Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

USING BLOCKCHAIN FOR MEDICAL DATA SHARING

Trabajo Final de Máster

Máster Universitario en Software y Sistemas

Author:

Diego Andres Yave Guzman

Tutor:

Sergio Paraíso Medina

Madrid, 2019

Contents

List of Figures

List of Acronyms

General acronyms

API: Application Programming Interface

DB: Database

EHR: Electronic Health Record

EMR: Electronic Medical Record

OMOP: Observational Medical Outcomes Partnership

PoW: Proof of Work

SDK: Software Development Kit

Acronyms referring to Hyperledger Fabric

A: Client Application

C: Channel

CA: Certificate Authorities

CC: Channel Configuration

Fabric: Hyperledger Fabric

L: Ledger

N: Network

NC: Network Configuration

O: Ordering Service

P: Peer node

R: Organization

S: Smart Contract

X: Consortium

Acronyms referring to MedRec

RC: Registrar Contract

PPR: Patient-Provider Relationship Contract

SC: Summary Contract

Acknowledgements

Special thanks to classmate Lucas and tutor.

Abstract

Today's organizations not only demand more and more data sharing techniques but also they ask for more security and scalability in a decentralized way. Medicine industry and healthcare are not the exception. Medical data is generally sensitive and needs to be protected against unauthorized access. Of course there are already many systems and protocols that aim to improve medical data sharing but more progress and innovation can still be done.

Blockchain technology continues causing impact on different fields. A blockchain network is a distributed public ledger where any transaction is witnessed and verified by network nodes. Throughout this work, blockchain is studied theoretically and practically so that it can be applied to the medical context.

In this work, it is presented a practical example of how blockchain can be introduced into the medical context following a particular blockchain framework. The case of study is about registering the medical data, especially the prescription of drugs, when a patient attends to medical examinations. It involves the data sharing of organizations so this is the specific case that this work intends to implement. For the implementation, Hyperledger Fabric, an enterprise-grade permissioned distributed ledger framework, was chosen.

Hyperledger Fabric was used to implement the case of study. The obtained results are the launch of a blockchain network with all the components, the implementation of client applications and the implementation of a smart contract. From Hyperledger Fabric and the case of study, more general conclusions were made about using blockchain for medical data sharing.

Resumen

En la actualidad, las organizaciones y empresas no solamente demandan más y más el intercambio de datos, sino que exigen más seguridad y escalabilidad en una forma descentralizada. La industria de la medicina y la atención sanitaria no son la excepción. Los datos médicos son generalmente sensibles y necesitan ser protegidos contra el acceso no autorizado. Por supuesto ya existen muchos sistemas y protocolos que apuntan a mejorar el intercambio de datos médicos, pero aún se puede hacer más progreso e innovación en ello.

La tecnología Blockchain continúa causando impacto en diferentes campos. Una red de blockchain es un libro mayor (ledger en inglés) público distribuido donde cualquier transacción es atestiguada y verificada por los nodos de la red. A lo largo de este trabajo, blockchain es estudiado teóricamente y prácticamente para que pueda aplicarse al contexto médico.

En este trabajo, se presenta un ejemplo práctico de cómo se puede introducir blockchain en el contexto médico siguiendo un framework de blockchain en particular. El caso de estudio es sobre el registro de datos médicos, especialmente la prescripción de medicamentos, cuando un paciente asiste a exámenes médicos. Implica el intercambio de datos de las organizaciones, por lo que este es el caso específico que este trabajo pretende implementar. Para la implementación, se eligió Hyperledger Fabric, un framework de ledger distribuido, permisivo, y con enfoque empresarial.

Se utilizó Hyperledger Fabric para implementar el caso de estudio. Los resultados obtenidos son el lanzamiento de una red blockchain con todos los componentes, la implementación de aplicaciones cliente, y la implementación de un smart contract. De Hyperledger Fabric y el caso de estudio, se hicieron conclusiones más generales sobre el uso de blockchain para el intercambio de datos médicos.

1 Introduction and Objectives

1.1. Introduction

Nowadays the amount of data that organizations manage is overwhelming because not only they have to manage their own data but also they have to share and consume it from other organizations and businesses. In the particular case of medical industry, it happens the same although sharing and consuming medical data should be managed more carefully because of the vulnerability of the delicate data.

