Radboud University

Master Thesis

Demystification of the Blockchain Phenomenon: A Matter of State Management

Hyperledger Fabric and Hyperledger Sawtooth compared

Author: R.A.L. (Robert) Hissink Muller

Supervisor: dr. S.J.B.A. (Stijn) Hoppenbrouwers

Second Examiner: dr. ir. E. (Eelco) Herder

External Supervisor: prof. dr. A.F. (Frank) Harmsen

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in the Information Sciences
at the
Institute for Computing and Information Sciences

May 2019

In cooperation with PNA Group

“All that will remain of us is what is written down.”

— *Robert Harris (from Dictator)*

Radboud University

Abstract

Faculty of Science
Institute for Computing and Information Sciences

Master of Science in the Information Sciences

Demystification of the Blockchain Phenomenon: A Matter of State Management

Hyperledger Fabric and Hyperledger Sawtooth compared

By R.A.L. (Robert) Hissink Muller

Blockchain technology is entitled as the next disruptive revolution after the rise of the internet. Still lots of misunderstandings exist on this matter in addition to a lack of scientific treatment. When discussing the blockchain phenomenon, identification of the type of blockchain is of major importance since the goals per blockchain type will differ. In the case of private permissioned blockchains, the network consists of identified participants. These participants have common interests, but may not fully trust each other. Hyperledger Fabric and Hyperledger Sawtooth are examples of such private permissioned blockchains. By implementing a functionality to comprehend business logic, these platforms enable organisations to digitize assets. These digitized assets have a state associated with it. By submitting transactions to the network that are stored in a blockchain, the state of digitized assets can be managed. This results in trustworthy, trackable information on the state of a digital asset, which can be beneficial for all kinds of business operations. This thesis explains this process by comparing the transaction flow (the processing of transactions) of Hyperledger Fabric and Hyperledger Sawtooth, showing the differences and similarities between both platforms.

Acknowledgements

Although I am officially the author of this thesis, I could not have done it on my own. Therefore I want to thank Frank Harmsen and Stijn Hoppenbrouwers for the opportunities they gave me, their loyal support, supervision and their believe in a positive outcome throughout this whole process. I thank PNA Group and their employees for the creative inspiration that they brought me for writing this thesis. Furthermore, I thank Mats Ouborg for his lessons in modelling and support in general; Ruud Seegers for his support and the substantive discussions we had and still have about blockchain technology and Karel Kubat for the (technical) insights he gave me thanks to his profound knowledge on blockchain technology. Lastly, I thank my closest family and Cecilia van Berkum for their love, support and positive attitudes towards my graduation process.

Thank you, and all others, who helped me during the creation of this work!

Table of Contents

Chapter 1
Introduction

1.1 About this thesis

Blockchain technology is entitled as the next disruptive revolution after the rise of the internet and it can probably be seen as the mother of all IT hypes so far. However, there are lots of misunderstandings on this matter and there is a great need for a solid academic treatment of the subject. One can distinguish between the first generation of blockchain technology with Bitcoin as best known example. Bitcoin's main goal was to enable people to transfer value, without an intermediate third party needed. All data in Bitcoin is public and no member is known. Besides that, it takes heaps of energy to function. The most prominent representative of the second generation blockchain technology is called Ethereum. Ethereum introduces a major new functionality named smart contracts. In chapter 2, more information on these two technologies will be given.

The immutability of the blockchain is a feature that can be widely applied. However, in most of the vast application areas there is a need to (digitally) identify the members and have selective privacy. This has led to the third generation of blockchain technology and with that the rise of Hyperledger. Hyperledger is the largest open source project in the history of the Linux Foundation. It contains 8 subprojects, and two of them have reached the first official release Fabric (July 2017) and Sawtooth (January 2018). Fabric and Sawtooth have several features in common but have at the same time some unique features.

There is an increasing amount of work to be done to make blockchain fit for the enterprise. The challenge is how to get from the hype to the proof of concept and from there to production. One of the ways to support this transformation is to formalize the blockchain phenomenon via conceptual models. This lack of formalization is also addressed by de Kruijff & Weigand (2017):

"There is a lack of formalization and standardization of terminology for the blockchain phenomenon. Not only because there are several implementation platforms, but also because the academic literature so far is predominantly written from either a purely technical or an economic application perspective. For blockchain to be accepted as a technology standard in

established industries, it is pivotal that ordinary internet users and business executives have a basic yet fundamental understanding of the workings and impact of blockchain."

In this thesis the definition of a conceptual model given by Proper, Bjeković, van Gils & Hoppenbrouwers (2017) is followed, where a conceptual model is defined as being a model where its purpose involves a need to capture knowledge about the represented domain. In other words, a model answering a need to understand and/or articulate the workings and/or structure of some domain.

1.2 Research question

As stated in the introduction, there is a great need for an academic treatment of the phenomenon blockchain. Moreover, one of the responsibilities of the Technical Steering Committee within the Hyperledger Project is to create sub-committees or working groups to focus on cross-project technical issues or opportunities. This master thesis aims to contribute to that treatment and to scientific research on blockchain technology in general. Furthermore, it aims to make blockchain technology understandable for businesses. The insights gained from this research can be used to build services and products based on these platforms. Regarding the goals of this master thesis, a comparison is done between Hyperledger Fabric's and Hyperledger Sawtooth's most fundamental process, the processing of transactions or the so called transaction flow, in order to show relevant similarities and differences. The main research question (RQ1) can therefore be formulated as:

"To what extent do the transaction flows of Hyperledger Fabric and Hyperledger Sawtooth differ?"

To be able to answer this research question, the different consensus algorithms will be identified; the transaction flows and the corresponding transactions modelled, based on a tuna fishery use case; and the different data structures of the blocks presented. Sub-question that are answered are:

- RQ1.1: *Which consensus algorithms are used?*
- RQ1.2: *Which elements influence the transaction flow of the platform?*
- RQ1.3: *How are transactions and blocks structured?*
- RQ1.4: *What are the end results of the submitted transactions?*

1.3 Methodology

The goal of this thesis is to gain insights on the Hyperledger Fabric and Sawtooth blockchain projects; to treat the blockchain phenomenon in a scientific manner and to make the technologies understandable for businesses. To accomplish this, a comparative study is chosen as methodology for this research. By comparing the blockchain platforms with one another, differences and similarities between the two subjects can be found, explained and elaborated on. For this comparison, model-based description is used and then model-based analysis in order to systematically identify and describe the main aspects/concepts up for comparison, and then carry out this comparison, rendering a complete and optimally meaningful result reflecting both essential similarities and differences between the platforms.

As formal conceptual modelling method CogNIAM is used. CogNIAM is used because 1) it is vastly used in the organisation this research is conducted for and 2) it bounds the user to a structured and profound approach of any matter (see paragraph 1.4). The two platforms are discussed in separated chapters, although the chapters follow a similar paragraph structure based on fundamental design choices and a simplified tuna fishery supply chain use case. One of the results of this comparison is a meta-model, discussed in chapter 5, that covers the fundamental aspects of both platforms. Meta-modelling is the approach used to identify main concepts similar to both platforms, serving as a unifying level of description (Brinkkemper, 1990) and can be used, for example, to gain insight into information systems and the structure of their engineering methods (Harmsen, 1997).

The meta-model is filled in differently for either platform, systematically and clearly showing the differences between platforms. Furthermore, the meta-model leads to interrelationship models of both platforms that present the instances of the platform and their interrelationships.

1.4 The CogNIAM method

CogNIAM is a method that works under the assumption that most of the communication between members of one or more communities about a specific domain can best be taken as a starting point. Such communication is in the form of declarative facts and such declarative facts can have any number of instantiated placeholders or variables. Therefore CogNIAM assumes that the most often form of communication used is the exchange of declarative n-ary facts (Meijer, Nijssen & Bulles, 2017). Since the goal of this thesis is to compare two blockchain

projects, yet in such a way that business executives have a fundamental understanding of the working and impact of blockchain, CogNIAM is used in this thesis as the basis of Fact Based Modeling. A domain specific model in CogNIAM consists of (Meijer et al., 2017):

1. Fact Types,
2. Object Types (nominalized Fact Types) and subtypes,
3. The Integrity Rules on the Fact Populations and Fact Population Transitions,
4. The Derivation Rules, that produce new Fact Instances from available Fact Populations,
5. Exchange Rules to add, remove, or modify (in case of combined facts) facts in the Fact Populations,
6. Event Rules to either start a Derivation Rule or an Exchange Rule,
7. Fact Communication Patterns to make sub community communication well understood, while maintaining the overall integrity of the system,
8. Rule Communication Patterns to express rules in a community preferred format and
9. Definitions for all terms to foster the highest degree on understand-ability among the various members of the sub communities.

Besides Fact Based Modeling, the CogNIAM method makes use of BPMN to provide the entire conceptual model in the form of a process, since most people prefer to think of a system as a process (Meijer et al., 2017). Business Process Model and Notation (BPMN) is the de facto standard for representing, in a very expressive graphical way, the processes occurring in virtually every kind of organisation one can think of. It's a modeling language for business processes that is formal enough but easily understandable also by final users and not only by domain experts (Chinosi & Trombetta, 2012).

1.5 Content of this thesis

The content of this thesis is structured as follows. In chapter 2 some literature on blockchain is discussed. Where background information, the business potential and different point of views are given. Chapter 3 discusses the design choices of Hyperledger Fabric and presents the tuna fishery supply chain use case via conceptual models, chapter 4 does the same but applied to Sawtooth. In chapter 5 the results of chapter 3 and 4 are analyzed by presenting a meta-model that is then filled in with instances for both platforms showing the differences and similarities between both projects. In chapter 6 the conclusion is given and lastly, in chapter 7, the work of this thesis is discussed and future research directions are presented.

Chapter 2
Literature

2.1 Blockchain

A blockchain can be seen as a distributed database with high redundancy. It is replicated using a peer-to-peer system and every participating node keeps a full copy (Nakamoto, 2008). The major benefits of a blockchain are that it provides highly reliable, non-mutable and trustworthy storage of records without relying on a third party. However, in contrast to a database, a blockchain stores no tables but transactions. In general these transactions transfer an asset from one individual to another, where the individual does not need to be human, and the asset can be of any kind (Hoops, 2017). The transactions are grouped up in blocks. These blocks can be seen as a grouping of transactions, marked with a timestamp, and a fingerprint of the previous block. A blockchain is therefore defined as a list of validated blocks, each linking to its predecessor all the way to the genesis block. Where the genesis block is defined as the first block in the blockchain (Antonopoulos, 2014).

A blockchain solves the basic problem of distributed consensus. This means that all nodes in the network agree on the newest block added to the chain and therefore implicitly agree on one chain of blocks. To achieve consensus, some form of voting is necessary among these nodes. This form of voting is called a consensus mechanism/algorithm. Most of the ledger's characteristics (e.g. transaction throughput and supported number of nodes) depend on this mechanism, therefore it can be seen as the heart of every distributed ledger technology and as its most defining feature (Hoops, 2017).

The past few years enterprises have introduced the term distributed ledger to describe a blockchain process. This distinction has been made to distinguish blockchain enterprise solutions from the public blockchain sphere and their cryptocurrencies. However, in this thesis the terms distributed ledger and blockchain will be used interchangeably and will describe the same phenomenon. This is also in accordance to the Hyperledger Sawtooth documentation (Hyperledger Sawtooth, 2019) where no clear distinction between these two terms is made.

2.2 Smart Contracts

With the rise of Ethereum, Smart Contracts were introduced, which extended the use of blockchains immensely, providing the basis of the hype that was to follow. However, just as with Blockchain itself, a lot of mystique surrounds this new phenomenon, which can also be derived from the wide array of definitions being used for it. The different definitions usually fall into one of following two categories (Stark, 2016): 1) the term is used to identify technology, as in code that is stored, verified and executed on a blockchain, referring to "smart contract code" ; 2) the term is used for a specific application of that technology as a complement, or substitute, for legal contracts, referring to "smart legal contracts".

Since the introduction of Ethereum, blockchains can run code and because this code is run on a blockchain, they have unique characteristics compared to other types of software. Stark (2016) identifies three main characteristics as follows. Firstly, the program itself is recorded on the blockchain, which gives it a blockchain's characteristic permanence and censorship resistance. Secondly, the program can itself control blockchain assets. Thirdly, the program is executed by the blockchain, meaning it will always execute as written and no one can interfere with its operation.

To people working with blockchain technology, the term "smart contracts" is often used to refer to this blockchain code, as can also be seen by the definition Ethereum (2017) uses: "*Applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference*".

Calling these programs contracts does not mean they need to resemble a contract as we know it. For example, "if Alice has voted yes, add Bob to the list of members in group Y" could be a form of a Smart Contract. In many cases smart contract code is not used in isolation but as a small piece in a larger application (Stark, 2016).

This also explains why the Fabric community introduced and uses a distinctive term called chaincode: "*Hyperledger Fabric leverages container technology to host smart contracts called "chaincode" that comprise the application logic of the system. A chaincode typically handles business logic agreed to by members of the network, so it may be considered as a "smart contrac*t" (Hyperledger Fabric, 2018).

Within the scope of this thesis, when discussing smart contracts, the smart contract code is meant, unless otherwise mentioned.

2.3 The business potential

It is not within the scope of this thesis to discuss all potential impacts blockchain technology could have, but only to provide a mere summary of the possibilities that come with this emerging technology. As pointed out by Koens & Poll (2018), blockchain is not the only technology that provides a solution to some of the cases outlined in this thesis. However, it cannot be denied that blockchain has some advantages over these already existing technologies, in addition to that it creates new value propositions for enterprises and governments (Tapscott & Tapscott, 2016).

2.3.1 Public versus private blockchains

There is a great debate on the potential disruptive effects of blockchain. In order to understand this debate, it is important to know where this technology is coming from.

As a reaction to the failing financial system which resulted in a worldwide monetary crisis in 2008, Bitcoin was introduced as the first peer-to-peer network, enabling people to take full control over their money, aimed at making trusted third parties redundant and in particular the financial system. This intention can also be deduced via the hidden message within the genesis block of the Bitcoin network "*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*". This message was embedded by Satoshi Nakamoto, bitcoin's creator (Antonopoulos, 2014). The Bitcoin network is available for anyone who wants to participate, without giving away their identity.

Years later in 2014, with the introduction of the Ethereum network and the ability of smart contracts, large corporation started to get interested in the blockchain space. This interest was not aimed at giving more power to the people and making trusted third parties redundant. It was aimed at creating trust between different organisations by securing the interactions among a group of entities, that share a common goal but which may not fully trust each other. With the interest of the large corporations, a new type of blockchain emerged called the private blockchain whereas the Ethereum and Bitcoin blockchains, from that moment on, would be referred to as public ones.

Nowadays, three different types of blockchains can be distinguished based on publicity of the data and the participation rules of the network (Koens & Poll, 2018, Androulaki et al., 2018). These can be described as follows:

1. *Public permissionless blockchain*. Virtually anyone can participate, and every participant is anonymous. In such a context, there can be no trust other than that the state of the blockchain, prior to a certain depth, is immutable. In order to mitigate this absence of trust, permissionless blockchains typically employ a "mined" native cryptocurrency or transaction fees to provide economic incentive to offset the extraordinary costs of participating in a form of byzantine fault tolerant consensus.

2. *Public permissioned blockchain*. A limited set of participants may write to the blockchain. Anyone may join the network and read the state.

3. *Private permissioned blockchain*. A set of known, identified and often vetted participants operating under a governance model that yields a certain degree of trust. A private permissioned blockchain provides a way to secure the interactions among a group of entities that have a common goal but which may not fully trust each other. By relying on the identities of the participants, a permissioned blockchain can use more traditional crash fault tolerant (CFT) or byzantine fault tolerant (BFT) consensus protocols that do not require mining.

It is very important to keep the different goals of each blockchain type in mind when discussing the blockchain phenomenon. Most blockchain enthusiasts are discussing a decentralized world where the trusted third parties can be omitted and the large corporations will lose significant power. This reasoning can clearly be grouped under the *public permissionless blockchain* goals. However, when organisations are enthusiastic about blockchain, they tend to focus on the inhouse effects of easy audit trails and the creation of trust among the organisations they work with. This sphere can be grouped under the private permissioned blockchains. To further distantiate with the public permissionless blockchain, the different term "distributed ledger" was introduced.

2.3.2 Myths around blockchain

Due to the hype circulating around blockchain, a lot of enterprises (and even individuals) have the fear of missing out, also called FOMO. For example, a lot of financial institutes, afraid of being disrupted by this new technology, are putting huge amount of resources on blockchain

innovations (Swartz, 2017). There are also companies joining the bandwagon only to improve their current stock value in an instance. Long Island Iced Tea announced it was changing its name to Long Blockchain Corp., what followed was a quickly reward with a 200 percent jump in stock price at the opening of trading. However, the company's relation to blockchain remains dubious. (Liao, 2017).

Because of all this, it stays a difficult task for managers, executives and other types of business people to distinguish the true potential of this emerging technology from the hoaxes. In a recent report of McKinsey (2018) an overview is shown about the myths surrounding blockchain. This overview (table 2.1) helps to distinguish the "what is" and "what is not" regarding blockchain technology.

Myth	Reality
1) Blockchain is Bitcoin	• Bitcoin is just one cryptocurrency application of blockchain. • Blockchain technology can be used and configured for many other applications.
2) Blockchain is better than traditional databases	• Blockchain's advantages come with significant technical trade-offs that means traditional databases often still perform better. • Blockchain is particularly valuable in low-trust environments where participants can't trade directly or lack an intermediary
3) Blockchain is immutable or tamper-proof	• Blockchain data structure is append only, so data can't be removed • Blockchain could be tampered with if > 50% of the network-computing power is controlled and all previous transactions are rewritten. Which is largely impractical
4) Blockchain is 100% secure	• Blockchain uses immutable data structures, such as protected cryptography • Overall blockchain system security depends on the adjacent applications. Which have been attack and breached
5) Blockchain is a "truth machine"	• Blockchain can verify all transactions and data entirely contained on and native to blockchain (e.g. Bitcoin) • Blockchain cannot assess whether an external input is accurate or truthful. This applies to all off-chain assets and data digitally represented on blockchain.

Table 2.1: Overview of the myths versus reality regarding blockchain (McKinsey, 2018)

Table 2.1 shows that a lot of misunderstanding exists on this new technology. Therefore, the question where the real potential value of blockchain lies is a legitimate one. This will be discussed in the next paragraph.

2.3.3 The real value

As seen by the myths around blockchain, there is a lot of misunderstanding about the technological concepts. This also leads to a lesser understanding of the real value blockchain can bring to businesses. Based on McKinsey (2018), a few things can be said about the strategic value of blockchain technologies: *blockchain's strategic value is mainly in cost reduction*; *blockchain does not need to be a disintermediator to generate value* and *common standards are essential.*

Blockchain technology can solve the need for an entity to be in charge of managing, storing, and funding a database, therefore it can become a new open-standard protocol for trusted records, identity, and transactions. Cost can be taken out of existing processes by removing intermediaries or the administrative effort of record keeping and transaction reconciliation. This can shift the flow of value by capturing lost revenues and creating new revenues for blockchain-service providers (McKinsey, 2018). These claims are also supported by other researchers, such as Iansiti & Lakhani (2017) and Swan (2015). In particular the non-mutable and trustworthy storage of records, as previously described by Hoops (2017), creates a shift in the value propositions for a lot of industries. In particular industries where the sharing of data and consumer goods are essential. A prime example of a use case where all the advantages of a blockchain can be found is in supply chain industries.

A lack of trust can be a problem in a supply chain and may contribute to increase the probability of occurrence of the bullwhip effect (Lee, Padmanabhan & Whang, 1997). An information distortion in the chain which is propagated from downstream to upstream members. Relationships based on trust can prevent this (Handfield & Bechtel, 2002). A shared, consensus-based and immutable ledger helps track the origin and the transformations undergone in the supply chain (Pilkington, 2016) as well as eliminating the need for intermediaries, provides trust among the actors and traceability over the business processes (Guerreiro, Silva & Sousa, 2018). This can be beneficial for business to business services as well as for the consumer, because both are now able to validate the origin of a product. Moreover, the distributed knowledge, leading to all actors viewing the same events that occurred in the supply chain, is fundamental to handle new or unexpected situations (Guerreiro, et al. 2018).

Another use case can be identified within the healthcare industry, where the availability and sharing of data among patients, insurers, researchers and pharmaceutical companies, can be controlled via blockchain technology. Blockchain based healthcare records can not only facilitate increased administrative efficiency, but also give researchers access to the historical,

non–patient-identifiable data sets. Moreover, patients could gain more control over their data in a blockchain environment and, via Smart Contract functionalities, charge pharmaceutical companies to access their medical data (McKinsey, 2018).

In order to improve record keeping, assets must be able to be digitized. McKinsey (2018) calls this the digitization potential of the asset. Assets like equities, which are digitally recorded and transacted, can be simply managed. For physical goods like real estate or grocery store products, other solutions must be found in order to use within a blockchain context. An example of such a solution will be presented via the tuna fishery use case in chapter 3 and 4 of this thesis.

2.4 The Hyperledger project

Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation. One of the main goals of the Hyperledger project is to create enterprise grade, open source, distributed ledger frameworks and code bases to support business transactions.

Hyperledger launched in 2016. Initially, the Hyperledger Technical Steering Committee welcomed two business blockchain framework codebases into incubation: Hyperledger Fabric, a codebase combining work by Digital Asset, libconsensus from Blockstream and OpenBlockchain from IBM; and Hyperledger Sawtooth, developed at Intel's incubation group (Hyperledger, 2018). Hyperledger's executive director Behlendorf (2016) said the following about Hyperledger's purpose:

> "Another way to think of this is to consider Hyperledger's potential role in the emerging landscape of public blockchain technologies. Most of today's open source blockchain efforts outside of Hyperledger are focused on permissionless chains, necessarily implementing a cryptocurrency as a means to fund mining and participation in consensus. This has tremendous challenges, and not all of them are technical [...] What may appear to be technical debates at first glance often are really about different visions for the roles these platforms should play in society and who should govern them. We've crossed this bridge before, though. The Domain Name System was fortunate enough to rise to ubiquity long before anyone outside the early Internet architects realized how important it was [...] If Hyperledger can forge a brand that is widely seen as the accepted default 'safe' deployment platform for enterprise

> teams, and be seen as a great home for active collaboration around new technologies, then I think we can say 'mission accomplished."

The Hyperledger project can thus be seen as an umbrella project, hosted by The Linux Foundation, and supported by major enterprises such as IBM and Intel, that consists of and governs multiple Distributed Ledger Technology projects, all focused on DLT enterprise solutions. From these projects, Hyperledger Sawtooth and Fabric will be discussed in this thesis, with a main focus on their transaction flow.

2.4.1 Hyperledger Fabric

The content in this paragraph about the Hyperledger Fabric project is mainly based on Androulaki et al. (2018) in combination with the official open source documentation on Hyperledger Fabric (2019) and is meant as an introduction to Hyperledger Fabric. More in depth information will be discussed in chapter 3.

Fabric is a modular and extensible open-source system for deploying and operating permissioned blockchains, aiming at resiliency, flexibility, scalability, and confidentiality. It is the first truly extensible blockchain system for running distributed applications. It supports modular consensus protocols which allows the system to be tailored to particular use cases and trust models. Fabric is also the first blockchain system that runs distributed applications written in standard, general-purpose programming languages, without systemic dependency on a native cryptocurrency. The permissioned model is realized by using a notion of membership (Androulaki et al., 2018).

