomas and Rose, Thomas. "Oracle-Based Process Automation in DLT Dominated Ecosystems with an Application to German Waterway Transportation". In: *2021 2nd Asia Service Sciences and Software Engineering Conference*. New York, NY, USA: Association for Computing Machinery, 2021, 130–136. ISBN: 9781450389082. URL: https://doi.org/10.1145/3456126.3456132.

[18] Philipp, Robert et al. "Blockchain and Smart Contracts for Entrepreneurial Collaboration in Maritime Supply Chains". In: *Transport and Telecommunication Journal*. Vol. 20. 4. Sciendo, 2019, pp. 365–378. DOI: 10.2478/ttj-2019-0030. URL: https://sciendo.com/article/10.2478/ttj-2019-0030.

[19] Rauchs, Michel et al. *Distributed Ledger Technology Systems: A Conceptual Framework*. 2019. DOI: 10.2139/ssrn.3230013.

[20] Reed, Drummond et al. "Decentralized Identifiers (DIDs) v1.0". In: (2021). URL: https://www.w3.org/TR/2021/CRD-did-core-20210609/ (visited on 06/09/2021).

[21] "Reshaping the Future of Global Trade with World's First Blockchain-based Bill of Lading - Business Overview and Technology Bluepaper". In: (2021). URL: https://cargox.info/files/CargoX-Business-Overview-Technology-Bluepaper.pdf (visited on 10/06/2021).

[22] Roeck, Dominik. "The foundation of distributed ledger technology for supply chain management". In: *Proceedings of the 53rd Hawaii International Conference on System Sciences*. 2020. DOI: 10.24251/HICSS.2020.553.

[23] "SINLOG – Distributed Ledger Technology (DLT) für die Binnenschifffahrt". In: (). URL: https://www.fit.fraunhofer.de/de/geschaeftsfelder/kooperationssysteme/blockchain/sinlog.html (visited on 10/05/2021).

[24] Sluijs, Christiaan. "Antwerp start-up T-Mining develops Blockchain solution for safe, efficient container release". In: (2017). URL: https://www.portofantwerp.com/en/news/antwerp-start-t-mining-develops-blockchain-solution-safe-efficient-container-release (visited on 02/11/2021).

[25] Sporny, Manu et al. "Verifiable Credentials Data Model 1.0". In: (2019). URL: https://www.w3.org/TR/2019/REC-vc-data-model-20191119/ (visited on 07/27/2021).

[26] Stahlbock, Robert et al. "Blockchain in der maritimen Logistik". In: *Blockchain : Grundlagen, Anwendungsszenarien und Nutzungspotenziale*. Ed. by Hans-Georg Fill and Andreas Meier. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, pp. 235–256. ISBN: 978-3-658-28006-2. DOI: 10.1007/978-3-658-28006-2_12. URL: https://doi.org/10.1007/978-3-658-28006-2_12.

[27] "TRADELENS - ÜBERBLICK". In: (2019). URL: https://www.ibm.com/downloads/cas/1ZVRWXPG (visited on 03/11/2021).

[28] Twenhöven, Thomas and Petersen, Moritz. "Impact and beneficiaries of blockchain in logistics". In: *Hamburg International Conference of Logistics (HICL) 2019*. Proceedings of the Hamburg International Conference of Logistics (HICL). 2019, pp. 443–468. ISBN: 978-3-750249-47-9. DOI: 10.15480/882.2479.

[29] "VeChain Whitepaper 2.0". In: (2019). URL: https://www.vechain.org/qfy-content/uploads/2020/01/VeChainWhitepaper_2.0_en.pdf (visited on 03/11/2021).

[30] Wang, Shuai et al. "An Overview of Smart Contract: Architecture, Applications, and Future Trends". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 108–113. DOI: 10.1109/IVS.2018.8500488.

[31] Wehrle, Klaus et al. "7. Distributed Hash Tables". In: *Peer-to-Peer Systems and Applications*. Ed. by Ralf Steinmetz and Klaus Wehrle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 79–93. ISBN: 978-3-540-32047-0. DOI: 10.1007/11530657_7.