Among other reasons, these vulnerabilities are why some hospitals and healthcare centers reject to share their data, and for those organization that decide to share their data, the troubles may come later. Of course there are already many systems, standards and protocols that aim to improve medical data sharing but record maintenance can be quite challenging [1], especially when there are many organizations involved. The interoperability changes between organizations represents additional barriers to get an effective data sharing. This lack of coordinated data management and exchange means that health records are fragmented instead of being cohesive [2].

As an example, through HIPPA (Health Insurance Portability and Accountability Act) privacy rule, providers can take up to sixty days to respond to a request for updating or removing a record in some situations [3]. So it's true that there are restrictions and difficulties for sharing data, and that's why medical data sharing craves innovation.

Although these difficulties, it is worth keeping with the progress since medical data sharing proves critical for research [4]. Also, sharing data is almost inevitable because hospitals don't work alone for the most part. For this reason, it's desirable to share medical data not only among a hospital staff but also among other hospitals and healthcare centers so that mutual collaboration is possible. Nonetheless, it's important to take into account issues related to security, privacy and data provenance as well.

Nowadays, Blockchain technology is still causing impact on different industries because it keeps evolving. It has attracted tremendous interest from different stakeholders such as finance, utilities, government agencies, and healthcare [5].

In this work, we explore a blockchain structure applied to a particular medical case of study. Firstly, blockchain fundamentals will be reviewed and Hyperledger Fabric, a blockchain framework, will be the chosen platform to implement the case of study.

Hyperledger Fabric is a blockchain framework ready for enterprise purposes that supports smart contracts, privacy and confidentiality among network participants. It will be used to propose a decentralized and collaborative network managed by medics and managers from different hospitals.

The case of study is about registering the medical data when a patient attends to a medical examination, especially the data related to prescription of drugs. The system should help doctors and hospital managers to track the information related to the drugs administration and prescription. In the case of patients that attend more than one hospital, it will help doctors to review the drug prescriptions from other hospitals. With this, the feature data provenance fits so that the data coming from different organizations can be verified.

1.2. Objectives

1.2.1. General Objective

As it was stated before and will be later too, blockchain keeps evolving over the time and innovating in diverse fields, including fields related to medicine and healthcare.

The general objective of this work is to introduce blockchain technology for medical data sharing.

1.2.2. Specific Objectives

1. Study blockchain not only from the point of view of cryptography but also from its application, which means as a framework for implementing decentralized computer resources.
2. Design a medical case of study to apply blockchain properly, as a way to review the implied considerations of applying blockchain in the medical context.
3. Implement the case of study with Hyperledger Fabric, which is the chosen blockchain framework, so that blockchain can be reviewed in practice.
4. Implement the feature data provenance with Hyperledger Fabric in the case of study, as an additional application of blockchain in the medical context.

1.3. Organization of This Work

The chapter one (this chapter) introduces the motivations, the medical context where blockchain is going to be applied, and the state-of-the-art concepts in a glance. Also, this chapter describes the objectives this work intends to achieve.

The chapter two makes an analysis about the state of the art in several aspects that are relevant to this work. Firstly, the fundamentals of blockchain will be described going from its first definition to how it has been evolving over the time so that it can be applied not only to financial services but also to other areas. Also, some important blockchain frameworks will be discussed. Secondly, clinical models will be studied to have a better understanding of the medical context this work aims to cover. Thirdly, prior work related to applying to blockchain to medical environment will be reviewed. Finally, Hyperledger Fabric, a blockchain platform, is going to be described in detail since it was the chosen one for the implementation. Hyperledger Fabric components and key concepts are described in this chapter.

The chapter three is about the design, architecture and implementation of the system. Firstly, the covered case of study is explained. This case is related to the medical examination of a patient and the involved drugs. Secondly, the architecture is presented going from an overview to a more detailed perspective. Finally, the development, configuration and execution aspects of the system are documented and explained detailing more about the decisions and the Hyperledger Fabric components.

Finally, the chapter four is about the results of the system and the investigation, which will be brought into discussion. Chapter five presents the obtained conclusions along with the further work.

2 State of the Art

2.1. Blockchain

2.1.1. Bitcoin and Blockchain

Blockchain was originally presented as a part of the cryptocurrency Bitcoin, a form of electronic cash. At [6] Nakatomo describes blockchain to address the challenge of ownership related to digital currencies and the security involved.

The blockchain is similar to a permanent book of records that keeps a log of all transactions that have taken place in chronological order. A blockchain is a growing list of records called blocks that are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data [7].

In the Figure 1 an example of blockchain related to transaction data is shown.