A distinctive feature of the Fabric architecture, compared to the order-execute design of traditional blockchains, is that it follows an execute-order-validate paradigm for the processing of transactions. This will be further discussed in chapter 3.

Another feature is the use of passive replication, this means that every transaction is executed (endorsed) only by a subset of the peers/nodes. A flexible endorsement policy specifies which peers (and how many) need to vouch for the correct execution of a given smart contract/chaincode. Fabric also uses active replication in the sense that the transaction's effects on state are only written after reaching consensus on a total order among them, in the deterministic validation step executed by each peer individually. The ordering of transactions is delegated to a modular component (the orderer) for consensus, which is stateless and logically decoupled from the client peers and the (non-)endorsing peer that execute transactions and

maintain the ledger. A Fabric network actually supports multiple blockchains connected to the same orderer, each of these connected blockchains are called a channel and may have different peers as its members, including different levels of access. The total order of transactions in each channel is separated from the others. However, since the main focus of this thesis is on the transaction flow level within one blockchain, this unique feature falls outside the scope and will therefore not be further discussed.

It is important to mention that due to the modular design of Fabric, not every deployment of the Fabric network behaves in the same manner (this also applies to Sawtooth networks). In this thesis, the production default settings will be used for comparison. More on this can be read in chapter 3 and 4.

2.4.2 Hyperledger Sawtooth

The content in this paragraph is mainly based on Hoops (2017) and the official open source documentation on Hyperledger Sawtooth (2019) and is meant as an introduction to Hyperledger Sawtooth. More in depth information will be discussed in chapter 4.

Sawtooth is an enterprise blockchain platform for building distributed ledger applications and networks. Sawtooth aims to be highly modular which should enable enterprises to make policy decisions best suited for their specific needs. The overall architecture of Sawtooth is quite similar to Bitcoin's, in the sense that it uses the traditional order-execute design. However, Sawtooth comes with a lot of additional features. Nodes in a Sawtooth network fill the roles of clients and/or validators. Where clients are individuals interacting with the network by querying the system or issuing transactions and validators are nodes involved in the consensus algorithm and can be used for clients to request inclusion of transactions into the current blockchain. This will be further discussed in chapter 4. These validators use the consensus algorithm Proof of Elapsed time (PoET) to agree on the information within the network (see paragraph 4.2.2.). Although Sawtooth supports other consensus mechanisms, PoET is the most prominent one.

One of the other distinctive features of Sawtooth is the separation of the Application level and the core system, therefore allowing developers to write smart contract logic in a language of their choice. There is even a Sawtooth-Ethereum integration project, called Seth, which allows Ethereum smart contracts to be deployed to Sawtooth.

The most distinctive Sawtooth feature is the concept of batches. A batch groups up one or more transactions of any type. Batches do not replace blocks, but can be seen as a new

component conceptually located in between transactions and blocks. A batch can either be accepted or declined in all of its entirety, but never can only a part of the transaction in a batch be accepted. This also means that even with one single transaction, a batch has to be created.

Chapter 3
Hyperledger Fabric

3.1 Introduction

This chapter contains the conceptual models for Hyperledger Fabric following the CogNIAM method. The models are based on the processes of a simplified fictional tuna fishery supply chain use case, since that is the default use case the Hyperledger community uses when discussing their different platforms in their documentation and online courses (e.g. edX.org). By putting the supply chain of tuna fishery on a blockchain, the traceability of tuna is enhanced, which makes it easier for processing plants and restaurant owners to track the origin of their product. This way illegal, unreported and unregulated fishing could be reduced and even eliminated (Degnarain, 2017, Waughray, 2017).

The same processes will be modelled for Fabric and Sawtooth to be able to compare both platforms adequately. Therefore, the actors in this use case will be the same for both platforms. These actors are as follows: 1) a fisherman that catches the tuna; 2) a tuna processing plant that processes the tuna; 3) a restaurant that will serve the tuna. Of course, this use case can be applied to many different types of products and processes (e.g. enhancing the traceability of fair trade products), but that is not within the scope of this thesis and would not contribute more to the comparison of the two platforms, therefore only the tuna fishery supply chain is modelled.

In the following paragraphs, the different processes of the tuna fishery supply chain are modelled in BPMN. In addition to that, the different transactions and produced blocks are modelled in Fact type diagrams (FTD's) accordingly.

3.2 Change of Paradigm

In order to improve the understanding of the different processes that are modelled in this chapter, some of the (earlier discussed) concepts about blockchains need to be clarified in the context of Hyperledger and in this chapter for Fabric in particular, although some of the concepts apply to Sawtooth as well.

3.2.1 World State

Bitcoin does not make use of a database and the current state of the chain is always calculated by going through all of the transactions in the blockchain. However, within Fabric, another approach is chosen (Blockgeeks, 2018).

A ledger in Fabric consists of two distinct components, namely the world state and a blockchain. The world state is a database that represents the latest (current) values for all keys included in the blockchain. By default these states are expressed as key-value pairs. The blockchain however, is a transaction log that records all the changes that determine this world state. This is illustrated by figure 3.1 (Hyperledger Fabric Documentation, 2019).

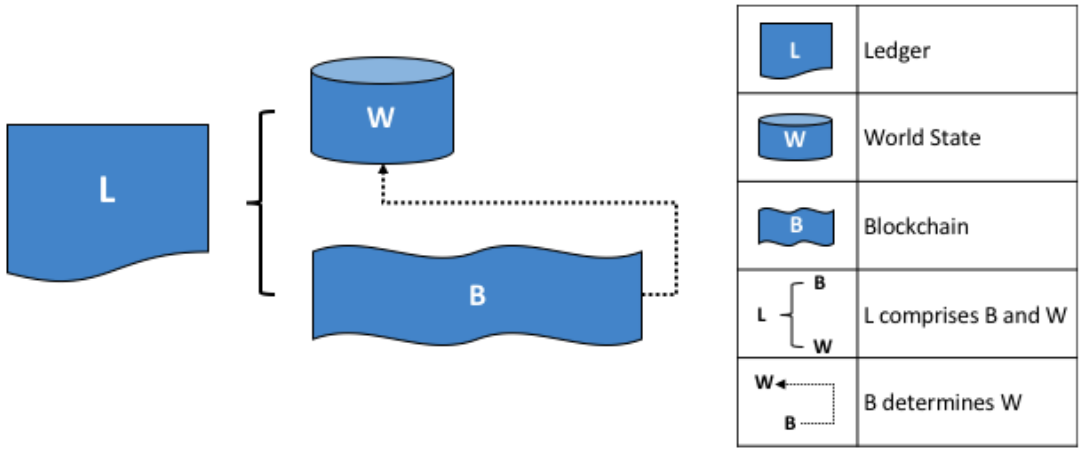


Figure 3.1: Ledger L comprises blockchain B and World State W. Blockchain B determines World State W. Also expressed as: World state W is derived from blockchain B.

As can be seen in figure 3.1, the world state is derived from the blockchain. Due to the immutable sequence of blocks, the blockchain is able to provide an untampered history of changes that have resulted in the current world state.

This setup is useful because applications usually need the current value of a ledger state and that current value can be easily retrieved by querying the database instead of calculating the state by traversing the entire blockchain.

3.2.2 Transactions and the digitization of assets

The digitization potential of assets is briefly mentioned in chapter 2. Carson et al. (2018) stated that in order to improve record keeping, assets must be able to be digitized. Hyperledger Fabric offers this digitization of assets.

Assets can range from the tangible (e.g. real estate and food) to the intangible (e.g. currencies and contracts) and are represented in Hyperledger Fabric as a collection of key-value pairs, with state changes recorded as transactions on a ledger (Hyperledger Fabric, 2019). Transactions are thus responsible for creating, altering and deleting assets, which are captured in the blockchain and represented in the world state as key-value pairs. As an example, figure 3.2 illustrates the creation of multiple cars as assets in the Hyperledger Fabric. This example is copied from the official Hyperledger Fabric Documentation (2019) and is used for its clear representation of a key-value pair in a blockchain context.

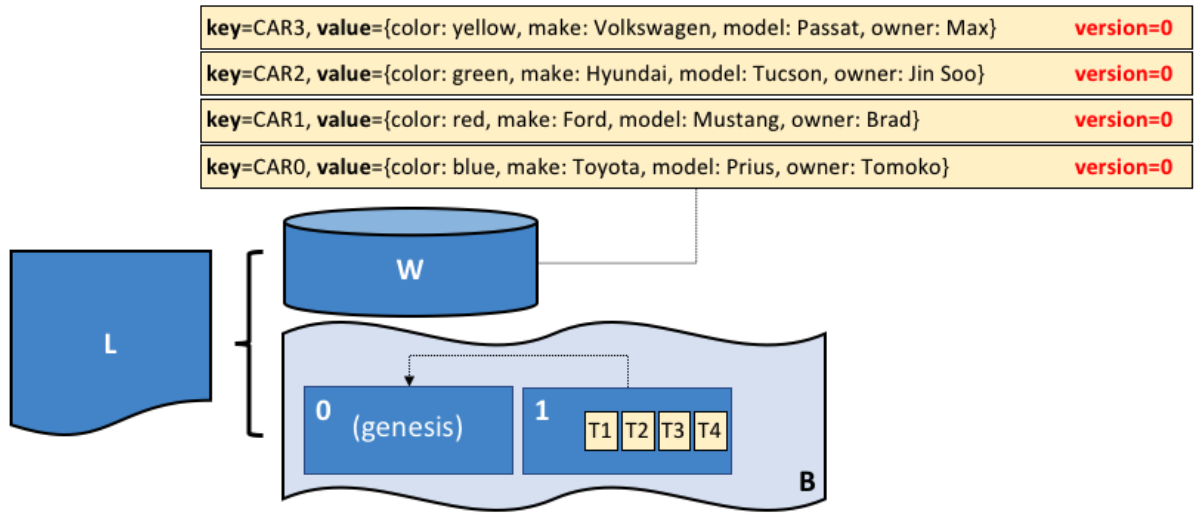


Figure 3.2: The ledger L, comprises a world state, W and a blockchain, B. W contains four states with keys: CAR1, CAR2, CAR3 and CAR4. B contains two blocks, 0 and 1. Block 1 contains four transactions: T1, T2, T3, T4.

As one can see, the blockchain in this ledger contains two blocks, where block 1 consists of four transactions (T1, T2, T3 and T4) representing four different cars (CAR0, CAR1, CAR2, CAR3). For instance, transaction 2 (T2) shows that CAR1 (the key) is a Red Ford Mustang owned by Brad (the value) and that the version number of this key-value pair is 0, indicating that the asset (key-value pair) is newly created.

In the following scenario where Brad's car is up for sale and John decides to buy his car, a new transaction (T5) will be sent to the network. T5 would then show that CAR1 is a Red Ford Mustang owned by John with version number 1, since the version number of a state is incremented every time the state changes (Hyperledger Fabric Documentation, 2019), which is in this case a change of ownership from Brad to John. Of course the world state will only be updated after T5 is approved as a valid transaction and added in a new block to the blockchain. The following paragraphs elaborate on that process within Fabric.

3.2.3 Consensus as a process

In chapter 2, the topic of distributed consensus was shortly discussed and defined as "*the agreement by all nodes on the newest block added to the chain, where a consensus mechanism/algorithm is necessary among these nodes to achieve this consensus.*" Fabric radically disregards this idea, stating that "*consensus has recently become synonymous with a specific algorithm, within a single function."* Instead, Fabric argues that "*consensus encompasses more than simply agreeing upon the order of transactions, it is defined as the full-circle verification of the correctness of a set of transactions comprising a block."* (Hyperledger Fabric Documentation, 2019).

Within the context of Fabric, consensus is achieved when the order and results of a block's transactions have met the explicit policy criteria checks. These checks take place during the lifecycle of a transaction (which is explained in more detail in paragraph 3.2.4). Consensus is therefore viewed as the byproduct of the ongoing verifications that take place during a transaction's lifecycle from proposal to commitment.

Taking this into consideration, Fabric still uses a (pluggable) protocol to order incoming transactions and submit them to the correct blocks. Fabric's out-of-the-box production solution for this makes use of Kafka. A distributed open-source stream-processing platform from Apache. Even though Kafka is sometimes seen as the consensus algorithm that is used within Fabric (as Hyperledger Architecture, Volume 1 (2017) also frames it), it is merely a service to order transactions and put them into blocks and cannot be compared to a nakamoto-style consensus algorithm, where different nodes compete each other to add the next block to the blockchain. This also means that by using Kafka, the network is crash-fault-tolerant (it can cope with failing nodes) but not Byzantine-fault-tolerant (it cannot cope with malicious nodes) (Friebe, 2017). Being unable to cope with malicious nodes is not necessarily bad in the context of a private permissioned blockchain, where all the participants of the network are already

known beforehand thus lowering the chance of malicious intents of nodes, but it does mean that the ordering service is prone to attacks (and failure) which can immediately affect the whole network. Moreover, this single point of failure design goes against the traditional philosophy of blockchain platforms, where an attack on one of the nodes should not influence the whole network's behaviour.

Note that Kafka is the default production solution, but due to Fabric's modular design it can be replaced by other type of protocols. When Kafka is replaced (or other protocols are added) it, of course, means that the before mentioned arguments do not apply anymore.

3.2.4 Execute-order-validate design

Most blockchain platforms, and especially the traditional ones (e.g. as seen in Bitcoin) follow an order-execute design. However, as Fabric aims to be highly modular and scalable, a different approach is chosen. Fabric uses an execute-order-validate design that is significantly different. The execute-order-validate design is separated in three different phases, namely the *execute, order* and *validate phase*. Furthermore, there are three different types of peers (nodes) in a Fabric network, all of them responsible for different parts of the transaction flow process. The different peers are: *client peer*, *endorsing peer*, *ordering peer*. There are *non-endorsing* peers (sometimes called committing peers) as well, but these are the same type of peer as the endorsing peer, only not invoked to endorse for a particular transaction. The execute-order-validate design works as follows (Androulaki et al., 2018):

- *Execution phase:* A client peer (client) sends a transaction proposal to one or more endorsers for execution. The endorsers simulate the proposal against the endorser's local blockchain state (without any synchronization with other peers). Each endorser then produces a value writeset, which is a state update and a value readset, which represents the version dependencies. After the simulation, the endorser creates an endorsement, which is a cryptographically signed message that contains the simulated readset and writeset, and sends this back to the client in a proposal response. The client collects a beforehand determined amount of endorsements (based on the endorsement policy) and checks if all endorsers produced the same execution results. If so, it then follows to create the transaction and passes it to the ordering peers which are responsible for the ordering service.

- *Ordering phase:* The ordering peer (orderer) receives all transactions send by the client, it then continues to establish a total order on all submitted transactions. Thereafter the transactions are grouped into blocks, producing a hash-chained sequence of blocks containing transactions. Lastly, it broadcasts the blocks to the endorsing and non-endorsing peers. Note that the orderers do not maintain any state of the blockchain, nor validates or executes transactions.

- *Validation phase:* After the ordering service is ended, both the endorsing and the non-endorsing peers receive the produced block. These peers then start the endorsement policy evaluation, this action validates the endorsement (created by the endorser) with respect to the endorsement policy. If the results are negative, the transaction is marked as invalid and its effects are disregarded. After this, a read-write conflict check is done for each transaction in the block. This check compares the versions of the keys in the readset field to the, locally stored, current state of the ledger and ensures they are all the same. If this is not the case, the transaction is marked as invalid and its effect are disregarded. Lastly, the ledger update phase starts. This phase ensures that the block is appended to the locally stored blockchain and the world state is updated. However, the validity checks of the endorsement policy evaluation and the read-write conflict check are persisted as well. This means that the blockchain stores even the invalid transactions, which is radically different from Bitcoin and Ethereum, where only the valid transactions are contained in the ledger.

In the upcoming paragraphs, the different processes of the tuna fishery supply chain are modelled in BPMN, visualizing a transaction flow in Fabric based on the execute-order-validation design. In addition to that, the different transactions and produced blocks are modelled in Fact type diagrams (FTD's) accordingly.

3.3 Transaction flow: Record a Catch

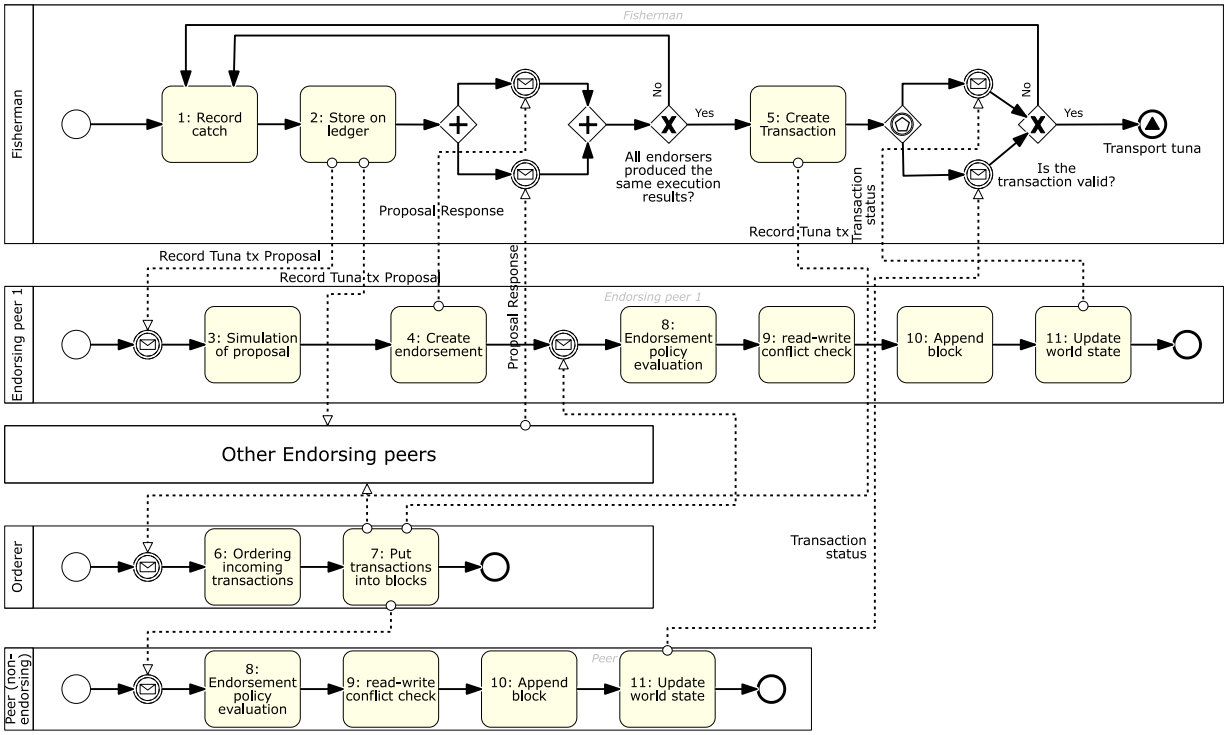


Figure 3.3: Transaction flow of a freshly caught tuna asset getting stored in Hyperledger Fabric

Figure 3.3 illustrates the transaction flow of a fisherman recording a tuna catch on a Hyperledger Fabric network. The sequential steps are as follows:

1) Record Catch: The process starts when the fisherman wants to record his caught Tuna. The fisherman is able to create a new asset, in this case freshly caught tuna. After each catch, the fisherman records information about each individual tuna via a simple data entry. This information contains, among other things, a ID number, the location and time of the catch and who caught the fish. These details will be saved in the world state as a key-value pair based on the specifications of a chaincode contract.

Execution phase (2-5)

2) Store on ledger: In this step, the fisherman sends the created tuna asset as a transaction to the network. The tuna transaction is now send to the endorsing peers as a proposal called "*Record Tuna tx proposal*". Note that in this model a distinction is made between recording the catch (meaning the fisherman enters the data) and storing the catch on the ledger (meaning the fisherman sends the entered data as a transaction to the peers).

3) Simulation of proposal: When the endorsing peers receive the “Record Tuna tx proposal”, the simulation starts which produces a state update (e.g. tuna is recorded) and a version number (logically “1” since the tuna asset is just created). Combined these are called the Read-Write (RW) sets.
4) Create endorsement: The information created in step 3 is now cryptographically signed by the endorser and send back to the fisherman in a proposal response.
5) Create transaction: The fisherman’s computer receives all the different proposal responses from the endorsing peers that were invoked and automatically checks if the execution results are the same. If this is not the case, the fisherman should start the process all over again. If the results are the same, the computer automatically creates the official transaction called “*Record Tuna tx*” and sends it to orderer. It depends on the network architecture how many orderers the network consists of. In this case, only one is modelled for the sake of simplicity. To preserve the overview of the model, no separate lane was made for the application, but note that there is a distinction between the fisherman as a person and the application on his computer creating the transactions. The fisherman does not actual need to check whether the transactions are the same, the application does this for him on the background.

Ordering phase (6-7)

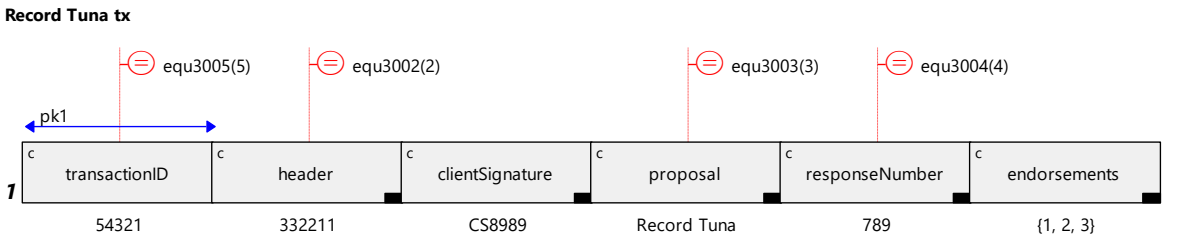
6) Ordering incoming transactions: The orderer receives the “Record Tuna tx” transaction, among all the other transactions of this channel, and continues to order them. This means a certain order among all incoming transactions is established to put the transactions into blocks.
7) Put transactions into blocks: Based on the order created in step 6, the “Record Tuna tx” transaction is now put into a block and send to the endorsing peers (from the execution phase) and the non-endorsing peers (all the peers in this channel). Note that from now on, there is no difference between the endorsing and the non-endorsing peers. Since Fabric is a distributed network, multiple peers doing the same tasks is essential.

Validation phase (8-11)

8) Endorsement policy evaluation: The endorsing and non-endorsing peers receive the produced block with the "Record Tuna tx" transaction in it. These peers start the endorsement policy evaluation which validates all the transactions in the blocks sequentially. The endorsement is checked with respect to the endorsement policy. If the results are negative, the "Record Tuna tx" transaction will be seen as invalid and no new tuna asset on the blockchain will be created. Otherwise, the Read-write conflict check will start.
9) Read-write conflict check: The peers check if the Read-Write sets from step 3 are still valid for the current world state. If this is not the case, the "Record Tuna tx" transaction will be marked as invalid and no new tuna asset on the blockchain will be created. After this, the block will be appended to the blockchain.
10) Append block: The block containing the "Record Tuna tx" transaction is added to the locally stored blockchain. Note that even if the transaction is marked as invalid in step 8 or step 9, the block is still added.
11) Update world state: Every peer updates the world state that is derived from the blockchain. If the "Record Tuna tx" is a valid transaction, then a new tuna asset is created on the blockchain. If "Record Tuna tx" is marked as an invalid transaction, then the organisations in this channel can still look into the invalid transaction, but no new tuna asset is created. The fisherman will now be notified by the different peers about the success or failure of his transaction. If the transaction is invalid, the process starts all over again. If the transaction is a success, the tuna is now ready to be transported to the processing plant.