[32] Wong, Rachel. "Transportation of wheat". In: (2014). URL: https://www.skuld.com/topics/cargo/solid-bulk/agricultural-cargoes/transportation-of-wheat/ (visited on 04/07/2021).

[33] Zhang, L. et al. "Poster: Towards Fully Distributed User Authentication with Blockchain". In: *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*. 2017, pp. 202–203. DOI: 10.1109/PAC.2017.28.

[34] Zhang, Lin et al. “Evaluation of Ethereum End-to-end Transaction Latency”. In: *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2021, pp. 1–5. DOI: 10.1109/NTMS49979.2021.9432676.

Appendices

A

Listing A1: Verifiable Credential [25]

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "http://example.edu/credentials/1872",
  "type": ["VerifiableCredential", "AlumniCredential"],
  "issuer": "https://example.edu/issuers/565049",
  "issuanceDate": "2010-01-01T19:73:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [{
        "value": "Example␣University",
        "lang": "en"
      }, {
        "value": "Exemple␣d'Universite",
        "lang": "fr"
      }]
    }
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-06-18T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "https://example.edu/issuers/keys
      /1",
    "jws": "eyJhbGciOiJSUI2U...PAYuNzVBAh4vGHSrQyHUdB"
  }
}
```

B

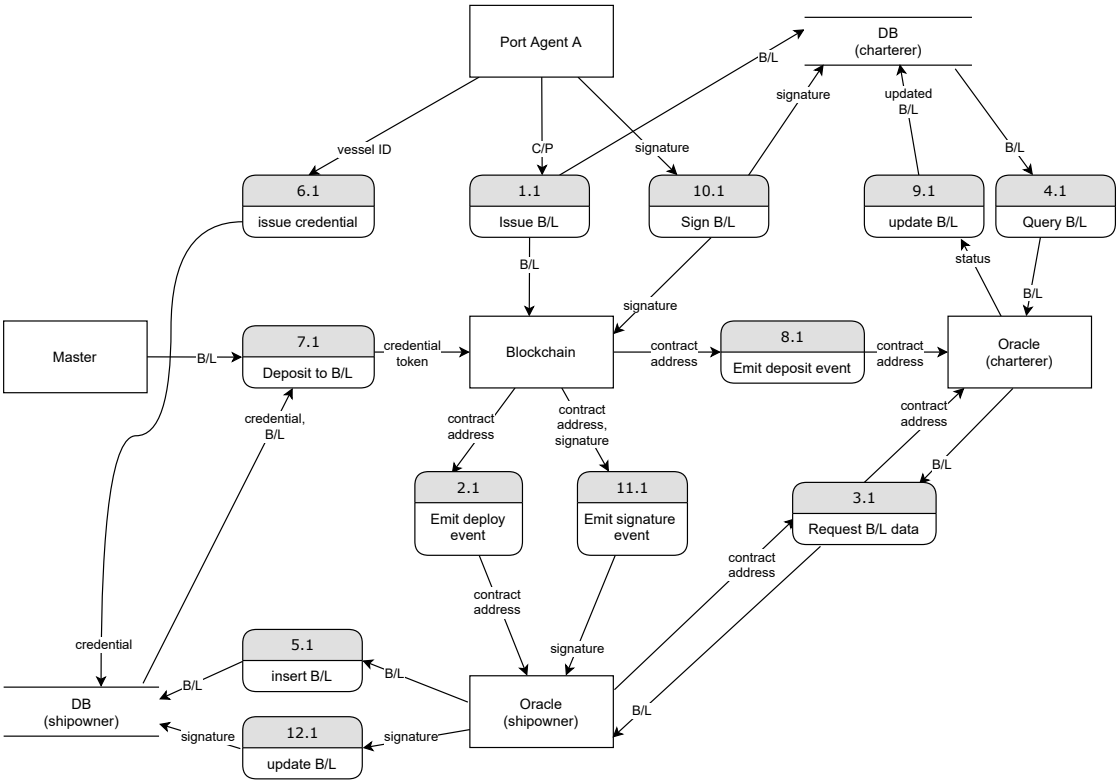