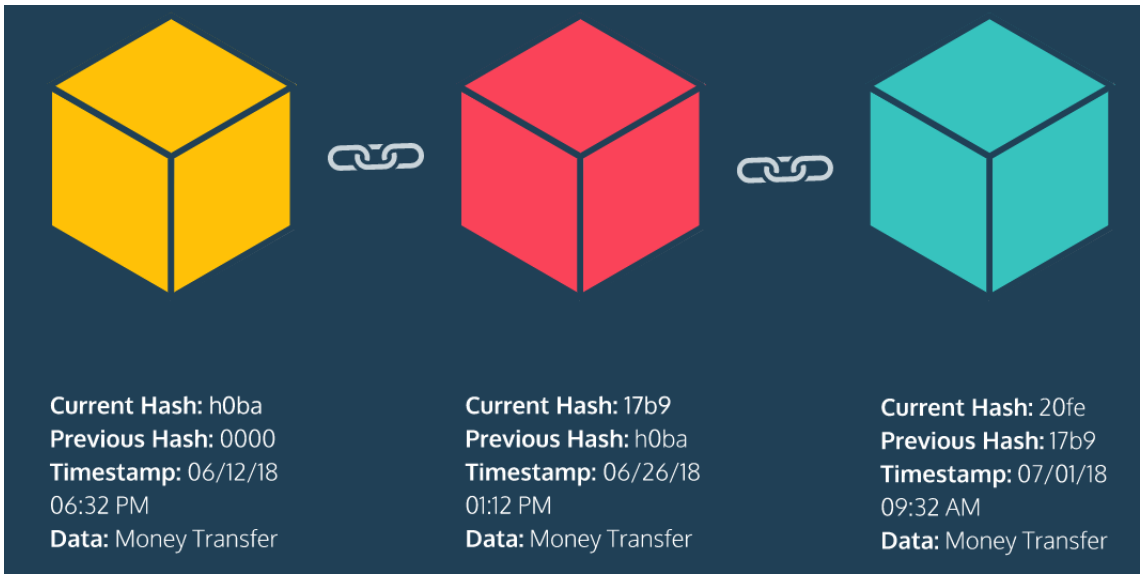

Figure 1 Blockchain structure and block properties

As can be seen, the yellow block is the first one and from there two more blocks were added. Blocks and their properties, which are the following, conform the blockchain.

- **Timestamp**: The time the block is created determines the location of it on the blockchain.
- **Transaction Data**: The information to be securely stored in the block.
- **Hash**: A unique code produced by combining all the contents within the block itself. Also known as a digital fingerprint.
- **Previous Hash**: This is how blocks are chained to one another. Each block has a reference to the block prior to its hash. All blocks but the first

one (called genesis) are pointing to the previous block. This is what makes the blockchain unique because this link will be broken if a block is tampered with.

Hashing, which is an application of cryptography, is fundamental to the design of the blockchain. It is a way to generate a seemingly random, but calculated string of letters and numbers from any input. This is accomplished by the use of a hash function. Blockchain uses a cryptographic hash function, meaning that the output is random but deterministic. This means the same input will always produce the same hash. That process is one-way, so the output (hash) cannot be used to produce the original input.

The way how hashing maintains the blockchain integrity is by verifying the matches. For example, if a participant decides to tamper with a block by modifying the transactions, the block's unique hash gets changed. However, the following block does not update to reflect this change since it still points to the previous block's hash. Thus, there is a mismatch between hashes and the link between blocks is broken. This results in an invalid copy of the blockchain. In this way, the records in the blockchain cannot be altered. In other words, the records are said to be immutable.

Because participants on the blockchain network are anonymous users on their computers, they can't trusted to verify transactions honestly. Proof-of-Work does nothing more than introduce an additional security constraint to verify transactions. This constraint takes the form of a computationally difficult math problem, which means to say that it takes a lot of time even for the computer to solve the problem [8].

Instead of randomly being chosen to broadcast their unconfirmed block, a special group of participants, also known as miners, now need to solve a problem in order to be eligible to broadcast their block. The problem or puzzle, also known as Proof-of-Work, takes the form of a guessing game that involves the use of hashing.

If a dishonest participant decides to tamper with a block, they would have to solve the Proof-of-Work for each subsequent block in order to introduce the tampered block into the network. This is computationally infeasible and almost impossible!