3.3.1 Record Tuna Transaction

In step 5 the client node, coming from the fisherman, sends a transaction named "Record Tuna tx" to the orderer. The content of this transaction is represented in a fact type diagram and can be seen in figure 3.4.



1: The transaction identified by transaction ID <transactionID> is signed by a client with signature <clientSignature> and contains a header with number <header>, a proposal named <proposal>, a response with number <responseNumber> and is endorsed by peers <endorsements>

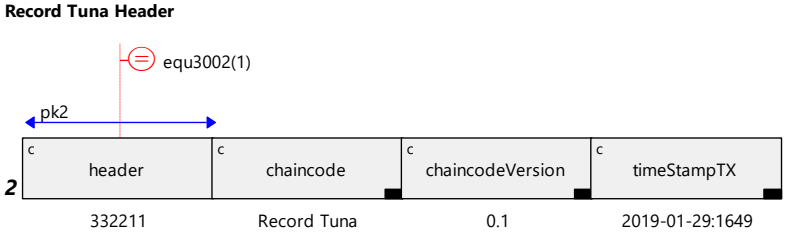
1) The transaction identified by transaction ID 54321 is signed by a client with signature CS8989 and contains a header with number 332211, a proposal named Record Tuna, a response with number 789 and is endorsed by peers {1, 2, 3}

Figure 3.4: A *Fabric transaction named "Record Tuna tx" modelled as a fact type diagram showing the different components*

As one can see in figure 3.4, a Fabric transaction consists of multiple fundamental components:

- Transaction ID: a unique number that identifies the transaction.
- Header: a number that describes some metadata about the transaction, which is illustrated in figure 3.5.
- Client Signature: a signature, based on asymmetric cryptography and created by the client application, used to check that the transaction has not been tampered with and that the transaction comes from a legitimate source.
- Proposal: the proposal provides the input parameters for the chaincode that determine the new world state. Illustrated in figure 3.6.
- Response: The response is produced by the endorsing nodes in the execution phase and is modelled here as a number, referring to the before and after values of the world state, as can be seen in figure 3.7.
- Endorsements: a list containing the endorsing peers that endorsed a particular transaction.

Note that since it is a conceptual model and therefore a simplistic view of reality, not all variables are represented, only the ones that are seemingly more important and distinctive than others, especially in regard to other blockchain platforms.



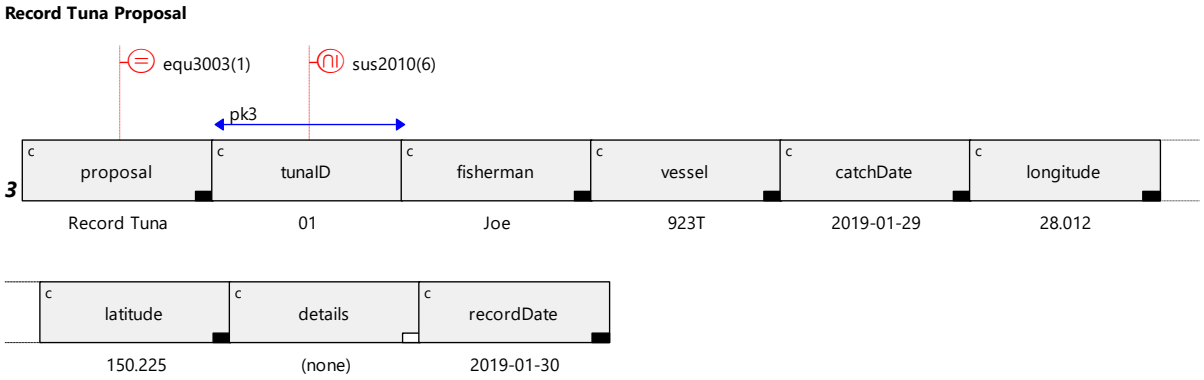
1: Header <header> refers to the chaincode function <chaincode> with chaincode version number <chaincodeVersion> and is sent on <timeStampTX>

1) Header 332211 refers to the chaincode function Record Tuna with chaincode version number 0.1 and is sent on 2019-01-29:1649

Figure 3.5: The header component of the "Record Tuna tx" transaction

Figure 3.5 shows an example of a header regarding the tuna transaction, where it can be seen that the header refers to which chaincode and chaincode version should be addressed and it refers to a timestamp, containing the exact date and time of when the transaction was sent.

Furthermore, the "Record Tuna tx" contains a proposal, which is one of the most essential parts of a transaction from a business perspective, as it contains real world information about the attributes of an asset, in this case a tuna. Such a proposal is modelled in figure 3.6 and one can see that an unique tuna ID represents a tuna asset with certain attributes that are useful in order to certify the legitimate source of a tuna.

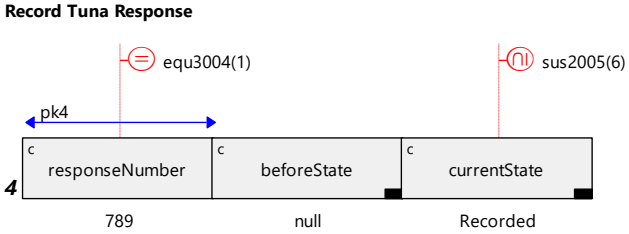


1: The proposal <proposal> is identified by a tuna with ID number <tunaID> which is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

1) The proposal Record Tuna is identified by a tuna with ID number 01 which is caught by Joe with vessel 923T on the date of 2019-01-29 at longitude 28.012 and latitude 150.225 with details (none) and is recorded on 2019-01-30

Figure 3.6: The proposal component of the "Record Tuna tx" transaction

Lastly, as part of the "Record Tuna tx", a response is modelled, illustrated in figure 3.7. This response is in parallel produced by multiple endorsing peers during the execution phase. Due to the endorsement policy that every endorsing peer should produce the same outcome, the response will be the same for each endorsing peer. Since the tuna with tuna ID 01 did not exist before the "Tuna Record tx" transaction, the state is null (not existing). However, once the transaction is validated and added to a block that is part of the blockchain, the state of the tuna with ID 01 is recorded. As from now on, the tuna also digitally exists.



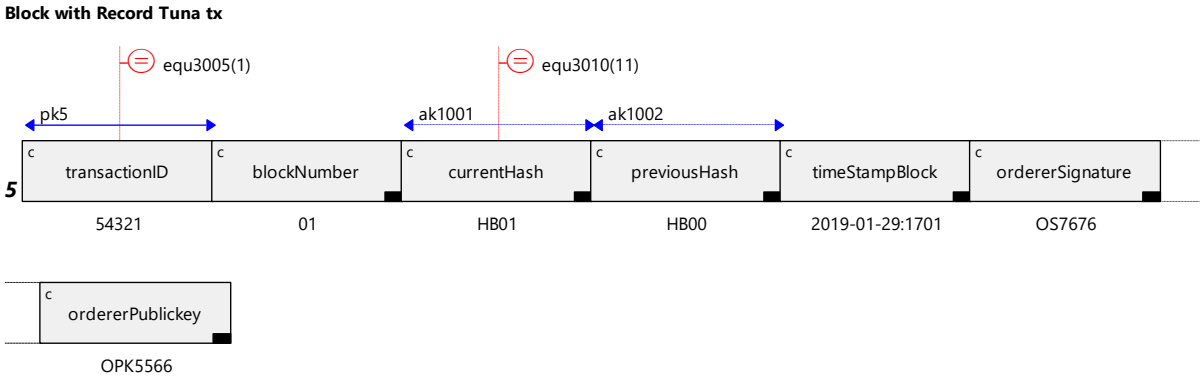
1: The response identified by response number <responseNumber> previously had world state <beforeState> and currently has world state <currentState>

1) The response identified by response number 789 previously had world state null and currently has world state Recorded

Figure 3.7: The response component of the "Record Tuna tx" transaction

3.3.2 Block and World State: Record Tuna

After the transaction “Record Tuna tx” has undergone the process of the ordering service and is validated once more by the peers, the new block containing this “Record Tuna tx” transaction is appended to the blockchain and this transaction has now affected the world state. As was discussed in paragraph 3.2, the world state is derived from the blockchain and represents the current world state. Figure 3.8 illustrates a block that contains the “Record Tuna tx” where the world state, illustrated in 3.9, is derived from.



1: The transaction identified by ID number <transactionID> is added to a block with block number <blockNumber>, which contains the current hash <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the orderer with signature <ordererSignature> and public key <ordererPublickey>

1) The transaction identified by ID number 54321 is added to a block with block number 01, which contains the current hash HB01, the hash of the previous block HB00, is created on 2019-01-29:1701 and signed by the orderer with signature OS7676 and public key OPK5566

Figure 3.8: A block appended to the blockchain containing the “Record Tuna tx” transaction

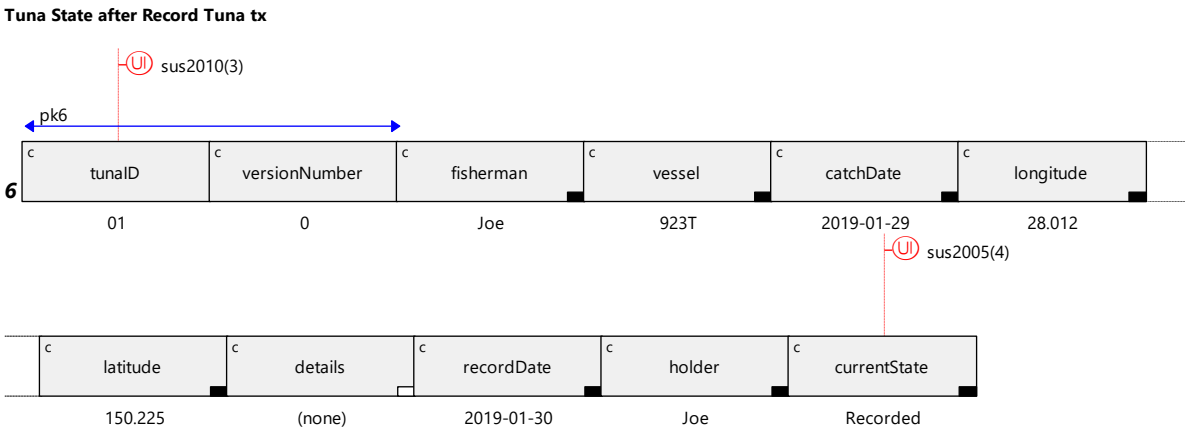
A block with a certain block number can contain multiple transactions or none at all, depending on how many transactions are submitted and the block size of one block. In this case, the block with block number 01 only contains one transaction, namely the “Record Tuna tx” identifiable by the unique transaction ID 54321.

Furthermore, an appended block has certain attributes associated with it, these are, among other things:

- The current hash that is produced by hashing all of the transactions contained by this block.
- The previous hash, which is a copy of the current hash of the previous block in the blockchain. A guarantee that this new block followed up on the then latest appended block.
- A timestamp with a specific date on which the block was created by the orderer.

- The signature of the orderer and its Public key. Both used in the underlying cryptographical procedures to prove the legitimacy of the creator of the blocks (the orderer).

The current world state derived from the latest appended block, modelled in figure 3.8, can be seen in figure 3.9. Almost similar to paragraph 3.2.2, where the example of cars as digitized assets is illustrated, the tuna asset is now presented. Most of the values presented in the world state are derived from the proposal (figure 3.6) within the transaction component. Moreover, there is a version number, just as in the car example; a holder, that shows which entity currently holds the asset (this will, in most cases, be the entity that sent the last transaction regarding a specific asset) and a current state in which the asset is in.



1: The state of the tuna identified by the combination of ID number <tunaID> and version number <versionNumber> is recorded by <fisherman> on <recordDate>, is currently hold by <holder>, has the following details <details>, is caught with vessel <vessel> on <catchDate> at longitude <longitude> and latitude <latitude> and its current state is <currentState>

1) The state of the tuna identified by the combination of ID number 01 and version number 0 is recorded by Joe on 2019-01-30, is currently hold by Joe, has the following details (none), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Recorded

Figure 3.9: The state of the newly created tuna after Record Tuna tx is validated and added to a block

In the next paragraph, the transaction flow of the Processing the Tuna, where the "Processed Tuna tx" transaction is submitted, is presented in the same manner as the Record a Catch, where the "Record Tuna tx" transaction is submitted. This paragraph, among other things, shows the effect this transaction has on the world state of the tuna asset.

3.4 Transaction flow: Processing the Tuna

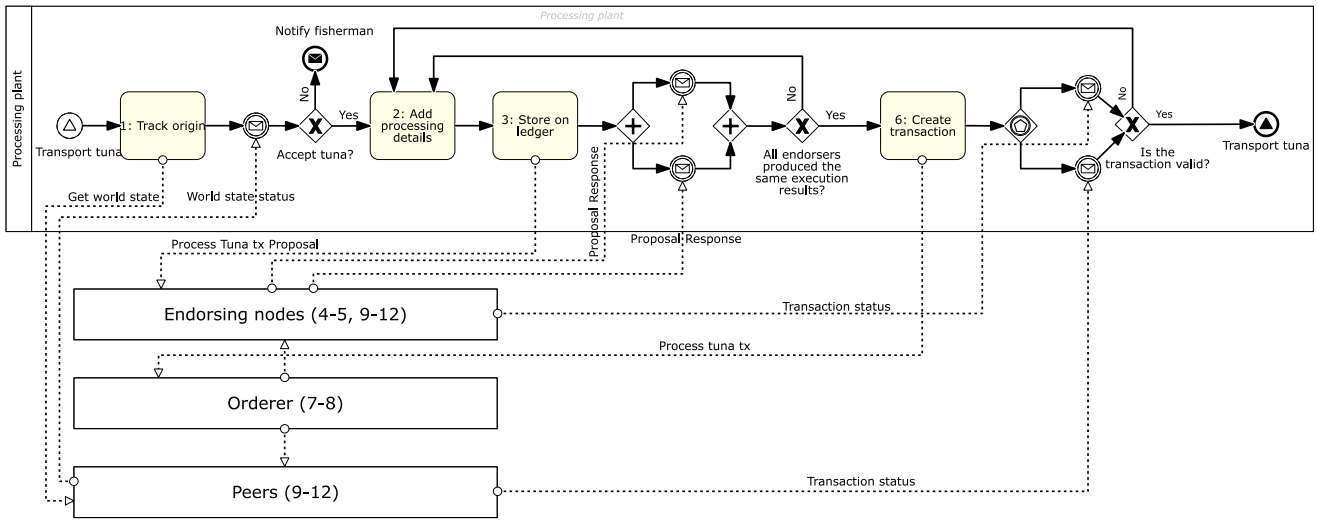


Figure 3.10: Transaction flow of the tuna asset being processed in Hyperledger Fabric

After the tuna is transported to a plant, it is ready to be processed. This transaction flow is illustrated in figure 3.10, where one can see that the process looks very similar to that of recording an asset. The sequential steps are as follows:

1) Track origin: The process starts with the client node, in this case the processing plant, verifying the origin of the received tuna. This is done by the client node querying one of the peers with a request to give information about the tuna with the corresponding ID number. The peer should then give information about the latest state of the tuna with the corresponding ID number, this information should correspond to the tuna state showed in figure 3.9.
2) Add processing details: If the origin of the tuna is verified and the tuna state is approved by the processing plant, then the processing plant can proceed to process the tuna. After the processing is done, details about the processing can be entered into the computer.

Execution phase (3-6)

3) Store on ledger: The processing plant continues to store the entered data on the ledger by sending the data in the form of a transaction proposal called "*Process Tuna tx proposal*" to the endorsing peers.

4-6) Simulation of proposal, Create endorsement, Create transaction: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3, Record a Catch, with the only difference being the name of the transaction “Process Tuna tx” and its content.

Ordering phase (7-8)

7,8) Ordering incoming transactions, Put transactions into blocks: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3, Record a Catch, with the only difference being the name of the transaction “Process Tuna tx” and its content.

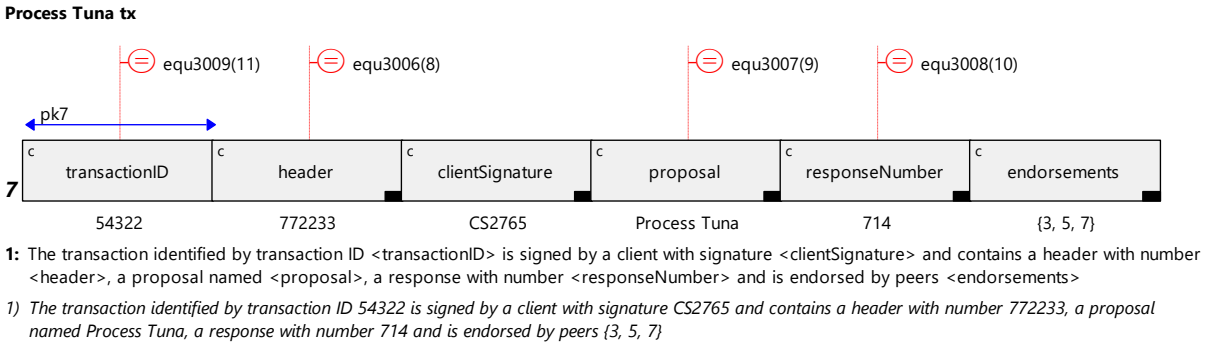
Validation phase (9-12)

9-11) Endorsement policy evaluation, Read-write conflict check, Append block: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3, Record a Catch, with the only difference being the name of the transaction “Process Tuna tx” and its content.

12) Update world state: Every peer updates the world state that is derived from the blockchain. If “Process Tuna tx” is a valid transaction, then the information about the tuna asset is updated. If “Process Tuna tx” is marked as an invalid transaction, then the information about the tuna will not be updated. The processing plant will now be notified by the different peers about the success or failure of the transaction. If the transaction is invalid, the process starts all over again. If the transaction is a success, the tuna, with the digitally stored information about the processing, is now ready to be transported to the restaurant. Note that in case of the processing plant rejecting the tuna asset, a transaction would be send to the endorsing peers as well. However, this transaction would not have information about the processing, but would only contain a state update from “Recorded” to “Rejected” instead of “Processed”. Since the transaction flow of a rejected tuna is largely the same as a processed one and therefore no added value for the comparison would be gained, it was not modelled but merely noted as “notify fisherman”.

3.4.1 Process Tuna Transaction

After the “Process Tuna tx” transaction is created in step 6, the client node, being the processing plant, sends it to the orderer. Just as with the “Record Tuna tx” transaction, the content of this transaction is represented in a fact type diagram and can be seen in figure 3.11.

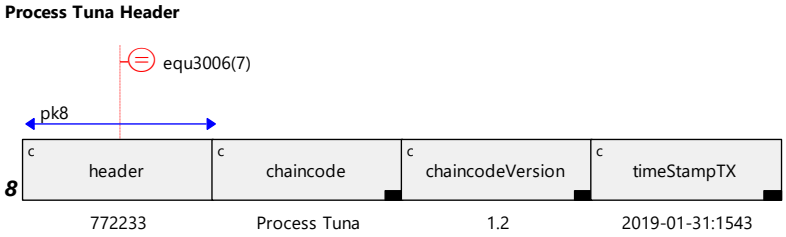


1: The transaction identified by transaction ID <transactionID> is signed by a client with signature <clientSignature> and contains a header with number <header>, a proposal named <proposal>, a response with number <responseNumber> and is endorsed by peers <endorsements>

1) The transaction identified by transaction ID 54322 is signed by a client with signature CS2765 and contains a header with number 772233, a proposal named Process Tuna, a response with number 714 and is endorsed by peers {3, 5, 7}

Figure 3.11: A Fabric transaction named “Process Tuna tx” modelled as a fact type diagram showing the different components

One can see that the transaction is made of the same components as the “Record Tuna tx” transaction, only this time with different instances.

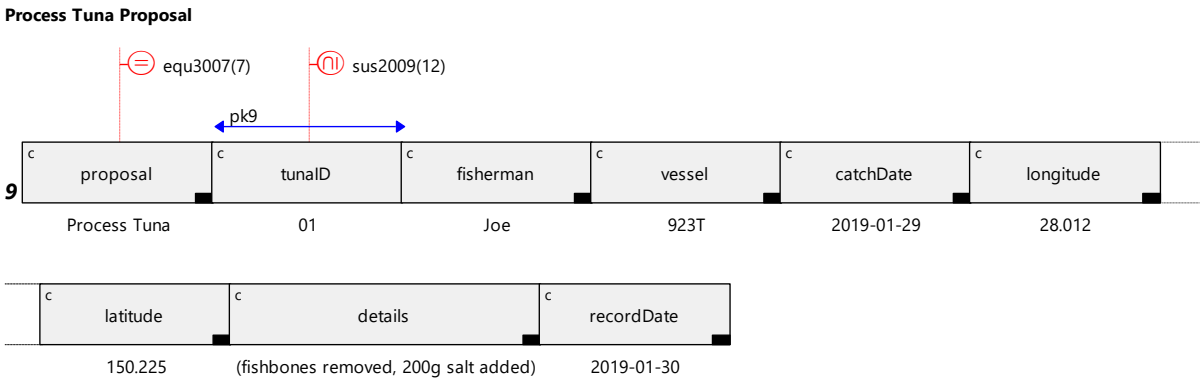


1: Header <header> refers to the chaincode function <chaincode> with chaincode version number <chaincodeVersion> and is sent on <timeStampTX>

1) Header 772233 refers to the chaincode function Process Tuna with chaincode version number 1.2 and is sent on 2019-01-31:1543

Figure 3.12: The header component of the “Process Tuna tx” transaction

The header (shown in figure 3.12) again refers to the specific application (chaincode) that needs to run (note that this time Process Tuna is invoked instead of Record Tuna) and the right chaincode version, as well as the exact date of when the transaction was sent.



1: The proposal <proposal> is identified by a tuna with ID number <tunaID> which is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

1) The proposal Process Tuna is identified by a tuna with ID number 01 which is caught by Joe with vessel 923T on the date of 2019-01-29 at longitude 28.012 and latitude 150.225 with details (fishbones removed, 200g salt added) and is recorded on 2019-01-30

Figure 3.13: The proposal component of the "Process Tuna tx" transaction

The proposal component still contains the information about the circumstances of the tuna catch and this transaction adds the details of the tuna's processing to that.

Lastly, the response is modelled, where information about the old state and the new state of the tuna is stored. This is illustrated in figure 3.14. Once this transaction is validated and put into a new block, the state of the tuna with ID 01 will change from "Recorded" to "Processed".

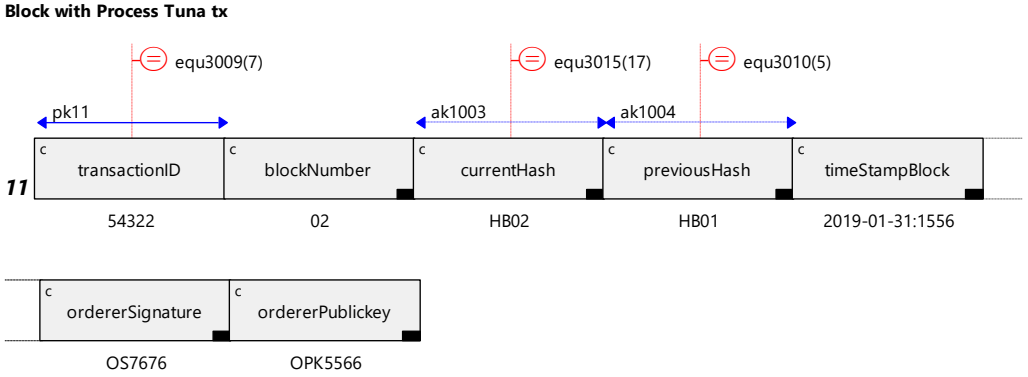
1: The response identified by response number <responseNumber> previously had world state <beforeState> and currently has world state <currentState>

1) The response identified by response number 714 previously had world state Recorded and currently has world state Processed

Figure 3.14: The Response component of the "Process Tuna tx" transaction

3.4.2 Block and World State: Process Tuna

As with every Fabric transaction, “Process Tuna tx” must undergo the process of the ordering service and is validated once more by the peers afterwards. The new block containing “Process Tuna tx” is appended to the blockchain and this transaction has now affected the world state. The block containing this transaction is modelled in figure 3.15.