Figure B1: Data Flow Diagram - Processing B/L Level 1

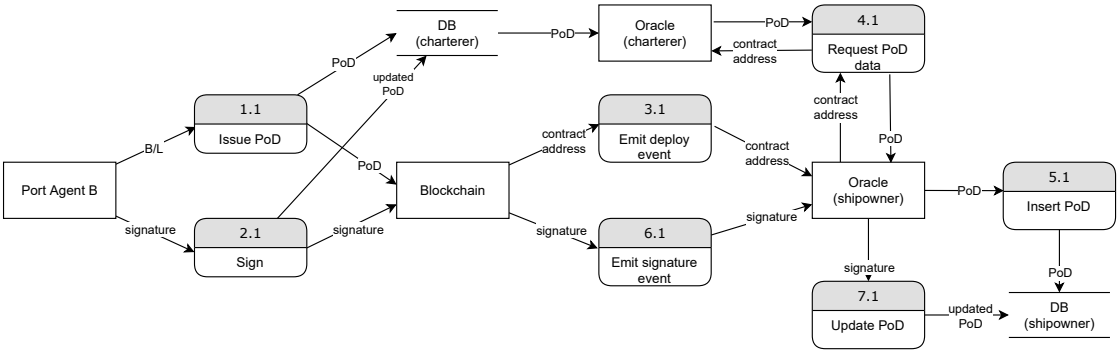

Figure B2: Data Flow Diagram - Processing PoD Level 1

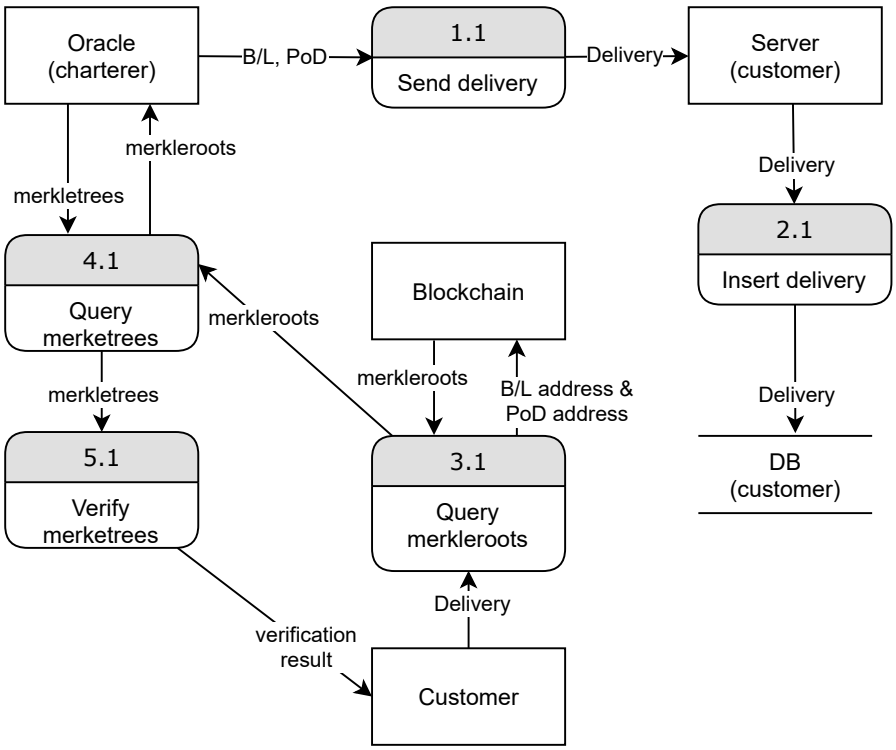

Figure B3: Data Flow Diagram - Verification Level 1

C

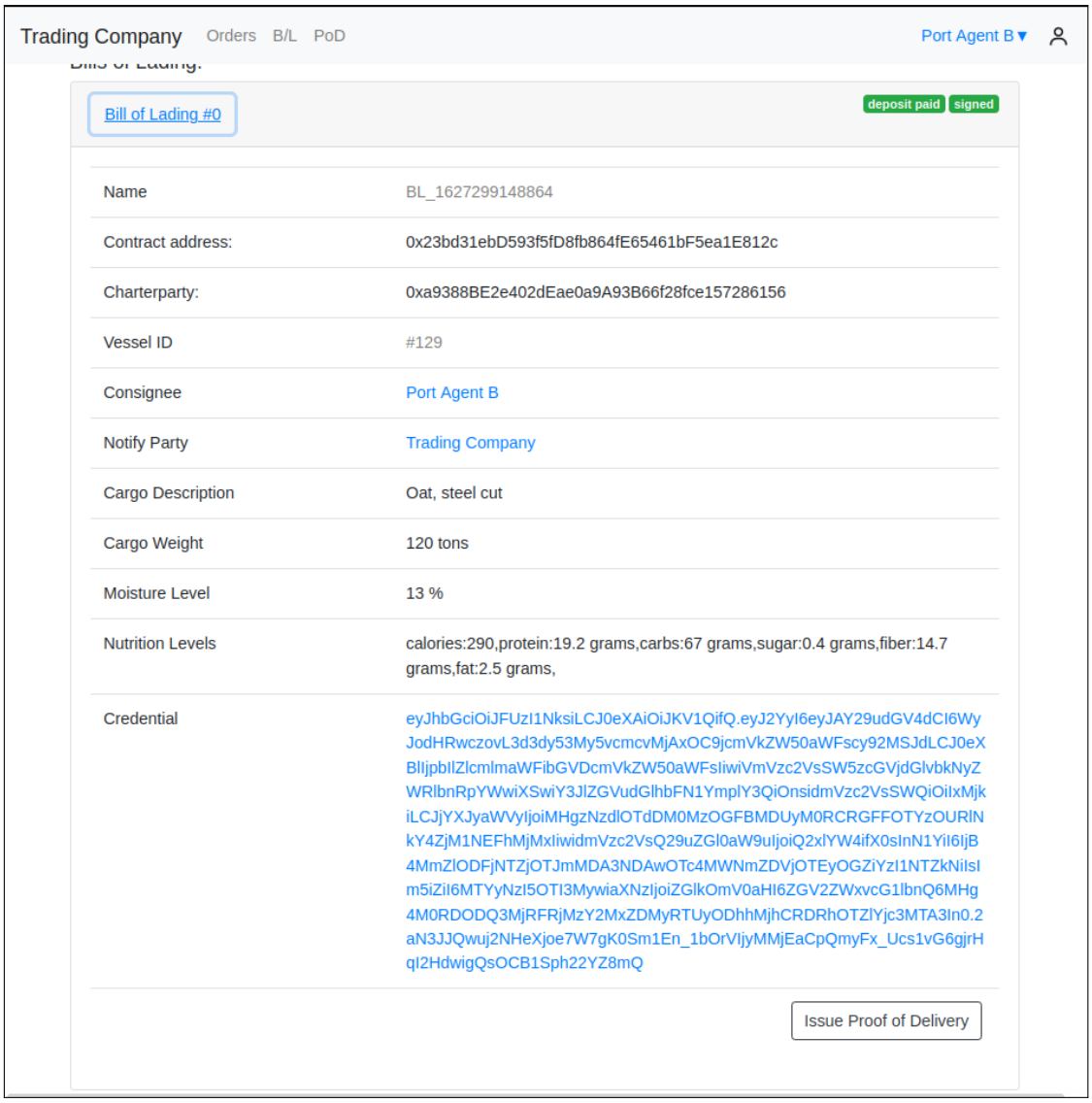

Figure C1: Screenshot of Before Issuing a Proof of Delivery

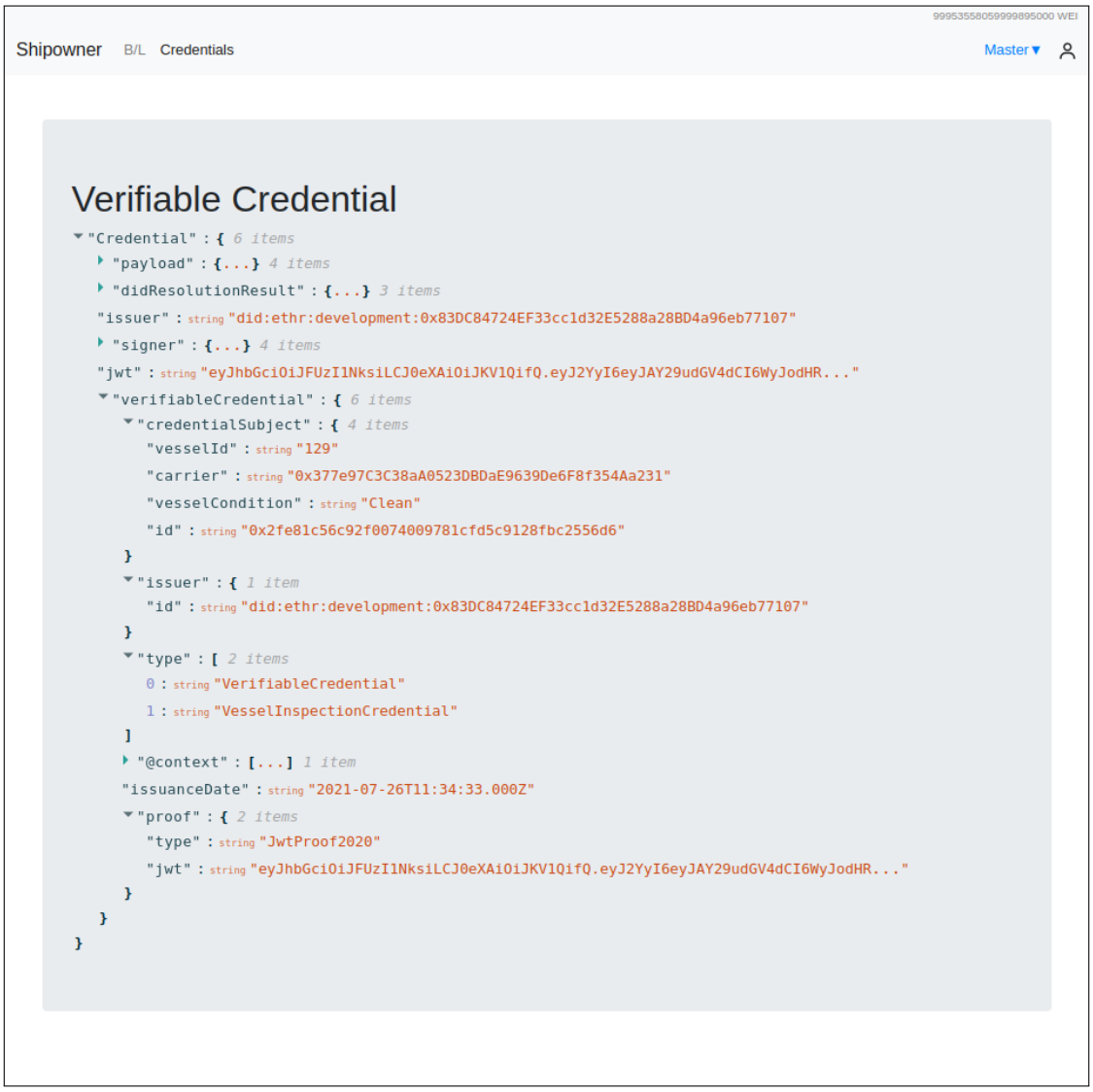

Figure C2: Screenshot of Verifiable Credential