2.1.2. Evolution of Blockchain

It may seem that blockchain is a new technology but as a matter of fact, it has been ten years since Bitcoin paper was published for the first time. At that time, financial services were taking advantage blockchain the most. However, nowadays other

industries are using blockchain as well, and it's constantly evolving, which is why blockchain cannot go unnoticed today.

Besides cryptocurrencies and financial services, blockchain has been used to implement and improve smart contracts, data provenance, supply chain and healthcare. Since this last one will be the object of our study, we can mention more specific examples. Public healthcare management, user-oriented medical research, medical data provenance, and drug counterfeiting are some specific examples from healthcare where blockchain was already introduced [5].

Blockchain has proven to be multipurpose so there are many implementations and infrastructures. In the case of Bitcoin, it is aimed to be open and decentralized so that everybody can be part of the bitcoin network. However, this behavior is not desired on business contexts. Enterprises have sensible data which is supposed to stay among employees, and as a matter of fact there is the kind of information that is only accessible to managers or heads of departments only.

Related to who can be participate in the blockchain, a blockchain can be permissionless or permissioned. In a permissionless blockchain anybody can potentially participate in the network, but it has to demonstrate that it's allowed to participate in the blockchain by solving computationally intensive puzzles. These puzzles are hashing exercises to whomever want to participate in the network and at the end, only the requester that is allowed should win to complete the puzzles. On the other hand, under a permissioned blockchain, there is control over who participates in the network, which is determined by a set organizations that acts like a consortium. Swanson describes the divergence between permissionless cryptocurrency systems and permissioned distributed ledger systems [9].

The permissioned blockchain and other privacy configurations are not included in a blockchain as a default feature. For this reason, the application of the original blockchain as it is can be impractical and is not ready to be used for enterprise purposes.

2.1.3. Platforms and Infrastructures

Many platforms and infrastructures have emerged since the first blockchain paper was published. Follow-up systems such as Namecoin, Litecoin and Peercoin adapted this original currency application of blockchain into other applications though rather simplistic ones.

From there, many adaptations were done related to data storage, health records, voting systems and smart contracts. The adaptations beyond Bitcoin are Ethereum, Bitshares, Counterparty, Truthcoin and Factom.

In the case of the adaptations that goes even beyond Bitcoin and blockchain, these are booming today. Ripple, Hyperledger, Ethereum, R3 Corda, Quorum and Eris are just some names.

Among the most important, there are Ethereum and Hyperledger project. They both are highly flexible, but in different aspects. Ethereum's powerful smart contracts engine makes it a generic platform for literally any kind of application. However, Ethereum's permissionless mode of operation and its total transparency comes at the cost of performance scalability and privacy. Hyperledger, and more specifically Hyperledger Fabric, solves performance scalability and privacy issues by permissioned mode of operation and fine-grained access control. Further, the modular architecture allows Fabric to be customized to a multitude of applications [10].

Ethereum was not tested and evaluated but from its review, the conclusion is that Ethereum is complete but too generic. Gavin Wood, cofounder of Ethereum, describes it in a very formal paper. It is presented as an application on a decentralized, but singleton, compute resource, implemented in generalized manner. Ethereum is an open source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality. It supports a modified version of Nakamoto consensus via transaction-based state transitions. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework [11].

The focus of this work is not the analysis and evaluation of blockchain frameworks but reviewing Ethereum was important to have the big picture. Now it's time to review the project that we are interested the most.

Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation and contributed by the most important companies such as IBM; Cisco, SAP, Intel, Digital Asset, among many others including leaders in finance, banking, IoT, supply chain, manufacturing and technology [12].

Figure 2 shows the Hyperledger Greenhouse with all frameworks and tools. This figure can give us the idea how big Hyperledger project really is. Also, it leaves us between seeing that there are many applications of blockchain according to the type of industry.

Figure 2 Hyperledger Greenhouse showing the business blockchain frameworks and tools

In this work, we will focus in one tool and one framework: Hyperledger Composer and Hyperledger Fabric.

Hyperledger Composer is a toolset to make developing blockchain applications easier. Composer is helpful to rapidly develop use cases and deploy a blockchain solution. It allows to model a business network and integrate existing systems and data with blockchain applications.

Hyperledger Composer tool is built on the Hyperledger Fabric framework basis. At the time of beginning with this investigation, Hyperledger Composer was installed and tested but it was left behind since it is not really useful for the purposes of this work. Hyperledger Composer uses a different terminology than Hyperledger Fabric and seems to have a more administrative business approach. Thus, investing time to Hyperledger Composer doesn't mean to be useful to understand Hyperledger Fabric.