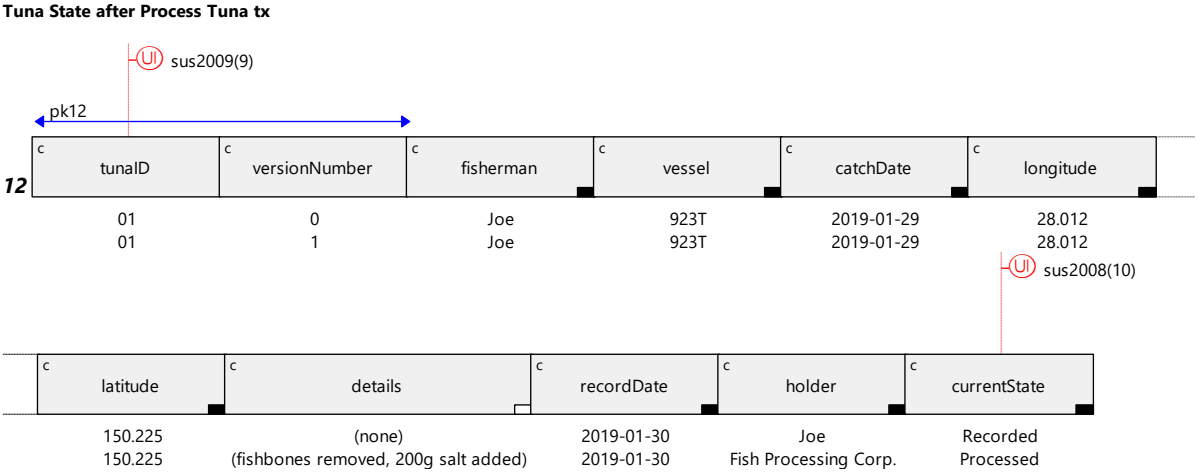


1: The transaction identified by ID number <transactionID> is added to a block with block number <blockNumber>, which contains the current hash <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the orderer with signature <ordererSignature> and public key <ordererPublickey>

1) The transaction identified by ID number 54322 is added to a block with block number 02, which contains the current hash HB02, the hash of the previous block HB01, is created on 2019-01-31:1556 and signed by the orderer with signature OS7676 and public key OPK5566

Figure 3.15: A block appended to the blockchain containing the “Process Tuna tx” transaction

Similar to the block containing the “Record Tuna tx” transaction, this block also contains a transaction with a unique transaction ID that refers to the “Process tuna tx” transaction.



1: The state of the tuna identified by the combination of ID number <tunaID> and version number <versionNumber> is recorded by <fisherman> on <recordDate>, is currently hold by <holder>, has the following details <details>, is caught with vessel <vessel> on <catchDate> at longitude <longitude> and latitude <latitude> and its current state is <currentState>

1) The state of the tuna identified by the combination of ID number 01 and version number 0 is recorded by Joe on 2019-01-30, is currently hold by Joe, has the following details (none), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Recorded

2) The state of the tuna identified by the combination of ID number 01 and version number 1 is recorded by Joe on 2019-01-30, is currently hold by Fish Processing Corp., has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Processed

Figure 3.16: The state of the tuna after Process Tuna tx is validated and added to a block

In addition to that, the block has certain attributes that identify its own origin and uniqueness. For example, note that the current hash of block 01 (shown in figure 3.8) is now the previous

hash of block 02, indicating a chain of blocks. The world state that derives from this block in combination with the previous blocks can be viewed in figure 3.16. The world state now shows the current and the previous state of the tuna, enabling one to obtain a clear overview of the history of the tuna with ID 01.

3.5 Transaction flow: Accepting the Tuna

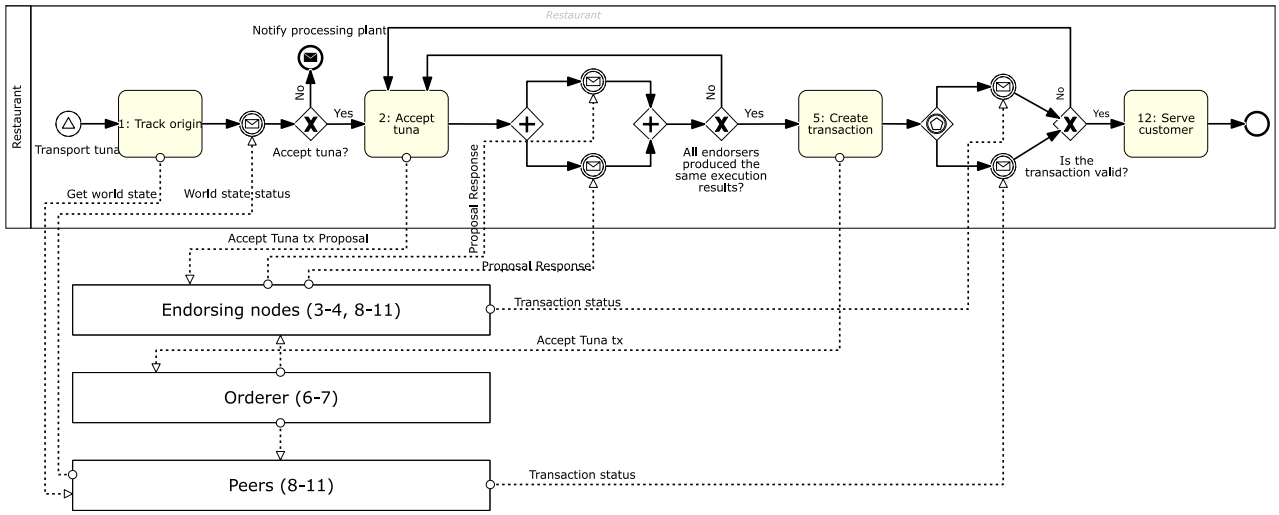


Figure 3.17: Transaction flow of the tuna asset being accepted in Hyperledger Fabric

Once the tuna is processed, it is transported to different vendors. In this use case, a restaurant receives the tuna which will prepare and serve it to their customers. The transaction flow of the restaurant receiving the tuna and serving it is illustrated in figure 3.17, where one can see that the process looks very similar to that of the "Process Tuna tx" transaction flow (figure 3.10). The sequential steps are as follows:

1) Track origin: The process starts with the restaurant checking the legitimacy of the tuna by verifying the different sources. This is done in the same manner as with Process Tuna tx, the client node queries one of the peers with a request to give information about the tuna with the corresponding ID number. The peer should then give information about the latest state of the tuna with the corresponding ID number, this information should now correspond to the tuna state showed in figure 3.16.
2) Accept tuna: Since the restaurant is the tuna's last stop, it now only needs to be accepted. That is if the origin of the tuna is verified and the tuna state is approved by the restaurant of course. Otherwise, a "Reject Tuna tx" will be send, as discussed in paragraph 3.4. Note that the "Accept tuna" step does not require any data entry by the client, in contrast to the previous two transaction flows. Therefore, the "Store on ledger" step is not specified, since this will automatically be done when the restaurant clicks the accept button on their computer. After accepting the tuna, a transaction proposal named "Accept tuna tx proposal" is send from the client node to the endorsing peers.

Execution phase (3-5)

3-5) Simulation of proposal, Create endorsement, Create transaction: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3 and 3.4, Record a Catch and Processing the tuna, with the only difference being the name of the transaction “Accept Tuna tx” and its content.

Ordering phase (6-7)

6,7) Ordering incoming transactions, Put transactions into blocks: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3 and 3.4, Record a Catch and Processing the Tuna, with the only difference being the name of the transaction “Accept Tuna tx” and its content.

Validation phase (8-11)

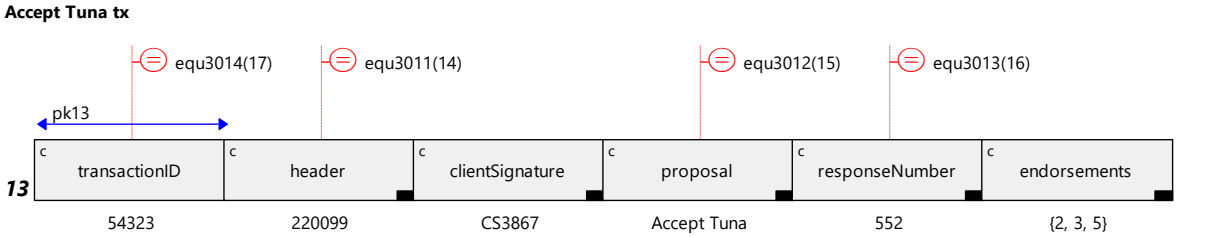
8-10) Endorsement policy evaluation, Read-write conflict check, Append block: The descriptions of these steps are omitted because they cover the same process as in paragraph 3.3 and 3.4, Record a Catch and Processing the Tuna, with the only difference being the name of the transaction “Accept Tuna tx” and its content.

11) Update world state: Every peer updates the world state that is derived from the blockchain. If the “Accept Tuna tx” is a valid transaction, then the state of the tuna is updated. If “Accept Tuna tx” is marked as an invalid transaction, then the state of the tuna is not updated. The restaurant will now be notified by the different peers about the success or failure of the transaction. If the transaction is invalid, the process starts all over again. If the transaction is a success, the tuna with ID 01 is ready to be served to a customer.

12) Serve customer: The restaurant serves tuna with ID 01 to a customer, ending the lifecycle of this asset.

3.5.1 Accept Tuna Transaction

The “Accept Tuna tx” is a straightforward transaction that is created in step 5 by the client node and send to the orderer. Similar to the previous two discussed transactions, the content of this transaction is also represented in a fact type diagram and can be seen in figure 3.18.

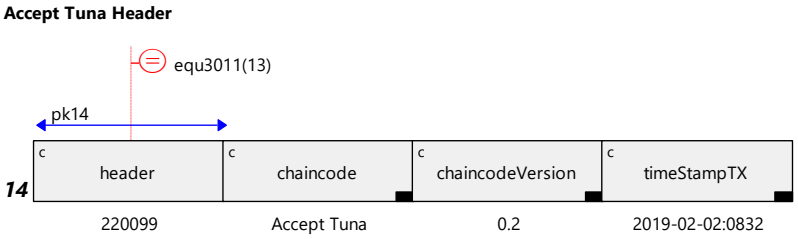


1: The transaction identified by transaction ID <transactionID> is signed by a client with signature <clientSignature> and contains a header with number <header>, a proposal named <proposal>, a response with number <responseNumber> and is endorsed by peers <endorsements>

1) The transaction identified by transaction ID 54323 is signed by a client with signature CS3867 and contains a header with number 220099, a proposal named Accept Tuna, a response with number 552 and is endorsed by peers {2, 3, 5}

Figure 3.18: *A Fabric transaction named “Accept Tuna tx” modelled as a fact type diagram showing the different components*

No noticeable differences are apparent when compared to “Record Tuna tx” and the “Process Tuna tx”, except for the different instances that are used.



1: Header <header> refers to the chaincode function <chaincode> with chaincode version number <chaincodeVersion> and is sent on <timeStampTX>

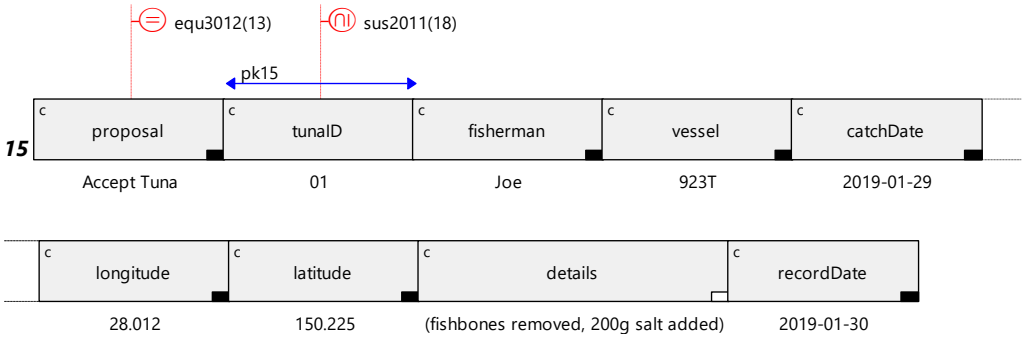
1) Header 220099 refers to the chaincode function Accept Tuna with chaincode version number 0.2 and is sent on 2019-02-02:0832

Figure 3.19: The header component of the “Accept Tuna tx” transaction

The header component clearly shows that it refers to the name of the application (chaincode) that must run, including the corresponding chaincode version, as well as the details of the transaction sent.

Furthermore, the proposal component stays the same as in the “Process Tuna tx”, since no new data entry is done by the restaurant. This results in the model illustrated in figure 3.20 where one can see that there is no noticeable difference compared to the Process Tuna Proposal (figure 3.13).

Accept Tuna Proposal



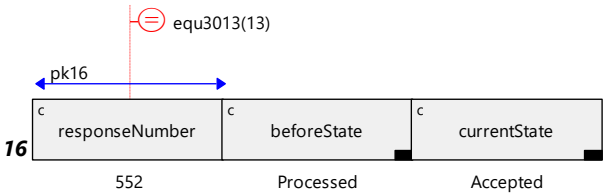
1: The proposal <proposal> is identified by a tuna with ID number <tunaID> which is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

1) The proposal Accept Tuna is identified by a tuna with ID number 01 which is caught by Joe with vessel 923T on the date of 2019-01-29 at longitude 28.012 and latitude 150.225 with details (fishbones removed, 200g salt added) and is recorded on 2019-01-30

Figure 3.20: The proposal component of the "Accept Tuna tx" transaction

Lastly, the response is modelled which contains arguably the most important element of this transaction, namely the state change from "Processed" to "Accepted" (figure 3.21).

Accept Tuna Response



1: The response identified by response number <responseNumber> previously had world state <beforeState> and currently has world state <currentState>

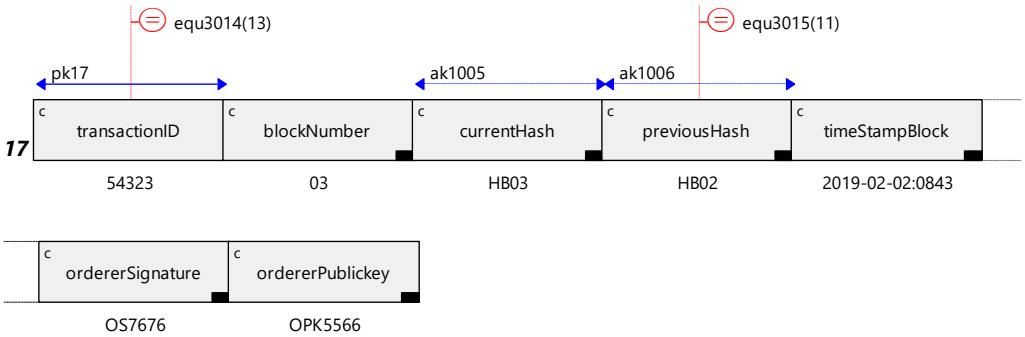
1) The response identified by response number 552 previously had world state Processed and currently has world state Accepted

Figure 3.21: The Response component of the "Accept Tuna tx" transaction

3.5.2 Block and World State: Accept Tuna

The ordering service ensures “Accept Tuna tx” is put into a block. The new block containing the “Process Tuna tx” is appended to the blockchain and this transaction now affects the world state. The block containing this transaction is modelled in figure 3.22.

Block with Accept Tuna tx



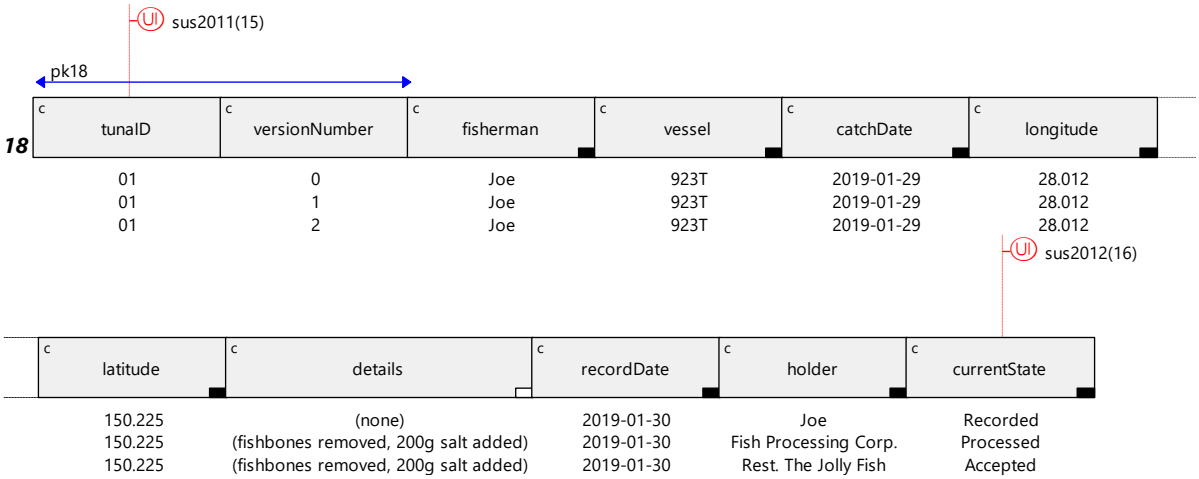
1: The transaction identified by ID number <transactionID> is added to a block with block number <blockNumber>, which contains the current hash <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the orderer with signature <ordererSignature> and public key <ordererPublickey>

1) The transaction identified by ID number 54323 is added to a block with block number 03, which contains the current hash HB03, the hash of the previous block HB02, is created on 2019-02-02:0843 and signed by the orderer with signature OS7676 and public key OPK5566

Figure 3.22: A block appended to the blockchain containing the “Accept Tuna tx” transaction

The sequence of blocks is continued due to the fact that the current hash in figure 3.15 is now the previous hash in figure 3.22. Moreover, the unique transaction ID is modelled, indicating that the block with block number 03 contains the “Accept Tuna tx” transaction.

Tuna State after Accept Tuna tx



1: The state of the tuna identified by the combination of ID number <tunaID> and version number <versionNumber> is recorded by <fisherman> on <recordDate>, is currently hold by <holder>, has the following details <details>, is caught with vessel <vessel> on <catchDate> at longitude <longitude> and latitude <latitude> and its current state is <currentState>

1) The state of the tuna identified by the combination of ID number 01 and version number 0 is recorded by Joe on 2019-01-30, is currently hold by Joe, has the following details (none), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Recorded

2) The state of the tuna identified by the combination of ID number 01 and version number 1 is recorded by Joe on 2019-01-30, is currently hold by Fish Processing Corp., has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Processed

3) The state of the tuna identified by the combination of ID number 01 and version number 2 is recorded by Joe on 2019-01-30, is currently hold by Rest. The Jolly Fish, has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Accepted

Figure 3.23: The state of the tuna after Accept Tuna tx is validated and added to a block

Figure 3.23 shows the latest state for the tuna with ID number 01 along with the entire track record of this particular tuna asset. Since a tuna is a consumable asset, one can deduce that a tuna being accepted by a restaurant will end the lifecycle of that asset. This, of course, is not the case for many other non-consumable assets. For example in real estate, where a house being bought by a consumer does not mean that the lifecycle of that asset ends.

Furthermore, this use case is meant to show the fundamentals of transaction flows along with the transactions itself in the context of Hyperledger Fabric, in order to make a proper comparison between two platforms. Therefore arguments about the realism of the shown supply chain would not contribute to the matter, since the use case is deliberately made simplistic in order to understand the basic concepts.

Chapter 4

Hyperledger Sawtooth

4.1 Introduction

This chapter contains the conceptual models for Hyperledger Sawtooth following the CogNIAM method. The same use case regarding the simplified fictional tuna fishery supply chain, as introduced in chapter 3.1, is modelled to ensure that both platforms can be compared adequately. In order to make a proper comparison the actors remain the same, namely: 1) a fisherman that catches the tuna; 2) a tuna processing plant that processes the tuna; and 3) a restaurant that will serve the tuna. In the following paragraphs, fundamental concepts of Sawtooth are discussed, the different processes of the tuna fishery supply chain are modelled in BPMN and the different transactions and produced blocks are modelled in Fact type diagrams (FTD's) accordingly.

4.2 Reinventing a better wheel

To enhance the understanding of Hyperledger Sawtooth and the processes within, the following paragraphs discuss some of its fundamental concepts. These concepts sometimes overlap the previous discussed features of traditional blockchains and sometimes they radically change the traditional design. As beforementioned, there are also some similarities with Fabric.

4.2.1 World State versus Global State

One of the more important design features that is implemented in Sawtooth, just like in Fabric, is the separation of a blockchain and a database. As mentioned in chapter 3, Bitcoin does not make use of a database and the current state of the chain is always calculated by going through all of the transactions in the blockchain (Blockgeeks, 2018).

In Fabric, a ledger is clearly defined as the set of two distinct components, namely the world state and a blockchain. In Sawtooth however, these definitions are less clear and contradict the terminology used in Fabric. For example, a ledger in Sawtooth is solely defined as a key-value store whose values are agreed on by all nodes (validators) in the network (Hyperledger Sawtooth F.A.Q., 2019).

Furthermore, the term World State is seldom used and instead the Global State is the common term to describe the current state derived from the blockchain (Hyperledger Sawtooth F.A.Q., 2019). To quote one of the answers from the Hyperledger Sawtooth F.A.Q. (2019):

"*The blockchain itself just stores transactions, not state, so reading the data in the last block does not say much by itself. Data in the blockchain is also immutable and can never change (except by adding new blocks). The radix tree is a different data structure that is used to make fast queries to the state.*"

Although the technical implementation of the data storing in the databases is indeed different and not within the scope of this thesis to discuss any further, the separation of a blockchain and a database is the same. This also means that Sawtooth is capable to digitize assets on its network.

4.2.2 Proof of Elapsed time

In opposition to Hyperledger Fabric, Sawtooth does use a single algorithm in production default to reach consensus among their nodes. This algorithm is called Proof of Elapsed time (PoET) and is very similar to the Proof of Work (PoW) algorithm, in the sense that it is a Nakamoto-style consensus algorithm meaning that all the nodes in the network participate and compete each other in order to be the first node that may broadcast a new block. The competition works fundamentally different though.

In the Proof of Work algorithm the computational power of a node is decisive, due to the fact that more computational power means a higher chance of solving the cryptographic puzzle (needed to add a new block) in lesser time (Antonopoulos, 2014). This also results in the usage of increasingly more expensive equipment in order to be more competitive, not to mention the amount of electricity it costs to power these computers.

The basic idea of the Proof of Elapsed Time algorithm is that every validator node in the network generates a random wait time for every new block that it wants to create. The validator who's timer expires first is allowed to broadcast the new block. The random wait time is generated by a function located in the Trusted Execution Environment (TEE) of a processor chip. Since Intel is one of the major driving forces behind Sawtooth, it is no surprise that at this moment, only Intel chips offer these TEE's. These TEE's are realized by using a Software Guard Extension (SGX). The SGX runs programs in the protected areas (TEE's) that are also called enclaves. A TEE ensures that a function located in such a protected area cannot be tampered with (Hoops, 2017).

One of the major benefits of PoET is that every computer, regardless of their computational power, has the same chances of producing the newest block. A disadvantage is the heavy dependence of Intel's services, which, in the context of a Permissioned Private Blockchain, can be considered as not that big of a problem.

4.2.3 Batches and Transaction Processors

By applying transactions, modifications to world/global state can be made. This is no different for Sawtooth. Nevertheless, there are some notable differences in comparison to other platforms.

First of all, Sawtooth introduces a new concept named Batches. A client creates a transaction and submits it to the validator node. The validator node then applies the transaction

which causes a change to state. However, before a transaction is actually sent to the validator node, the client wraps it inside a batch, even when only one single transaction is sent. All transactions within a batch are committed to state together or not at all. This means that in Sawtooth, batches are the atomic unit of state change and not transactions. Note that batches do not replace blocks, but can be seen as a new component conceptually located in between transactions and blocks. Batches are implemented to simplify dependency management from a technical perspective between transactions and it can be beneficial when, for example, two changes are to be made, where applying only one of these changes might have a catastrophic effect on the system (Hoops, 2017).

Secondly, Sawtooth separates the application level from the core system and it does so by implementing two concepts named Transaction Processors and Transaction Families. The Transaction Processor component understands the business logic within Sawtooth and is therefore responsible for the validation and processing of the transaction's payload and updates the global state based on the rules defined by the associated transaction family. A transaction family is thus the business logic that defines a set of operations or transaction types that are allowed on the blockchain within a Sawtooth network. Sawtooth transaction families separate the transaction rules and content from the Sawtooth core functionality. A transaction family implements a data model and transaction language for an application. Transaction Families can be seen as the smart contracts for Sawtooth, just like chaincode is for Fabric.

4.2.4 Order-execute design

Fabric uses the execute-order-validate design where the processing of transactions is divided in three different phases, namely the execute, order, and validate phase. In Sawtooth however, the traditional design is chosen, in the sense that first the transactions are ordered and then they will be executed, hence the name order-execute design.

Furthermore, there are only two types of peers/nodes in a Sawtooth network, these are: *client node* and *validator node*. Unlike the different nodes in Fabric, every validator node has the same responsibilities in the transaction flow process. The order-execute design comes down to the following:

- *Ordering phase:* A client node sends a transaction, wrapped inside a batch, to all the validator nodes in the network. Within the validator node, the journal component is responsible for the processing of batches and blocks. Once a validator receives a batch,

the journal starts to check for dependencies between transactions. A batch is considered complete once all of its dependent transactions exist in the current chain or are waiting to be included into a block. When all dependencies have been processed, the batch itself is tested on its validity. If the batch is considered valid, the transaction within the batch is processed by the transaction processor. Otherwise, the batch is dropped.

- *Execution phase*: The transaction processor first validates the transaction and then executes it. If the transaction is considered as invalid, the batch is dropped. The execution of a transaction means that the transaction processor computes the effects of the transaction on the global state, but only when the transaction is included to a block and added to the blockchain, the global state is updated. After the execution of a transaction, the batch will be included in a block. Since all the validators started this exact same procedure when they received the batch, an election is needed to decide which validator may broadcast its block. For this, the PoET consensus algorithm is used. Every validator creates a timer by calling the create timer function in the enclave and sends the results to each other. The validator with the shortest wait time is elected to broadcast its block to the other validators. Once the elected validator has broadcasted its block, it will append its own block to the blockchain and then update its world state accordingly. The other validators, however, will receive the broadcasted block from the elected validator. Since a blockchain is meant to create trust between untrusting entities, the validators do not trust each other and are therefore forced to compute all the transactions within the received block. This means that even though the validators already computed the transaction in the ordering phase, this is disregarded and every validator will do it once more for the received block. First, the other validators check the dependencies for the incoming block, similar to the dependency check for the batch. Then continue to validate the legitimacy of the incoming block. If the block is valid, the transaction processor will validate the transaction. Otherwise the block is dropped by the validator. If the transaction is valid, it will be executed. Otherwise, the block is also dropped. After execution, the incoming block is appended to the blockchain, the global state is updated and the client is informed about the success of the transaction. Note that 1) in most cases a validator will have multiple transaction processors. For each transaction family, there is an associated transaction processor. And 2) a major difference between Fabric and Sawtooth is the fact that in Sawtooth, invalid transaction

are not added to a block and therefore not stored in the blockchain. Thus, adding a transaction to a block, means that the transaction is de facto a success.

4.3 Transaction flow: Record a Catch

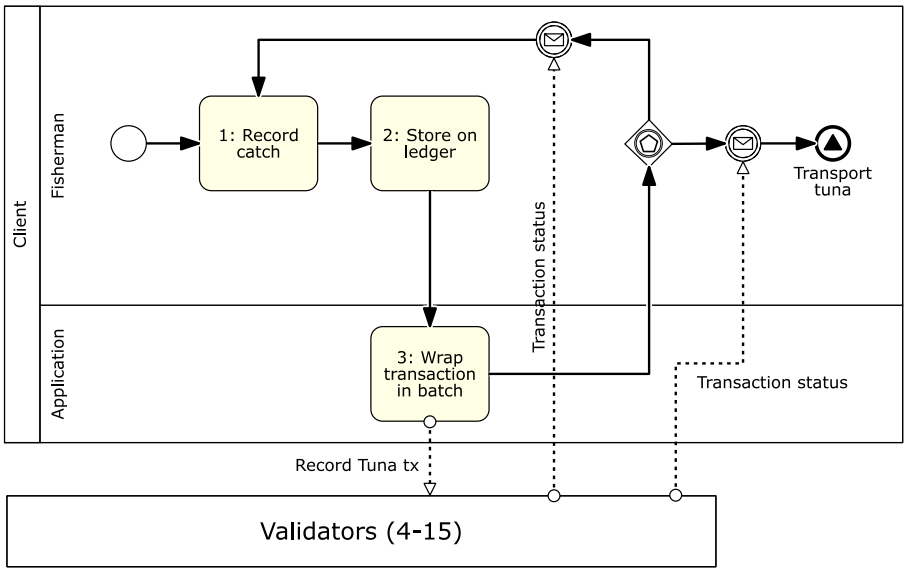


Figure 4.1: Transaction flow of a freshly caught tuna asset getting stored in Hyperledger Sawtooth from a client's perspective

In order to provide a structured overview of the transaction flow, the different transaction flow phases are split up among three separate models. Figure 4.1 illustrates the transaction flow of a fisherman recording a tuna catch on a Hyperledger Sawtooth network from a client's perspective. Figure 4.2 shows the transaction flow from a validator nodes' perspective. The sequential steps for the entire transaction flow are as follows:

1) Record Catch: The process starts when the fisherman wants to record his caught Tuna. The fisherman is able to create a new asset, in this case freshly caught tuna. After each catch, the fisherman records information about each individual tuna via a simple data entry. This information contains, among other things, a ID number, the location and time of the catch and who caught the fish.

Ordering phase (2-5)

2) Store on ledger: The fisherman sends the created tuna asset as a transaction called "*Record Tuna tx*" to the network. However, since each transaction must be wrapped inside a batch, the client will first proceed to do so.
3) Wrap transaction in batch: The transaction is wrapped inside a batch by the client and then send to all the validator nodes in the network. The fisherman does not

need to do it himself, but it will be done automatically by the client's application once the transaction is sent by the fisherman. In this use case, the client that puts the transaction in the batch is the same entity as the computer of the fisherman but this does not have to be the case.

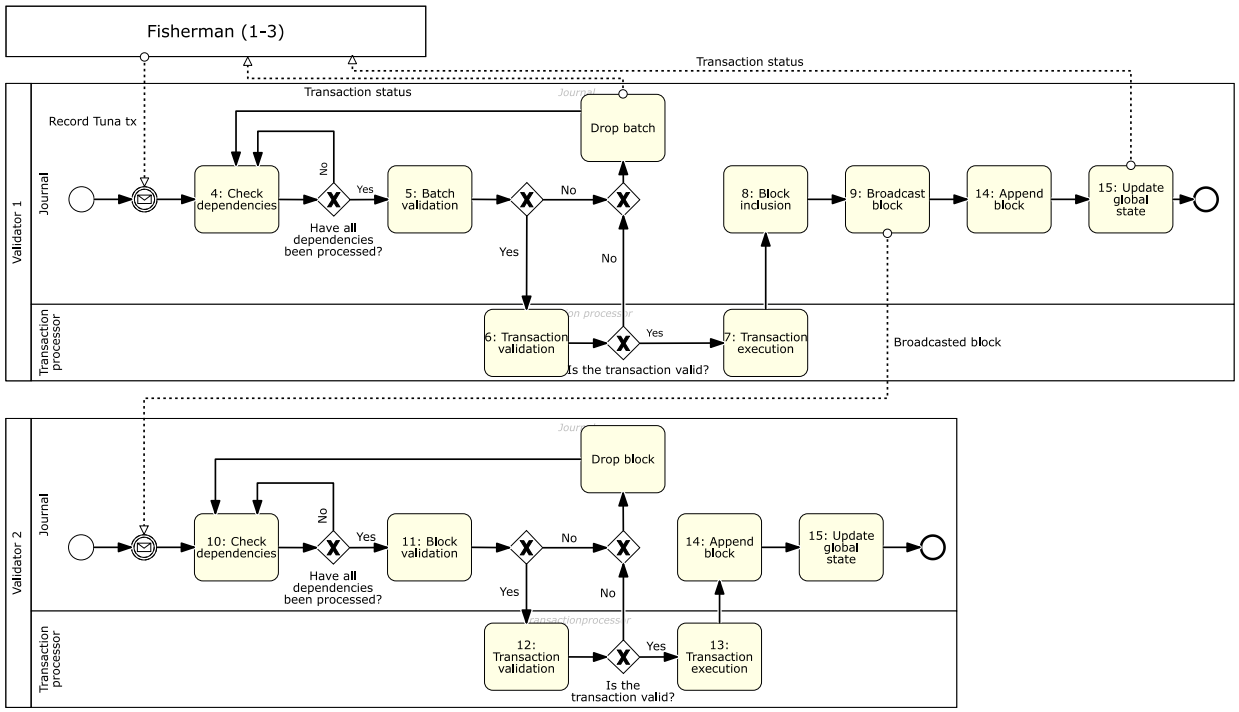


Figure 4.2: Transaction flow of the "Record Tuna tx" in Hyperledger Sawtooth from the validator nodes' perspective

4) Check dependencies: Once a validator receives the batch containing the "Record Tuna tx" transaction, the journal starts to check for dependencies between transactions. A batch is considered complete once all of its dependent transactions exist in the current chain or are waiting to be included into a block.
5) Batch validation: When the transaction dependencies are checked, the batch itself is tested on its validity. If the batch is valid, the "Record Tuna tx" transaction within the batch is processed by the transaction processor. Otherwise, the batch is dropped and the fisherman will be informed about the failure of his transaction.

Execution phase (5-15)

6) Transaction validation: When the batch is considered valid, the "Record Tuna tx" transaction within the batch is validated by the transaction processor. If the transaction is valid, the transaction processor will proceed to execute it.

Otherwise, the batch is dropped and the fisherman will be informed about the faulty “Record Tuna tx” transaction.

7) Transaction execution: If the “Record Tuna tx” transaction is valid, the effects of this transaction on the global state is computed by the transaction processor. However, only when the transaction is included to a block and added to the blockchain, the global state is updated.

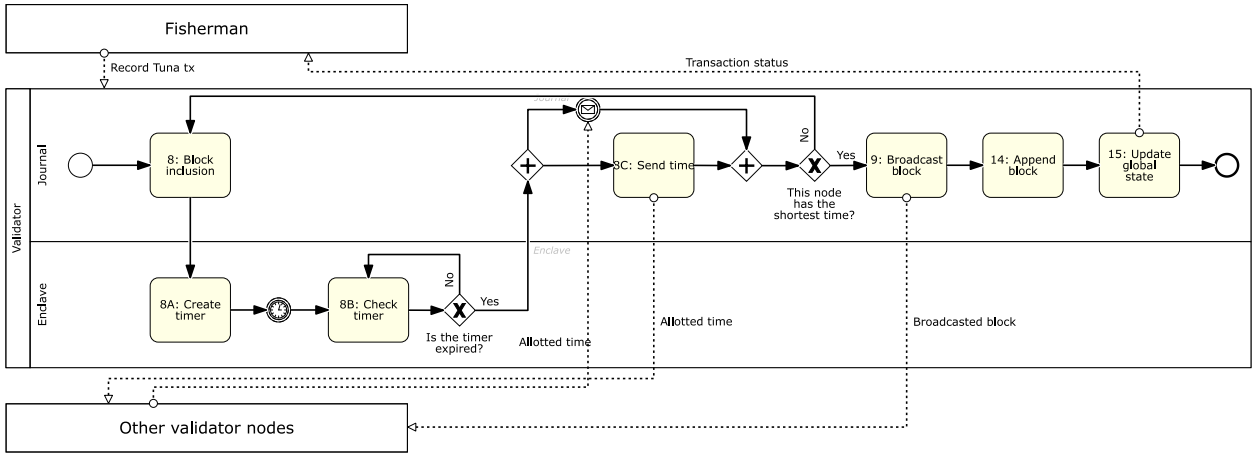


Figure 4.3: The procedure of the Proof of Elapsed Time consensus algorithm

8) Block inclusion: After the execution of a transaction, the batch will be included in a block. The Proof of Elapsed Time algorithm is used to determine which validator may broadcast its block. This procedure is shown in figure 4.3.

   8A) Create timer: Every validator creates a timer for its block by calling the create timer function in the enclave (TEE).

   8B) Check timer: The timer is checked in a beforehand determined frequency.

   8C) Send time: If the timer is expired, each of the validator nodes send each other their wait time results.

9) Broadcast block: The validator with the shortest wait time is elected to broadcast its block to the other validators. The other validators will receive the broadcasted block from the elected validator.

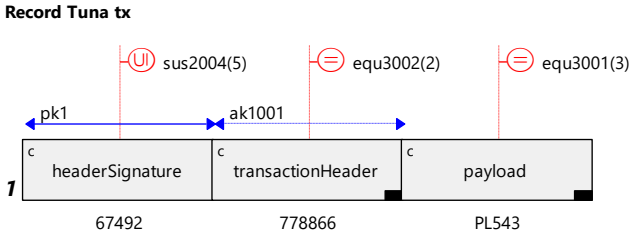
10) Check dependencies: The other validators will start to check the dependencies for the incoming block in a similar manner as with the batch dependency check.

11) Block validation: Once the dependencies of the block are checked, the block itself is tested on its validity. If the block is considered valid, the “Record Tuna tx” transaction will be validated. Otherwise, the block is dropped by the validators.

12) Transaction validation: The “Record Tuna tx” transaction is validated once more by the validators that did not win the election to ensure that the elected validator did not tamper with the transactions in its block. Note that when even only one transaction is considered as invalid, a block will be dropped.
13) Transaction execution: If the “Record Tuna tx” transaction is considered as valid by the other validators, the effects of this transaction on the global state is computed by the transaction processors of these validators.
14) Append block: The broadcasted block will now be appended to the blockchain by the nonelected validators.
15) Update global state: Each validator updates the global state based on the new sequence of blocks. The fisherman will then be informed about the success of his transaction and the tuna will now be ready for transport to the processing plant. Note that the elected validator will append its own block after he broadcasts it to the other validators and updates the global state accordingly. But only when a multitude of validators approves the broadcasted block and the global state changes it produces, the fisherman is informed.

4.3.1 Record Tuna Transaction

In Step 3 the client node sends a transaction named “Record Tuna tx” to all the validator nodes in the network. The content of this transaction is represented in a fact type diagram and can be seen in figure 4.4.



1: The transaction identified by header signature <headerSignature> contains a header with number <transactionHeader> and the payload <payload>

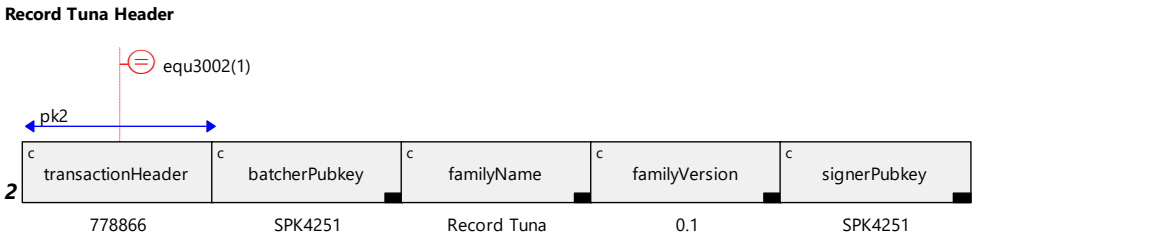
1) The transaction identified by header signature 67492 contains a header with number 778866 and the payload PL543

Figure 4.4: A Sawtooth transaction modelled as a fact type diagram showing the different components

A Sawtooth transaction consists of multiple fundamental components:

- Header Signature: a unique number created by the client signing the transaction header. Due to its uniqueness, the header signature is also called the transaction ID.
- Transaction Header: a number containing metadata about the transaction, which is illustrated in figure 4.5.

- Payload: the payload provides the input parameters for the family transaction that determine the new world state. Illustrated in figure 4.6.



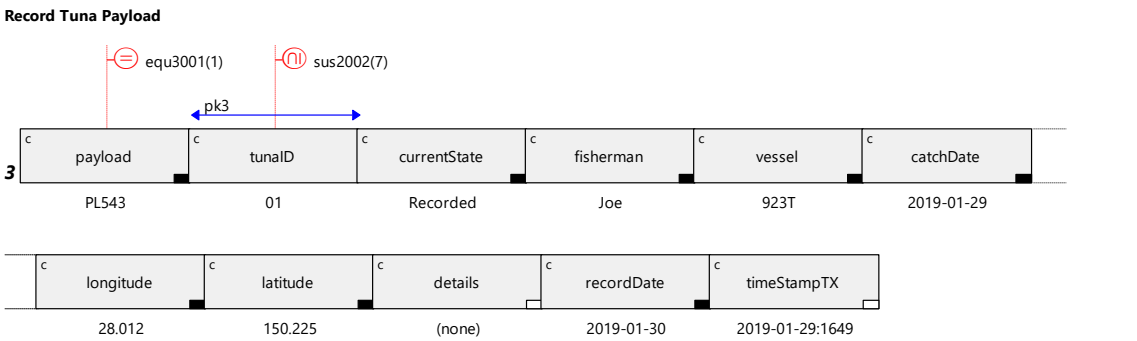
1: The transaction header identified by <transactionHeader> refers to the transaction family with name <familyName> and family version number <familyVersion>, is signed by a client with signature <signerPubkey> and put in a batch by a client with signature <batcherPubkey>

1) The transaction header identified by 778866 refers to the transaction family with name Record Tuna and family version number 0.1, is signed by a client with signature SPK4251 and put in a batch by a client with signature SPK4251

Figure 4.5: The header component of the "Record Tuna tx" transaction

Figure 4.5 shows the transaction header regarding the "Record Tuna tx" transaction. The transaction header, among other things, refers to which family name and family version should be addressed. Moreover, it shows which client added the "Record Tuna tx" transaction to a batch and which client signed the transaction. Since these clients are the same, namely the computer of the fisherman, the public keys are the same, which can be seen from the instances of the model.

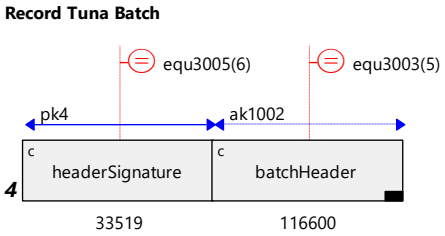
Furthermore, the "Record Tuna tx" transaction contains a payload which is illustrated in figure 4.6. As with the proposal component of a Fabric transaction, the payload component in Sawtooth is one of the most essential parts of a transaction from a business perspective. In the payload, information about the attributes of an asset is stored as key-value pairs. This information is especially useful to certify the legitimate source of a tuna. In contrast to fabric, sending a timestamp associated with the transaction is optional. On the question why timestamps are not implemented as a default part of the transaction header or a block the answer is "*using timestamps in a distributed network is troublesome, mostly due to complex clock synchronization issues among peers. You could add a timestamp in your transaction family's transaction payload*." (Hyperledger Sawtooth F.A.Q., 2019)



1: The payload <payload> sent on <timeStampTX> is identified by a tuna with ID number <tunaID> which has a current state of <currentState> and is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

Figure 4.6: The payload component of the "Record Tuna tx" transaction

As mentioned previously, the “Record Tuna tx” transaction must be wrapped inside a batch. This batch is also represented in a fact type diagram and can be seen in figure 4.7.



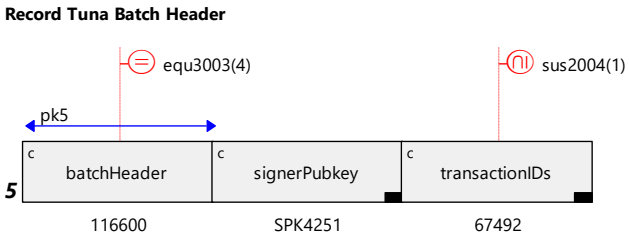
1: The batch identified by header signature <headerSignature> refers to batch header <batchHeader>

1) The batch identified by header signature 33519 refers to batch header 116600

Figure 4.7: The components of the batch that contains the “Record Tuna tx” transaction

A batch is made of two components:

- Header Signature: a unique number created by the client signing the batch header. Due to its uniqueness, the header signature is also called the batch ID.
- Batch Header: a number referring to the public key of the client that signed the batch and the transactions the batch contains. The batch header is illustrated in figure 4.8.



1: The batch with batch header <batchHeader> contains the transaction(s) <transactionIDs> and is signed by a client with signature <signerPubkey>

1) The batch with batch header 116600 contains the transaction(s) 67492 and is signed by a client with signature SPK4251

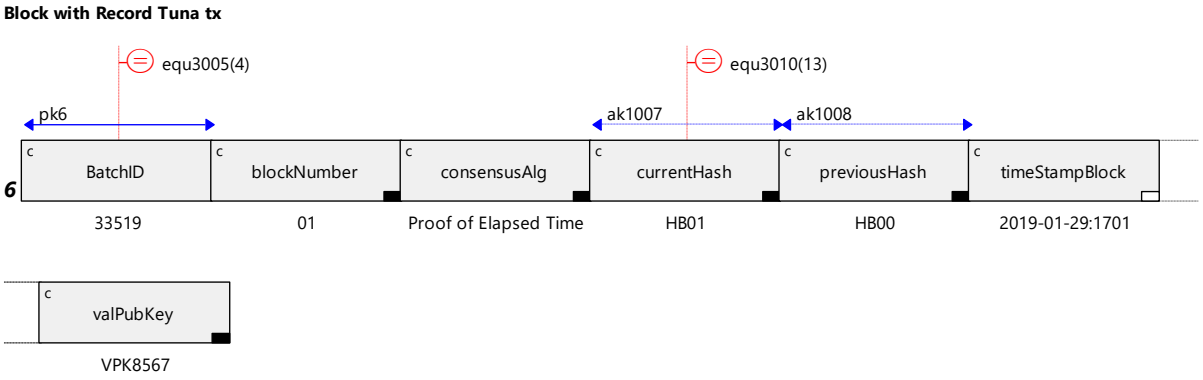
Figure 4.8: The components of the batch header for the batch that contains the “Record Tuna tx” transaction

Figure 4.8 shows that the batch is signed by a client with the public key SPK4251, this is the same client as the one which sent the transaction. This is due to the fact that the computer of the fisherman sends the transaction, puts the transaction in a batch and signs the batch. Furthermore, it shows that it contains the “Record Tuna tx” transaction, as can be seen by transaction ID number 67492. This transaction ID is the same number as the header signature of the “Record Tuna tx” transaction illustrated in figure 4.4.

4.3.2 Block and Global State: Record Tuna

Once the “Record Tuna tx” transaction is added to a block and that block is appended to the blockchain, the global state is updated accordingly. The global state is derived from the

blockchain and represents the current (latest) global state. Figure 4.9 illustrates a block that contains the “Record Tuna tx” where the global state, illustrated in 4.10, is derived from.



1: The batch identified by batch ID number <BatchID> is added to a block with block number <blockNumber> via the consensus algorithm <consensusAlg> and the block contains the current has <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the validator with signature <valPubKey>

1) The batch identified by batch ID number 33519 is added to a block with block number 01 via the consensus algorithm Proof of Elapsed Time and the block contains the current has HB01, the hash of the previous block HB00, is created on 2019-01-29:1701 and signed by the validator with signature VPK8567

Figure 4.9: A block appended to the blockchain containing the “Record Tuna tx” transaction

The component that immediately stands out in model 4.9 is the Batch ID. A Sawtooth block does not store the transaction itself, but it stores the batches which contain the transactions. A block with a certain block number can contain multiple batches or none at all, depending on how many transactions are submitted and the block size of one block. In this case, the block with block number 01 only contains one batch with one transaction, namely the “Record Tuna tx”. This batch is identified by the unique batch ID 33519 and refers the model shown figure 4.7. Furthermore, an appended Sawtooth block consists of:

- The consensus algorithm that is used to reach consensus among the validators about the sequence of blocks.
- The current hash that is produced by hashing all of the batches contained by this block.
- The previous hash, which is a copy of the hash of the previous block in the blockchain. A guarantee that this new block followed up on the then latest appended block.
- A (optional) timestamp with a specific date on which the block was created by the elected validator.
- The public key of the validator that created the block.

The current global state that is derived from this block is shown in figure 4.10. Similar to paragraph 3.2.2 and 4.2.3, where the examples of cars as digitized assets and the tuna asset in Fabric is illustrated, the tuna asset in Sawtooth is now presented. Most of the values presented in the global state are derived from the payload (figure 4.6) within the transaction component. Moreover, there is a holder that shows which entity currently holds the asset and a current state

in which the asset is in. It is important to mention that in this model a version number is shown associated with the tuna ID, in reality this would be some kind of state root hash.

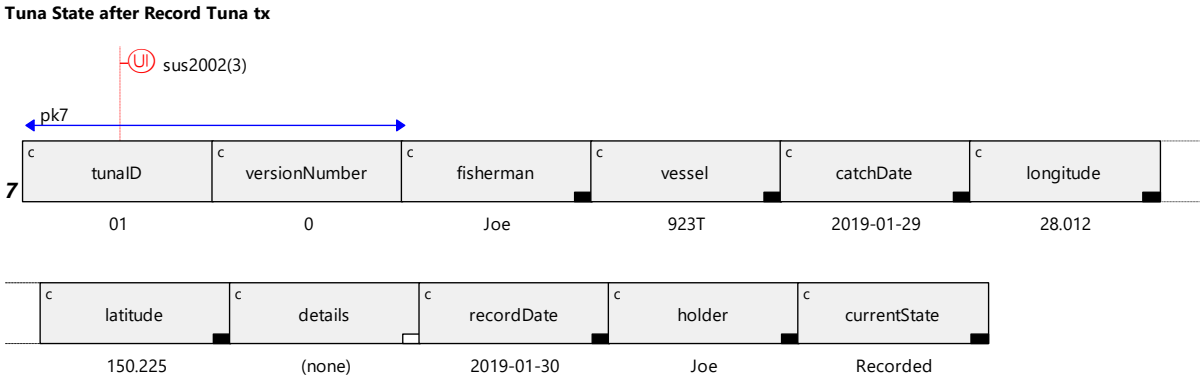


Figure 4.10: The state of the tuna after "Record Tuna tx" is validated and added to a block

In the next paragraph, the transaction flow of Processing the Tuna, where the "Processed Tuna tx" transaction is submitted, is presented in the same manner as the transaction flow of Record a Catch is presented in this paragraph. The next paragraph shows, among other things, how the transaction "Processed Tuna tx" effects the global state of the tuna asset.

4.4 Transaction flow: Processing the Tuna

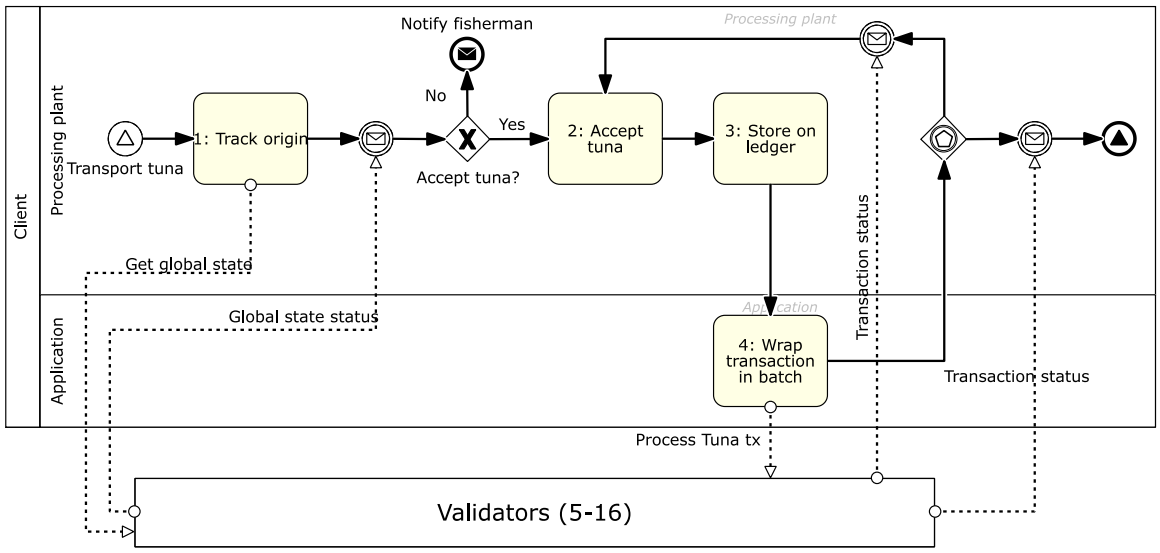


Figure 4.11: Transaction flow of the tuna asset being processed in Hyperledger

After the tuna is transported to a plant, it is ready to be processed. This transaction flow is illustrated in figure 4.11. After the transaction invocation, the tasks of the validators are similar to that of recording a catch, therefore the validators' perspective model is omitted. The sequential steps are as follows:

1) Track origin: The process starts with the processing plant verifying the origin of the received tuna. This is done by querying one of the validators with a request to give information about the received tuna. The validator should then give information about the latest state of the tuna with the corresponding ID number, this information should correspond to the tuna state showed in figure 4.10.
2) Add processing details: If the origin of the tuna is verified and the tuna state is approved by the processing plant, the processing plant can proceed to process the tuna. After the processing is done, details about the processing can be entered into the computer.

Ordering phase (3-5)

3) Store on ledger: The processing plant continues to store the entered data on the ledger by sending the data in the form of a transaction called "*Process Tuna tx*" to the validators.
4) Wrap transaction in batch: Before the "Process Tuna tx" is actually sent to the validators, the client's application wraps the transaction in a batch. Now the transaction is sent to all the validators in the network.

5,6) Check dependencies, Batch validation: The descriptions of these steps are omitted because they cover the same process as discussed in paragraph 4.3 Record a Catch, with the only difference being the name of the transaction “Process Tuna tx” and its content.

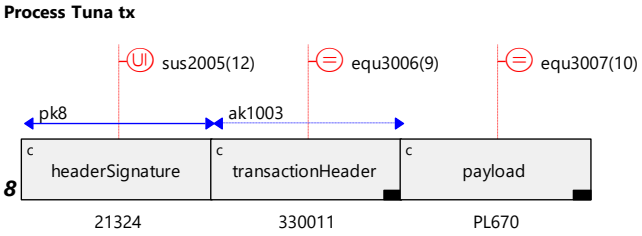
Execution phase (7-16)

7-15) Transaction validation, Transaction execution, Block inclusion, Broadcast block, Check dependencies, Block validation, Transaction validation, Transaction execution, Append block: The descriptions of these steps are omitted because they cover the same process as discussed in paragraph 4.3 Record a Catch, with the only difference being the name of the transaction “Process Tuna tx” and its content.

16) Update global state: Every validator updates the global state according to the information associated with the “Process Tuna tx” transaction that is now included in the blockchain. The processing plant will be notified by the different validators about the success of the transaction. The tuna with the digitally stored information about the processing is now ready to be transported to the restaurant. Note that in case of the processing plant rejecting the tuna asset, a transaction would be sent to the validators as well. However, this transaction would not have information about the processing, but would only contain a state update from “Recorded” to “Rejected” instead of “Processed”. Since the transaction flow of a rejected tuna is largely the same as a processed one and therefore no added value for the comparison would be gained, it was not modelled but merely noted as “notify fisherman”.

4.4.1 Process Tuna Transaction

The “Process Tuna tx” transaction is sent by the processing plant in step 3. The content of this transaction is represented in a fact type diagram and can be seen in figure 4.12.

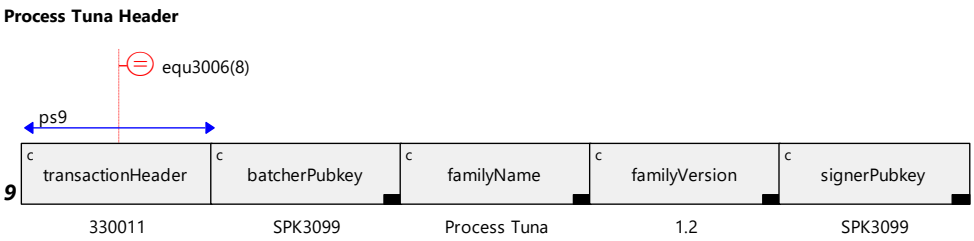


1: The transaction identified by header signature <headerSignature> contains a header with number <transactionHeader> and the payload <payload>

1) The transaction identified by header signature 21324 contains a header with number 330011 and the payload PL670

Figure 4.12: The Sawtooth transaction named "Process Tuna tx" modelled as a fact type diagram showing the different components

Figure 4.12 shows that the Process Tuna tx consists of the same three fundamental components as the "Record Tuna tx" transaction, the only difference being the different instances used.

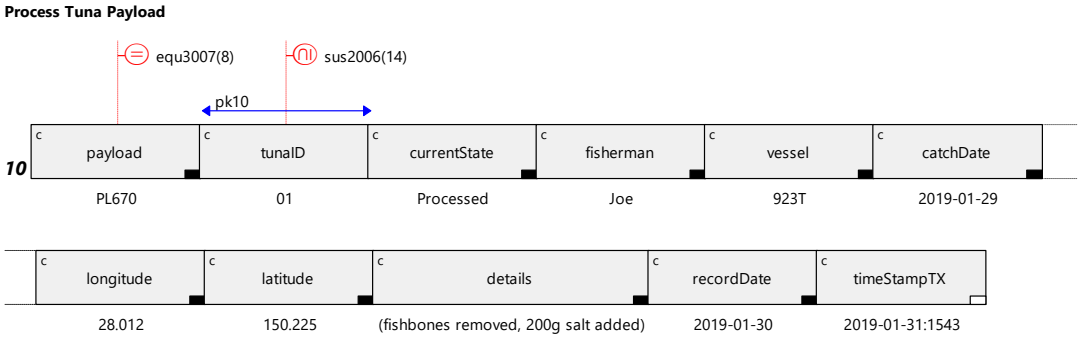


1: The transaction header identified by <transactionHeader> refers to the transaction family with name <familyName> and family version number <familyVersion>, is signed by a client with signature <signerPubkey> and put in a batch by a client with signature <batcherPubkey>

1) The transaction header identified by 330011 refers to the transaction family with name Process Tuna and family version number 1.2, is signed by a client with signature SPK3099 and put in a batch by a client with signature SPK3099

Figure 4.13: The header component of the "Process Tuna tx" transaction

The transaction header refers to some essential information about the transaction. Note that this time the "Process Tuna" family name is addressed with a corresponding family version (figure 4.13).

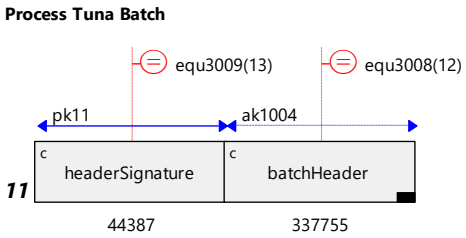


1: The payload <payload> sent on <timeStampTX> is identified by a tuna with ID number <tunaID> which has a current state of <currentState> and is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

1) The payload PL670 sent on 2019-01-31:1543 is identified by a tuna with ID number 01 which has a current state of Processed and is caught by Joe with vessel 923T on the date of 2019-01-29 at longitude 28.012 and latitude 150.225 with details (fishbones removed, 200g salt added) and is recorded on 2019-01-30

Figure 4.14: The payload component of the Process Tuna tx transaction

The payload of "Process Tuna tx" (figure 4.14) largely contains the same information as the transaction "Record Tuna tx", except that the processing plant's data entry can now be seen as an instance in the details field and the timestamp of this transaction is of course of a different time and date.

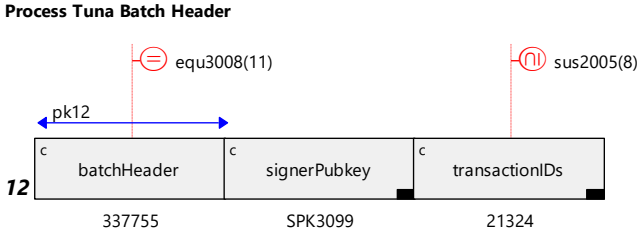


1: The batch identified by header signature <headerSignature> refers to batch header <batchHeader>

1) The batch identified by header signature 44387 refers to batch header 337755

Figure 4.15: The components of the batch that contains the "Process Tuna tx" transaction

The batch that contains the "Process Tuna tx" transaction is shown in figure 4.15. This batch also consists of the same two fundamental components as the batch that contains the transaction "Record Tuna tx".



1: The batch with batch header <batchHeader> contains the transaction(s) <transactionIDs> and is signed by a client with signature <signerPubkey>

1) The batch with batch header 337755 contains the transaction(s) 21324 and is signed by a client with signature SPK3099

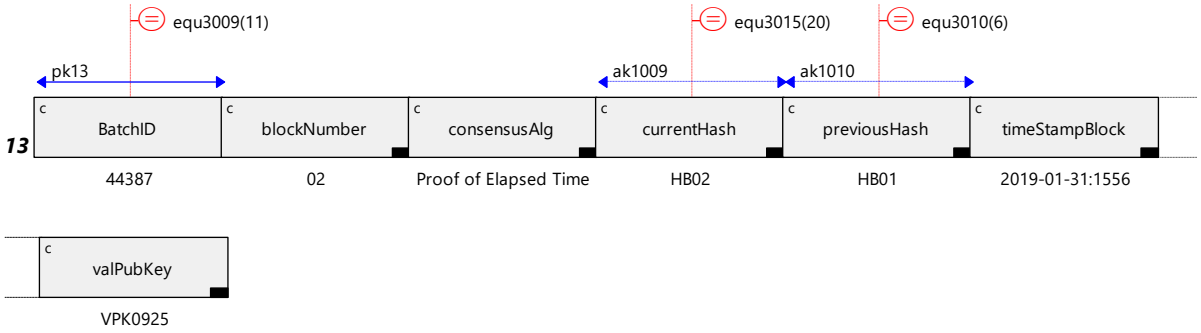
Figure 4.16: The components of the batch header for the batch that contains the Process Tuna tx transaction

Figure 4.16 shows that this batch contains the "Process Tuna tx" transaction as one can see by the transaction ID number 21324 that corresponds with the header signature shown in 4.12.

4.4.2 Block and Global State: Process Tuna

The block that is created once the tuna is processed and "Process Tuna tx" is added to a block can be seen in figure 4.17. This block also shows that not the transactions, but the batches are stored in the block as is visible by the batch ID instance that corresponds with the header signature instance of figure 4.15. Furthermore it is visible that a sequence of blocks is created due to the fact that the previous hash of the block with block number 02 corresponds with the current hash of block number 01 shown in figure 4.9.

Block with Process Tuna tx



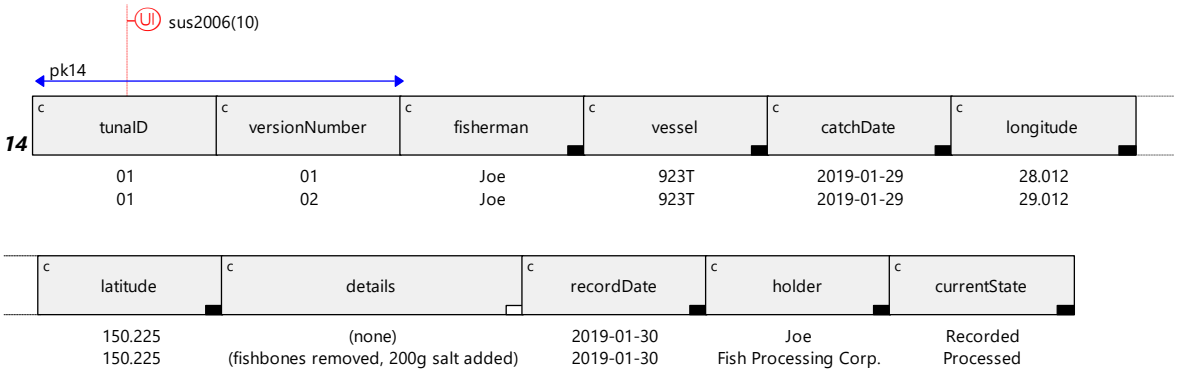
1: The batch identified by batch ID number <BatchID> is added to a block with block number <blockNumber> via the consensus algorithm <consensusAlg> and the block contains the current has <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the validator with signature <valPubKey>

1) *The batch identified by batch ID number 44387 is added to a block with block number 02 via the consensus algorithm Proof of Elapsed Time and the block contains the current has HB02, the hash of the previous block HB01, is created on 2019-01-31:1556 and signed by the validator with signature VPK0925*

Figure 4.17: A block appended to the blockchain containing the "Process Tuna tx" transaction

The effect this block has on the global state of the tuna is illustrated in figure 4.18. This model shows that the tuna changed of ownership, visible via the instance change of the holder field. Moreover, the state of the tuna has been changed from "Recorded" to "Processed" and the version number is increased from 01 to 02. In reality, this field would contain some state root hash that would change. The rest of the fields remain the same, since no changes are made to them in the "Processing Tuna tx" transaction.

Tuna State after Process Tuna tx



1: The state of the tuna identified by the combination of ID number <tunaID> and version number <versionNumber> is recorded by <fisherman> on <recordDate>, is currently hold by <holder>, has the following details <details>, is caught with vessel <vessel> on <catchDate> at longitude <longitude> and latitude <latitude> and its current state is <currentState>

1) *The state of the tuna identified by the combination of ID number 01 and version number 01 is recorded by Joe on 2019-01-30, is currently hold by Joe, has the following details (none), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Recorded*
2) *The state of the tuna identified by the combination of ID number 01 and version number 02 is recorded by Joe on 2019-01-30, is currently hold by Fish Processing Corp., has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 29.012 and latitude 150.225 and its current state is Processed*

Figure 4.18: The state of the tuna after "Process Tuna tx" is validated and added to a block

4.5 Transaction flow: Accepting the Tuna

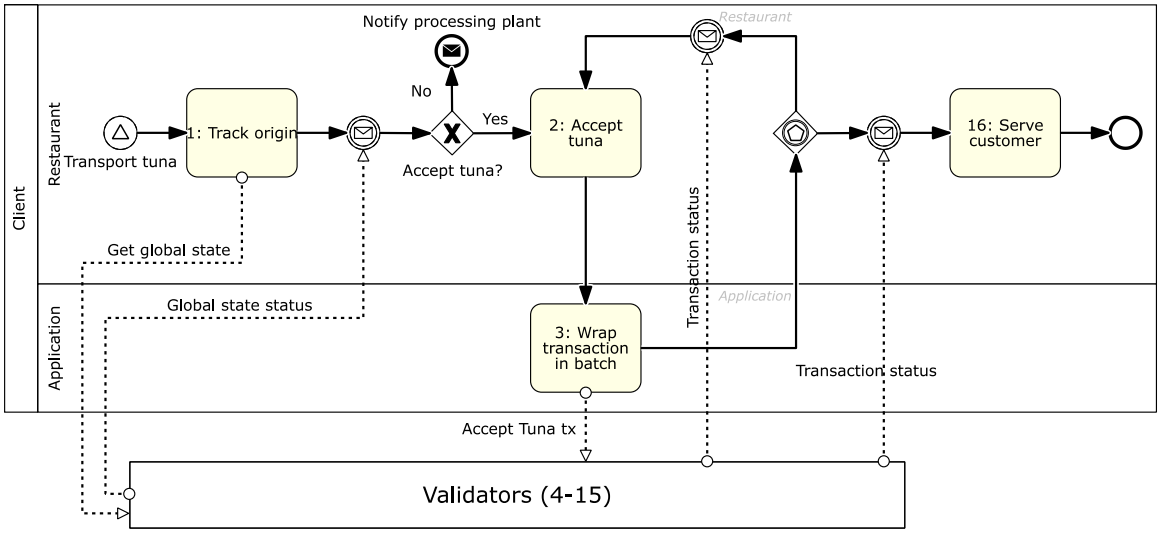


Figure 4.19: Transaction flow of the tuna asset being accepted in Hyperledger Sawtooth

After the tuna is processed, it is transported to different vendors. In this situation, the tuna is sent to a restaurant which will prepare and serve it to their customers. The processing of the transaction within the validators is identical to processing of the "Record Tuna tx" and the "Process Tuna tx" transactions and therefore the validators' perspective model is omitted. The "Accept tuna tx" transaction flow is illustrated in figure 4.19. The sequential steps are as follows:

1) Track origin: The process starts with the restaurant checking the legitimacy of the tuna by verifying the origin of the received tuna. This is done in the same manner as with the "Process Tuna tx". The client queries one of the validators with a request to give information about the tuna. The validator should then give information about the latest state of the tuna with the corresponding ID number, this information should correspond to the tuna state showed in figure 4.18.
2) Add processing details: The restaurant will be the tuna's last stop, therefore it only needs to be accepted, that is, if the state of the tuna is approved by the restaurant. In contrary to the previous two transactions, the "Accept Tuna tx" does not require a data entry by the client. When the restaurant accepts the tuna, the "*Accept Tuna tx*" transaction will automatically be sent to the validators.

Ordering phase (3-5)

3) Wrap transaction in batch: Before the "Accept Tuna tx" is actually sent to the validators, the client's application wraps the transaction in a batch. Now the transaction is sent to all the validators in the network.

4,5) Check dependencies, Batch validation: The descriptions of these steps are omitted because they cover the same process as discussed in paragraph 4.3 Record a Catch and 4.4 Processing the Tuna, with the only difference being the name of the transaction "Accept Tuna tx" and its content.

Execution phase (6-15)

6-14) Transaction validation, Transaction execution, Block inclusion, Broadcast block, Check dependencies, Block validation, Transaction validation, Transaction execution, Append block: The descriptions of these steps are omitted because they cover the same process as discussed in paragraph 4.3 Record a Catch and 4.4 Processing the Tuna, with the only difference being the name of the transaction "Accept Tuna tx" and its content.

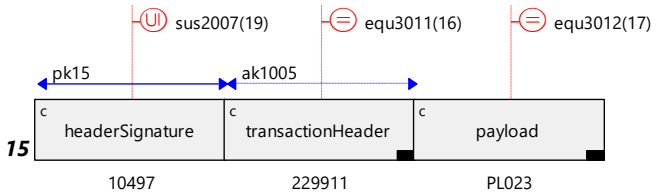
15) Update global state: Every validator updates the global state according to the information associated with the "Accept Tuna tx" transaction that is now included in the blockchain. The restaurant will be notified by the different validators about the success of the transaction. The tuna is now ready to be served to customers.

16) Serve customer: The restaurant serves tuna with ID 01 to a customer. This ends the lifecycle of this asset.

4.5.1 Accept Tuna Transaction

The "Accept Tuna tx" transaction is sent by the restaurant in step 2. The content of this transaction is represented in a fact type diagram and can be seen in figure 4.20.

Accept Tuna tx



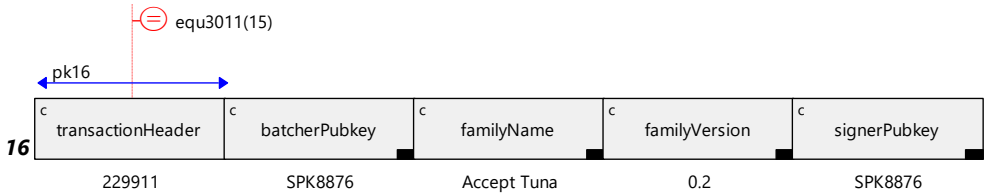
1: The transaction identified by header signature <headerSignature> contains a header with number <transactionHeader> and the payload <payload>

1) The transaction identified by header signature 10497 contains a header with number 229911 and the payload PL023

Figure 4.20: The Sawtooth transaction named "Accept Tuna tx" modelled as a fact type diagram showing the different components

Figure 4.20 illustrated the "Accept Tuna tx" and shows that it consists of the same three fundamental components as the previous two transaction, except that different instances are being used.

Accept Tuna Header



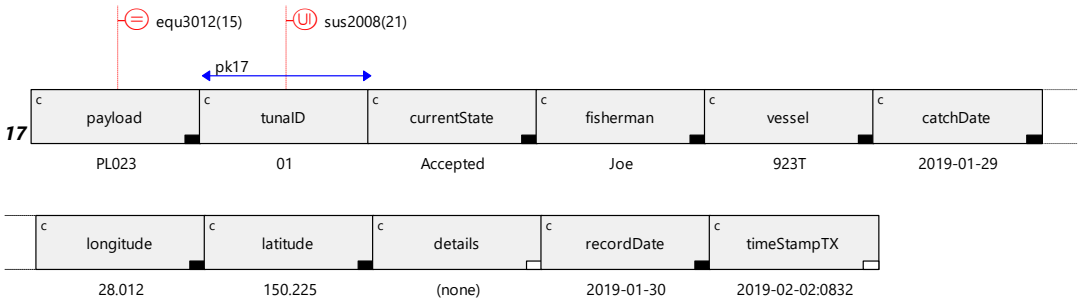
1: The transaction header identified by <transactionHeader> refers to the transaction family with name <familyName> and family version number <familyVersion>, is signed by a client with signature <signerPubkey> and put in a batch by a client with signature <batcherPubkey>

1) The transaction header identified by 229911 refers to the transaction family with name Accept Tuna and family version number 0.2, is signed by a client with signature SPK8876 and put in a batch by a client with signature SPK8876

Figure 4.21: The header component of the "Accept Tuna tx" transaction

The transaction header refers to information about the transaction and note that this time the Accept Tuna is addressed in the family name (figure 4.21).

Accept Tuna Payload

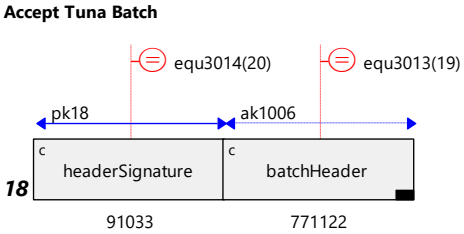


1: The payload <payload> sent on <timeStampTX> is identified by a tuna with ID number <tunaID> which has a current state of <currentState> and is caught by <fisherman> with vessel <vessel> on the date of <catchDate> at longitude <longitude> and latitude <latitude> with details <details> and is recorded on <recordDate>

1) The payload PL023 sent on 2019-02-02:0832 is identified by a tuna with ID number 01 which has a current state of Accepted and is caught by Joe with vessel 923T on the date of 2019-01-29 at longitude 28.012 and latitude 150.225 with details (none) and is recorded on 2019-01-30

Figure 4.22: The payload component of the "Accept Tuna tx" transaction

The payload of "Accept Tuna tx" (figure 4.22) contains the same information as the previous two transactions since no new data entry is made. The only differences are that the state of the tuna is changed from "Processed" to "Accepted" and the timestamp of this transaction is of a different time and date.

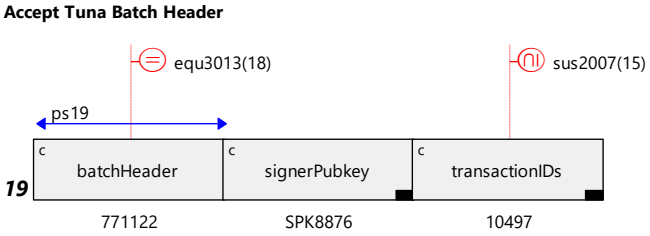


1: The batch identified by header signature <headerSignature> refers to batch header <batchHeader>

1) The batch identified by header signature 91033 refers to batch header 771122

Figure 4.23: The components of the batch that contains the "Accept Tuna tx" transaction

The batch containing the "Accept Tuna tx" transaction can be seen in figure 4.23. This batch also consist of the same two fundamental components as the previous two batches.



1: The batch with batch header <batchHeader> contains the transaction(s) <transactionIDs> and is signed by a client with signature <signerPubkey>

1) The batch with batch header 771122 contains the transaction(s) 10497 and is signed by a client with signature SPK8876

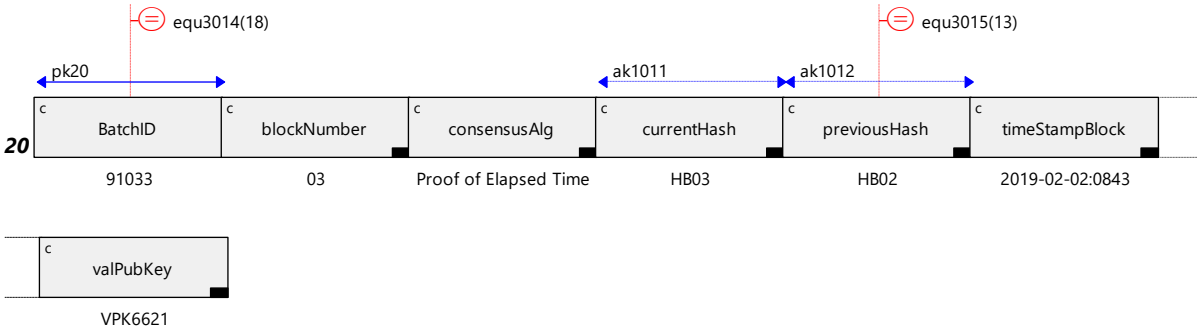
Figure 4.24: The components of the batch header for the batch that contains the "Accept Tuna tx" transaction

Figure 4.24 shows that this batch contains the "Accept Tuna tx" transaction as one can see by the transaction ID number 10497 that corresponds with the header signature shown in 4.20.

4.5.2 Block and Global State: Accept Tuna

The block containing the "Accept Tuna tx" can be seen in figure 4.25. Although the figure shows that not the transaction, but the batch is stored in the block. This is visible by the batch ID instance "91033" that corresponds with the header signature instance of figure 4.23. Moreover, the sequence of blocks is continued and is seen by the fact that the previous hash of the block with block number 03 corresponds with the current hash of block number 02 shown in figure 4.17.

Block with Accept Tuna tx



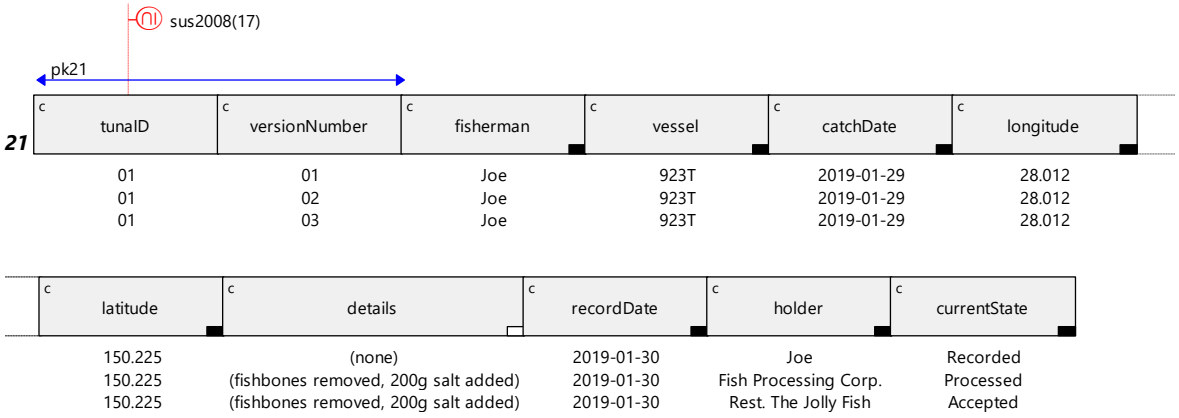
1: The batch identified by batch ID number <BatchID> is added to a block with block number <blockNumber> via the consensus algorithm <consensusAlg> and the block contains the current has <currentHash>, the hash of the previous block <previousHash>, is created on <timeStampBlock> and signed by the validator with signature <valPubKey>

1) *The batch identified by batch ID number 91033 is added to a block with block number 03 via the consensus algorithm Proof of Elapsed Time and the block contains the current has HB03, the hash of the previous block HB02, is created on 2019-02-02:0843 and signed by the validator with signature VPK6621*

Figure 4.25: A block appended to the blockchain containing the "Accept Tuna tx" transaction

The fact type diagram in figure 4.26 shows the effect the block in figure 4.25 has on the global state. The tuna changed of ownership once more, visible via the instance change of the holder field. Furthermore, the state of the tuna has been changed from "Processed" to "Accepted" and the version number is increased from 02 to 03.

Tuna State after Accept Tuna tx



1: The state of the tuna identified by the combination of ID number <tunaID> and version number <versionNumber> is recorded by <fisherman> on <recordDate>, is currently hold by <holder>, has the following details <details>, is caught with vessel <vessel> on <catchDate> at longitude <longitude> and latitude <latitude> and its current state is <currentState>

1) *The state of the tuna identified by the combination of ID number 01 and version number 01 is recorded by Joe on 2019-01-30, is currently hold by Joe, has the following details (none), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Recorded*
2) *The state of the tuna identified by the combination of ID number 01 and version number 02 is recorded by Joe on 2019-01-30, is currently hold by Fish Processing Corp., has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Processed*
3) *The state of the tuna identified by the combination of ID number 01 and version number 03 is recorded by Joe on 2019-01-30, is currently hold by Rest. The Jolly Fish, has the following details (fishbones removed, 200g salt added), is caught with vessel 923T on 2019-01-29 at longitude 28.012 and latitude 150.225 and its current state is Accepted*

Figure 4.26: The state of the tuna after "Accept Tuna tx" is validated and added to a block

Chapter 5
Comparison Analysis

5.1 Introduction

This chapter presents an comparison analysis based on the information provided in chapter 3 and chapter 4. This analysis is presented via a meta-model that provides a direct overview of the overlapping concepts (similarities) used between both projects. The differences between the platforms are pointed out by presenting models of this meta-model applied to both platforms and by modelling the interrelationships of the instances. Via these models, the answers to the research questions are given.

5.2 Overarching principles

Distinctive platform processes are presented in Chapter 3 and 4, nevertheless there are a lot of similarities visible between the two projects. Therefore, the following overarching description applies to both Hyperledger Fabric and Hyperledger Sawtooth.

Hyperledger Fabric and Hyperledger Sawtooth main goal is to create trust between untrusting parties in a business context. Therefore the network consists of beforehand known participating organizations that operate the peers divided among them. The creation of trust is accomplished via a decentralized network consisting of peers/nodes that needs a level of agreement among the nodes, called consensus, in order to verify and validate information between them. The network has the functionality to digitize assets. These digital assets can be stored on the network by creating and submitting transactions. In order to understand the logic within the transactions, the network needs some kind of business logic processing functionality. In the network a separation is made between 1) the sequence of blocks called a “blockchain”, which contains all the transactions; and 2) the information that is derived from this blockchain, called the state. Every digital asset has a state attributed to it. Transactions are used to alter the digital assets (data objects). When one speaks about the alteration of a digital asset, it actually means that the state of the digital asset is altered. A transaction can invoke a certain already existing application on the network that will run once it is addressed by that transaction. It is these applications that one refers to when discussing smart contracts. Furthermore, the

information needed to fill in the variables for a certain application are also included in the transactions.

From this description, 7 overarching principles applicable to both platforms are derived and presented in table 5.1.

Hyperledger	
Fabric	**Sawtooth**
Creating trust between untrusting parties	
Distributed peers	
Consensus among peers	
Business logic comprehension	
Digitization of assets	
Separation of blockchain and state	
State management via transactions	

Table 5.1: Overarching principles between Hyperledger Fabric and Hyperledger Sawtooth

5.3 Meta-model

Although these 7 principles are applicable to both projects, the implementation to accomplish their overarching goal is indeed different. In table 5.2, a brief overview of the differences between the two platforms is given. These concepts are chosen based on the fact that they are fundamental components of both platforms but the implementation differs from one another.

Hyperledger		
	Fabric*	**Sawtooth***
Peer type	Different peer types	One peer type
Consensus	Consensus as a Process	Consensus as an Algorithm
Business logic functionality	Chaincode	Transaction Family
State management	World State	Global State
Peer tasks	Division of tasks	Each peer does all tasks

* Production default setting

Table 5.2: Differences between Hyperledger Fabric and Sawtooth in a production default setting

The components are fundamental because every Hyperledger Fabric or Sawtooth network consists of distributed peer types (nodes) that have 1) certain tasks and 2) agreement about

information on the network via consensus. Furthermore, in order to be able to process transactions and with that, digitize the assets, a certain business logic functionality needs to exist on the network. Lastly, because the state of the digitized asset is managed, the asset is trackable, which is in the end the main goal of the cooperating enterprises. State management also implies a separation of a blockchain and a state.

When the overlapping concepts of table 5.2 are presented in a simple meta-model, figure 5.1 is the result.

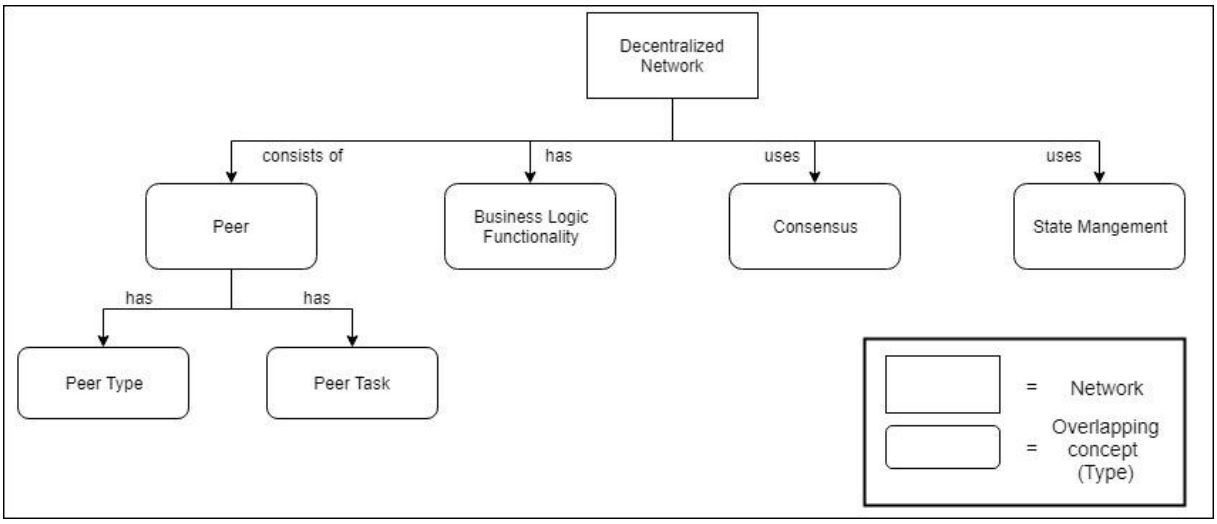


Figure 5.1: Meta-model of Hyperledger and Sawtooth

While the model in figure 5.1 provides indeed a clear overview of the general concepts in both the Hyperledger projects, it does not express any of the interrelationships between the concepts. For this, first the instances of both networks are needed to fill in the overlapping concepts, then the instances can be related to one another. This will be done for the two platforms in the upcoming two paragraphs.

5.3.1 Hyperledger Fabric Interrelationships

Figure 5.2 shows the instances of Fabric applied to the concepts of the model illustrated in figure 5.1.

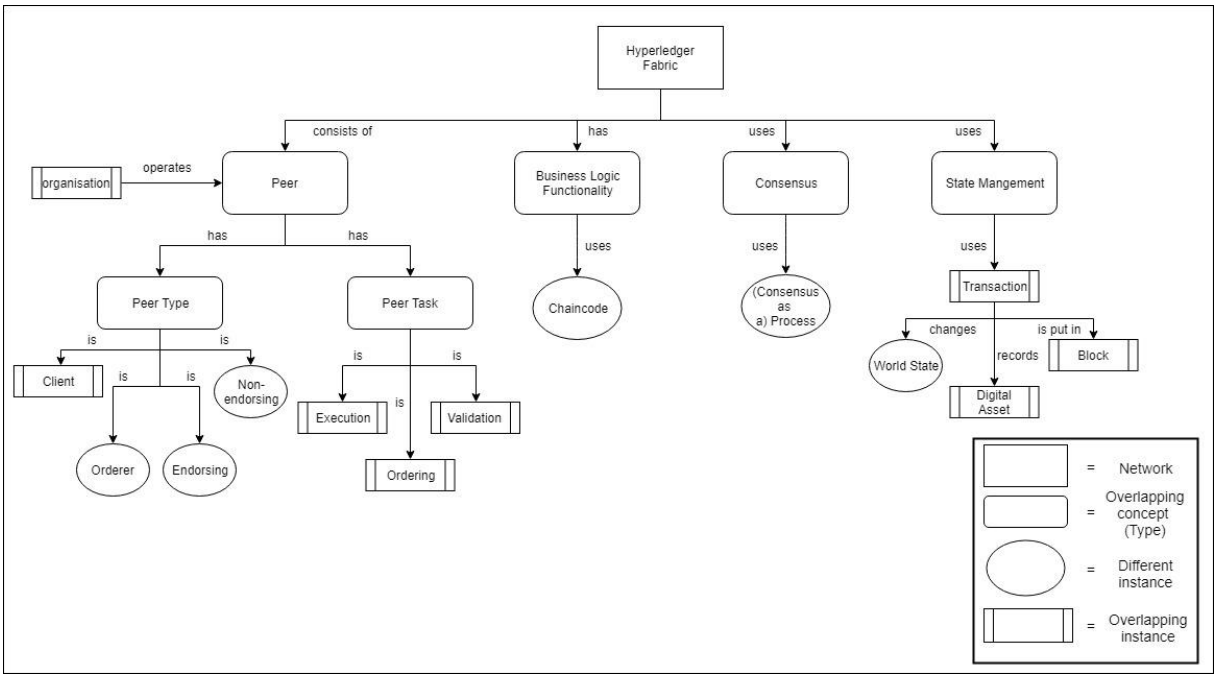


Figure 5.2: Meta-model of figure 5.1 applied to Hyperledger Fabric

Figure 5.2 clearly shows which instances belong to which type, however, it still does not explain the interrelationships of the instances. Therefore the expressivity of this model can be considered as minimal, besides the overview it provides of Hyperledger Fabric. To show the interrelationships of the instances, the overlapping concepts are omitted and only the instances and their relationships are modelled. The result can be seen in figure 5.3.

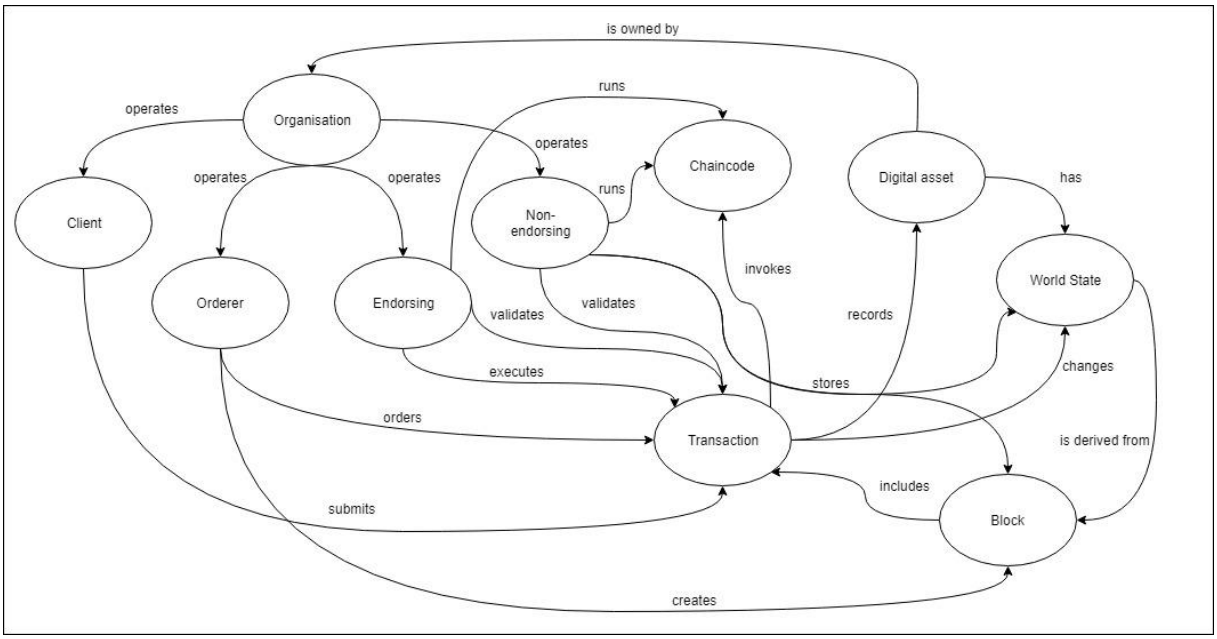


Figure 5.3: A model of Hyperledger Fabric's instances interrelationships

The justification for this interrelationships model is as follows. An organisation operates different peers in the network, depending on the roles divided between the enterprises when the network was launched. In reality, this does not mean that every organisation operates each peer. For example, in the tuna fishing use case it is not needed that a notary firm operates a client peer, but it can operate an orderer to safeguard the process of block creation and ordering. However, the organisations that have a direct influence on the state of the digital asset will need a client peer to submit transactions that can change the digital asset's state. All these peers have a certain relationship with the transaction instance. A transaction is submitted, executed, ordered and validated by the different type of peers, which makes the division of tasks visible. Furthermore, a transaction invokes chaincode in order to comprehends what to do with the information inside the transaction. From this transaction, the digital asset is recorded or the world state associated with that digital asset changed. This world state is derived from the sequence of blocks and stored by a non-endorsing peer. A block includes the transactions submitted to the network, is created by an orderer and is also stored by a non-endorsing peer. Lastly, the digital asset is owned by the organisation that submitted the transaction associated with that digital asset.

This model immediately shows that the transaction is the key component of the network and that a transaction is processed by different types of peers. An observant reader may also have noticed that one instance shown in figure figure 5.2, is not present in figure 5.3, that is the consensus instance. This is due to the fact that consensus is reached via an intangible process instead of a single algorithm being used (as is explained in paragraph 3.2.3). Therefore, the consensus is implicitly embedded in the process of execution, ordering and validation and not present as an instance in figure 5.3.

In the next paragraph, similar models are presented for Hyperledger Sawtooth.

5.3.2 Hyperledger Sawtooth Interrelationships

Figure 5.4 shows the instances of Sawtooth applied to the concepts of the meta-model illustrated in figure 5.1.

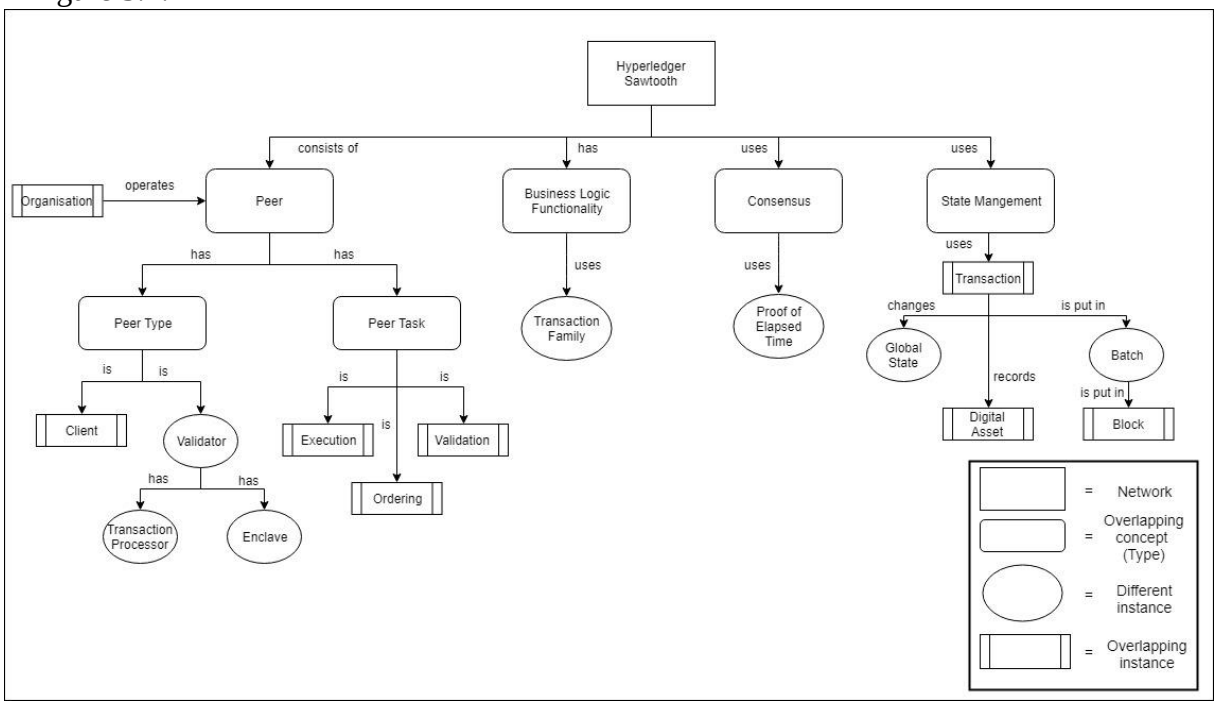


Figure 5.4: Meta-model of figure 5.1 applied to Hyperledger Sawtooth

Figure 5.4 provides an overview of Sawtooth's instances attached to the overlapping concepts, but the expressivity of this model is minimal, due to the fact that the interrelationships of the instances are not visible. To show the interrelationships between the instances, the overlapping concepts are omitted and only the instances and their relationships are modelled. The result can be seen in figure 5.5.

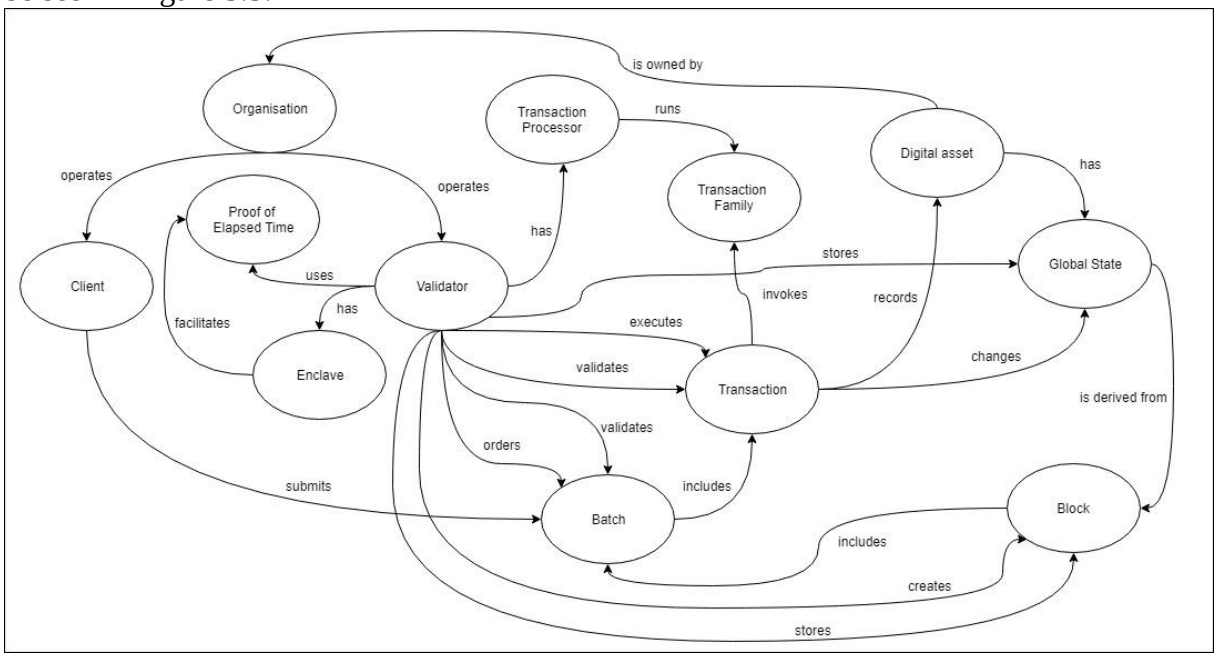


Figure 5.5: A model of Hyperledger Sawtooth's instances interrelationships

The justification for this interrelationships model is as follows. There are only two types of peers. Most organisations operate a client peer and preferably a validator peer. However, similar to Fabric, this does not mean that every organisation operates each peer. Again, in the tuna fishing use case it is not needed that a notary firm operates a client peer, but it can operate a validator peer to enhance the decentralization of peers and thus making the network less vulnerable. The client peer submits a batch that has inside it a transaction. Ordering, execution and validation are all done by the validator peer. There is no division of tasks, in contrary to Fabric. The validator peers agree with one another by using a consensus algorithm called Proof of Elapsed time, which is made possible by the enclave inside the validator peer (see paragraph 4.2.2). The submitted batch is ordered by the validator, both the batch and transaction are validated, but only the transaction can be executed. This transaction invokes a certain Transaction Family that is run by the transaction processor and enables the validator peer to comprehend the business logic inside the transaction. Furthermore, the transaction can record a digital asset or change the global state of a digital asset. That global state is derived from the sequence of blocks and is stored by a validator peer. A block includes the batches submitted to the network, is created by a validator and is also stored by a validator peer. Lastly, the digital asset is owned by the organisation that submitted the transaction associated with that digital asset.

This model shows that the transaction, just as with Fabric, is also the key component of the network. Moreover, it is visible that the validator peer is responsible for all the processing phases.

5.4 Transactions, Blocks and States compared

Based on the information provided in chapter 3 and 4, paragraph 5.3 gives a direct overview of the different elements in both networks and their responsibilities in regard to the processing of a transaction (also called the transaction flow). Furthermore, it shows that the transaction is the key component of both networks and that the end result, a digital asset's state that is agreed on by different peers, is derived from the sequence of blocks. But is the structure of a transaction significantly different from one another and does this influence the transaction flow? While significantly is hard to define in this context, one could argue that there are indeed differences to be found in the way transactions are structured in Fabric and Sawtooth. Paragraph 3.3.1, 3.4.1 and 3.5.1 showed the three discussed transactions for Fabric and paragraph 4.3.1, 4.4.1, and 4.5.1 for Sawtooth. The fundamental components each discussed transaction consisted of is presented for both networks in table 5.3.

Hyperledger	
Fabric	**Sawtooth**
Transaction ID	Header Signature
Header	Transaction Header
Proposal	Payload
Client Signature	Signer Public key*
Response	
Endorsements	

*Component of the Transaction Header

Table 5.3: Fundamental components of Hyperledger Fabric's and Sawtooth's transactions

Fabric's Transaction ID corresponds with Sawtooth's Header Signature; Fabric's Header corresponds with Sawtooth's Transaction Header; Fabric's Proposal corresponds with Sawtooth's Payload and Fabric's Client Signature can be found in Sawtooth as a component of the Transaction Header called Signer Public key. The other fundamental transaction components of Fabric are Response and Endorsements and these are not present in Sawtooth. However, this is easily explainable since both Response and Endorsements contain information that is needed for verification purposes caused by the execute-order-validate design that Fabric uses (see paragraph 3.2.4). These components have no further consequences. What these paragraphs further showed was that the intent of the transaction had no influence on the structure of the transaction, but only changed the information inside the Proposal for Fabric or

Payload for Sawtooth and invoked a different chaincode or transaction family. To conclude, the structure of a transaction does not influence the transaction flow, one could even argue the opposite. The design choices for the network (such as a division of tasks or the use of batches) could influence a transaction's structure.

Besides the transaction's structure, the multiple blocks and their structures are also presented in chapter 3 and 4. The block's structures of both platforms are almost identical. The noteworthy differences are that in Hyperledger Sawtooth not the transaction, but a batch is included in a block and that information about the used consensus algorithm is given. Since Fabric does not make use of a single algorithm, information about the consensus algorithm is, logically, not included in a Fabric block. Paragraph 3.3.2, 3.4.2, 3.5.2, 4.3.2, 4.4.2 and 4.5.2 all show that the hashes of the previous block is incorporated in the next block, creating a sequence of blocks called the blockchain.

More importantly, the Fact Type Diagrams in these paragraphs show how information about the state of a digital asset is presented and derived from these blockchains, which makes the separation of blockchain and state visible on a data structured level. Although the technical implementation of the data storing in the databases is different between the two platforms (and not within the scope of this thesis to discuss any further), both projects present the information about a digital asset in a key-value form (state). Therefore it can be concluded that World State and Global State describe the exact same concept, namely: the storing of information in a database, presented in a key-value form, providing the state for every digital asset. Thus, the use of World State and Global State is only different in name (see paragraph 3.2.1 and 4.2.1 for more information).

5.5 Differences in the Transaction Flows

The goal of this thesis is to gain insights in the blockchain phenomenon in a scientific manner by unraveling the mysteries surrounding it and by doing so, making it more understandable for businesses. To accomplish this, an exploration is done on the most fundamental process of two private blockchain networks, the processing of transactions or the so called transaction flow.

To answer the question to what extent the transaction flows of Hyperledger Fabric and Hyperledger Sawtooth differ, one should look at the network design choices. Based on chapter 3 and chapter 4, it can be concluded that a transaction flow consists of three essential phases (or tasks), namely: ordering, execution and validation. All these three tasks are present in both platforms. However, most fundamental differences between the two projects (as seen in table 5.2, figure 5.3 and figure 5.5) originate from the division of tasks. By creating different types of peers and dividing these three essential tasks among these peers, as Fabric does, the transaction flow is deliberately changed. Furthermore, the tasks are not only divided, but the order of the tasks is changed as well. Fabric chooses to first execute a transaction, then order the transaction and in the end to validate the transaction. Sawtooth chooses to stick to the more traditional design, where a transaction is first ordered, then executed and in the end validated. In that sense, Sawtooth looks very similar to Bitcoin and Ethereum by using one type of peer responsible for every task without changing the order of the tasks.

A follow-up question one could ask is if this difference in task order has any consequences when using one of the platforms. From a technical perspective, it most certainly has. Although not researched in this thesis, variables that are probably influenced by these choices are for example network speed and transaction throughput. However, from an information perspective for businesses, the two platforms act almost the same. Paragraph 3.3.2, 3.4.2, 3.5.2, 4.3.2, 4.4.2 and 4.5.2 all showed the information in a similar manner. Therefore, it can be concluded that the end product of both platforms is "*trustworthy, trackable information on the state of a digital asset*"; that both these platforms present this information in the same manner and that the route to this trusted information is different due to the differences in peer types and task order.

Chapter 6
Conclusion

In this thesis a comparison between the blockchain platforms Hyperledger Fabric and Hyperledger Sawtooth was made. The goal of this thesis was to gain insights in the blockchain phenomenon in a scientific manner and to make the technologies understandable for businesses, as this need is ever increasing (de Kruijff & Weigand, 2017). In order to achieve this goal, a comparative study was chosen, focusing on the transaction flow as the most fundamental process of a blockchain platform. This was conducted via a conceptual modelling approach that helped to systematically identify and describe the main concepts up for comparison and to carry out the comparison, resulting in meaningful similarities and differences between the platforms. As formal conceptual modelling method CogNIAM was used.

The literature showed that determination of the type of blockchain is important before a comparison between blockchains is made, since the goals per type will differ. It was concluded that Hyperledger Fabric and Hyperledger Sawtooth both are, primarily, private permissioned blockchains and therefore the network consists of identified participants. Both blockchain platforms are used to secure interactions among a group of entities that have common goals but which may not fully trust each other. With this as background, a simplified tuna fishery supply chain use case was presented for both networks.

The use case showed that both platforms have the potential to digitize assets; that there is a separation of the blockchain and state and that the state is managed via transactions submitted to the network. As answer to RQ1.4, the end goal of a transaction is therefore to manage the state of a digital asset. The data structure of transactions and blocks had no significant differences (answering RQ1.3), as well as the presentation of digital asset's state. Furthermore, answering RQ1.1, Hyperledger Fabric sees consensus as a process and therefore implemented no specific algorithm, in contrary to Hyperledger Sawtooth that implemented the Proof of Elapsed Time algorithm. Elements that influence the transaction flow are harder to define (RQ1.2), since the transaction flow is subject to a multitude of factors. However, it was concluded that especially the peer types and task order heavily influences the transaction flow. This also answers RQ1, the implementation of Hyperledger Fabric's different peer types in comparison to Hyperledger Sawtooth's single peer type causes changes in the processing of the

transactions or the so called transaction flow. These changes are primarily in the order of the tasks and the responsibilities of the (different) peers.

To conclude, although the technical implementation is different, from a business information perspective the two platforms act almost the same. The end product of both platforms is trustworthy, trackable information on the state of a digital asset; both these platforms present this information in the same manner but the route to this trusted information is different due to the differences in peer types and task order.

Chapter 7
Discussion

In regard to the conclusions presented in chapter 6, a few remarks about the limitations of this research need to be made.

The first limitation of this research is the scope, since the focus was primarily on the transaction flow, other (smaller) features that might be useful when implementing a blockchain in an enterprise context were overlooked. For example, the use of channels within a Hyperledger Fabric blockchain seems a promising feature to increase anonymity on the network. This can be important for businesses that want or need some details of a transaction to be invisible for other participants of the network. However, more research is needed on the technical implications of such channels, especially in regard to network speed and the possibility of bloating the network. In addition to that, people are working on other solutions for increased anonymity, for example the use of zero-knowledge proofs which allows a prover to convince others that a certain statement is true, without revealing any additional information (Benhamouda, Halevi, & Halevi, 2018).

The second limitation of this thesis is the focus on a business perspective. As previously mentioned in paragraph 5.5, other fundamental variables can be taken into account when looking at both platforms from a technical point of view that can be decisive in the choice for a certain platform. Especially variables as network speed, transaction throughput and the amount of peers a network supports. Such a technical comparison would be interesting to see in future research.

The third limitation of this thesis is the lack of implemented use cases in the real world. Since hardly any implemented use cases exist yet (due to the immaturity of the platforms), the choice for a fictional use case was a forced one. However, in future research, it would be interesting to see if the conclusions of this thesis last when the platforms are implemented and researched in real world organisations.

Lastly, a general limitation on the use of blockchains in an enterprise context is the immutability. While the immutability is an indispensable feature that secures the trustworthiness of information on the blockchain, it also causes trouble from a legal point of view. For example, the right to be forgotten would be troublesome for an organisation when it stores personal data on a blockchain. Therefore, a future research direction that is interesting are the implications of the law on the implementation of blockchain networks in businesses.

References

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., . . . & Muralidharan, S. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference* (p. 30). ACM.

Antonopoulos, A. (2014). *Mastering Bitcoin: unlocking digital cryptocurrencies.* O'Reilly Media, Inc.

Behlendorf, B. (2016, September 13). *Meet Hyperledger: An "Umbrella" for Open Source Blockchain & Smart Contract* . Retrieved from https://www.hyperledger.org/blog/2016/09/13/meet-hyperledger-an-umbrella-for-open-source-blockchain-smart-contract-technologies

Benhamouda, F., Halevi, S., & Halevi, T. (2018). Supporting private data on Hyperledger Fabric with secure multiparty computation. *IEEE International Conference on Cloud Engineering (IC2E)* (pp. 357-363). IEEE.

Blockgeeks. (2018, August 22). *What Is Hyperledger? The Most Comprehensive Video Ever! [Video file]*. Retrieved from https://www.youtube.com/watch?v=k4KKrQOV6SE

Brinkkemper, S. (1990). *Formalisation of Information Systems Modelling.* Dissertation, University of Nijmegen, Thesis Publishers, Amsterdam.

Chinosi, M., & Trombetta, A. (2012). BPMN: An introduction to the standard. *Computer Standards & Interfaces, 34(1)*, 124-134.

de Kruijff, J., & Weigand, H. (2017). Understanding the blockchain using enterprise ontology. *International Conference on Advanced Information Systems Engineering* (pp. 29-43). Springer, Cham.

Degnarain, N. (2017). *Can technology help us tackle illegal fishing?* Retrieved from https://www.weforum.org/agenda/2017/05/can-technology-help-tackle-illegal-fishing/

Friebe, T. (2017, November 3). *Trust your competitor? How you can do that with a Hyperledger Fabric blockchain.* Retrieved from https://medium.com/blockchainspace/trust-your-competitor-how-you-can-do-with-hyperledger-fabric-5939bacffe76

Guerreiro, S., Silva, D., & Sousa, P. (2018). Decentralized Enforcement of Business Process Control Using Blockchain. *Enterprise Engineering Working Conference* (pp. 69-87). Springer, Cham.

Gupta, M. (2017). *Blockchain for dummies.* IBM Limited Edition, US.

Handfield, R., & Bechtel, C. (2002). The role of trust and relationship structure in improving supply chain responsiveness. *Industrial marketing management, 31(4)*, 367-382.

Harmsen, A.F. (1997). *Situational Method Engineering.* Moret Ernst & Young.

Hoops, J. (2017). An introduction to Public and Private Distributed Ledgers. *Seminars FI/IITM: Network Architectures and Services*, (pp. 41-48).

Hyperledger. (2017). *Hyperledger Architecture, Volume 1: Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus.* Retrieved from https://www.hyperledger.org/resources/publications

Hyperledger. (2018). *About Hyperledger*. Retrieved from https://www.hyperledger.org/about

Hyperledger. (2018). *Hyperledger Fabric*. Retrieved from https://www.hyperledger.org/projects/fabric

Hyperledger Fabric. (2019). *Hyperledger Fabric Documentation.* Retrieved from https://hyperledger-fabric.readthedocs.io

Hyperledger Sawtooth. (2019). *Hyperledger Sawtooth Documentation.* Retrieved from https://sawtooth.hyperledger.org/docs/core/releases/1.1.5/

Hyperledger Sawtooth. (2019). *Hyperledger Sawtooth F.A.Q.* Retrieved from https://sawtooth.hyperledger.org/faq/glossary/

Iansiti, M., & Lakhani, K. (2017). The truth about blockchain. *Harvard Business Review, 95(1),* 118-127.

Koens, T., & Poll, E. (2018). The Drivers Behind Blockchan Adoption: The Rationality of Irrational Choices. *Workshop on Large Scale Distributed Virtual Environments (LSDVE).* EuroPar.

Lee, H., Padmanabhan, V., & Whang, S. (1997). Information distortion in a supply chain: The bullwhip effect. *Management science, 43(4),* 546-558.

Liao, S. (2017, 12 21). *Tea, juice, and vape companies add blockchain to their names to profit on bitcoin mania*. Retrieved from https://www.theverge.com/2017/12/21/16805598/companies-blockchain-tech-cryptocurrency-tea

McKinsey. (2018). *Blockchain beyond the hype: What is the strategic business value?* Retrieved from https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/blockchain-beyond-the-hype-what-is-the-strategic-business-value

Meijer, K., Nijssen, M., & Bulles, J. (2017). An Evaluation of a Design Science Research Artefact in the Field of Agile Enterprise Design. *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"* (pp. 212-219). Springer, Cham.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Pilkington, M. (2016). 11 Blockchain technology: principles and applications. In *Research handbook on digital transformations* (p. 225).

Proper, H., Bjeković, M., van Gils, B., & Hoppenbrouwers, S. (2017). Towards Grounded Enterprise Modelling. *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"* (pp. 141-151). Springer, Cham.

Stark, J. (2016). *Making sense of blockchain smart contracts*. Retrieved from http://www.coindesk.com/

Swan, M. (2015). *Blockchain: Blueprint for a new economy.* O'Reilly Media, Inc.

Swartz, L. (2017). Blockchain Dreams: Imagining Techno-economic Alternatives After Bitcoin. In *Another Economy Is Possible: Culture and Economy in a Time of Crisis* (pp. 82–105). Polity Cambridge.

Tapscott, D., & Tapscott, A. (2016). *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world.* Penguin.

Waughray, D. (2017). *Tuna 2020 Traceability Declaration: Stopping illegal tuna from coming to market.* Retrieved from https://www.weforum.org/agenda/2017/06/tuna-2020-traceability-declaration-stopping-illegal-tuna-from-coming-to-market/