Besides that, one of the main reasons to stop reviewing Hyperledger Composer was the fact that Hyperledger Composer project started to be discontinued. Simon Stone, one of the heads behind Hyperledger Composer, communicated that even though a lot of efforts has been invested in the project, they will mostly stop bringing new features to it. In synthesis the main reasons are regarding its growing architecture and the difficulties

to maintain it. According to the note, they will focus more on bringing more features straight to Hyperledger Fabric.

On the other hand, in the case of Hyperledger Fabric framework (just Fabric for short), it will be reviewed in detail in the later sections. Fabric will be the cornerstone of this current work.

2.2. Clinical Models

The main goal of this work is more about applying blockchain to the medical context than clinical models. So, they won't be studied exhaustively. Either way, it's important to review them so that the next section Prior Work can be taken advantage of.

There are several clinical models, and some of them are more than just models because they can implement a data exchange protocol or provide an API. Among them, it can be mentioned IHE, OpenEHR, i2b2, OMOP, HL7 RIM, and FHIR, which will described below [13].

Although the case of study of this work is aimed to be model-independent, for simplicity OMOP data model was chosen because it is a simple and widely used model.

2.2.1. IHE

IHE stands for Integrating the Healthcare Enterprise. It is an initiative by healthcare professionals and industry to improve the way computer systems in healthcare share information. IHE promotes the coordinated use of established standards to address specific clinical needs in support of optimal patient care.

2.2.2. OpenEHR

OpenEHR is a technology for e-health, an open standard specification that describes the management and storage, retrieval and exchange of health data in electronic health records (EHRs). The various artefacts of OpenEHR are produced by the OpenEHR community and managed by the OpenEHR Foundation, an international non-profit organization. In OpenEHR, all health data for a person is aimed to be stored in a one lifetime, vendor-independent, person-centered EHR.

2.2.3. i2b2

Informatics for Integrating Biology and the Bedside (i2b2) is an NIH-funded National Center for Biomedical Computing (NCBC). The i2b2 Center is developing a scalable computational framework to address the bottleneck limiting the translation of genomic findings and hypotheses in model systems relevant to human health. New computational paradigms and methodologies are being developed and tested in several diseases.

I2b2 defines a system to store clinical data from different sources in a homogeneous and consistent way. It follows an architecture that is divided in components which form together the i2b2 hive. Each hive's cell is a functional part of the system but it works separately from the other cells.

2.2.4. HL7 RIM

The HL7 RIM, which stands for Health Level 7, and Reference Information Model, is a critical component of the HL7 V3 family of standards. It is the root of all information models and structures developed as part of the V3 development process.

The Reference Information Model (RIM) is the combined consensus view of information from the perspective of the HL7 working group and the HL7 affiliates. The RIM is the ultimate source from which all HL7 Version 3 standards draw their information-related content.

The RIM, along with Data Types and Vocabularies are the foundation for all information modeling within HL7. The constrained models derived from these serve as documents, data for services, and messages. As such, they are a part of every HL7 Version 3 standard and have the same customers as do the standards defined from them.

2.2.5. FHIR

Fast Healthcare Interoperability Resources (FHIR) is a next generation standards framework created by HL7. FHIR combines the best features of HL7 v2, HL7 v3 while leveraging the latest web standards and applying a tight focus on implementability.

FHIR solutions are built from a set of modular components called resources. These resources can easily be assembled into working systems that solve real world clinical and administrative problems at a fraction of the price of existing alternatives.

FHIR is suitable for use in a wide variety of contexts including mobile phone apps, cloud communications, EHR-based data sharing, and server communication in large institutional healthcare providers.

2.2.6. OMOP

As it was stated before, in this work OMOP was chosen for being simple and widely used among the others models. Thus, OMOP will be reviewed in more detail.

OMOP stands for Observational Medical Outcomes Partnership. The concept behind OMOP approach is to transform data contained within those databases into a common format (data model) as well as a common representation (terminologies, vocabularies, coding schemes), and then perform systematic analyses using a library of standard analytic routines that have been written based on the common format.

In the common data models that OMOP defines, there is The Standardized Clinical Data Tables. These tables contain the core information about the clinical events that occurred longitudinally during valid observation periods for each person, as well as demographic information for the person. Below Figure 3 provides an entity-relationship diagram highlighting the tables within the Standardized Clinical Data portion of the OMOP Common Data Model.

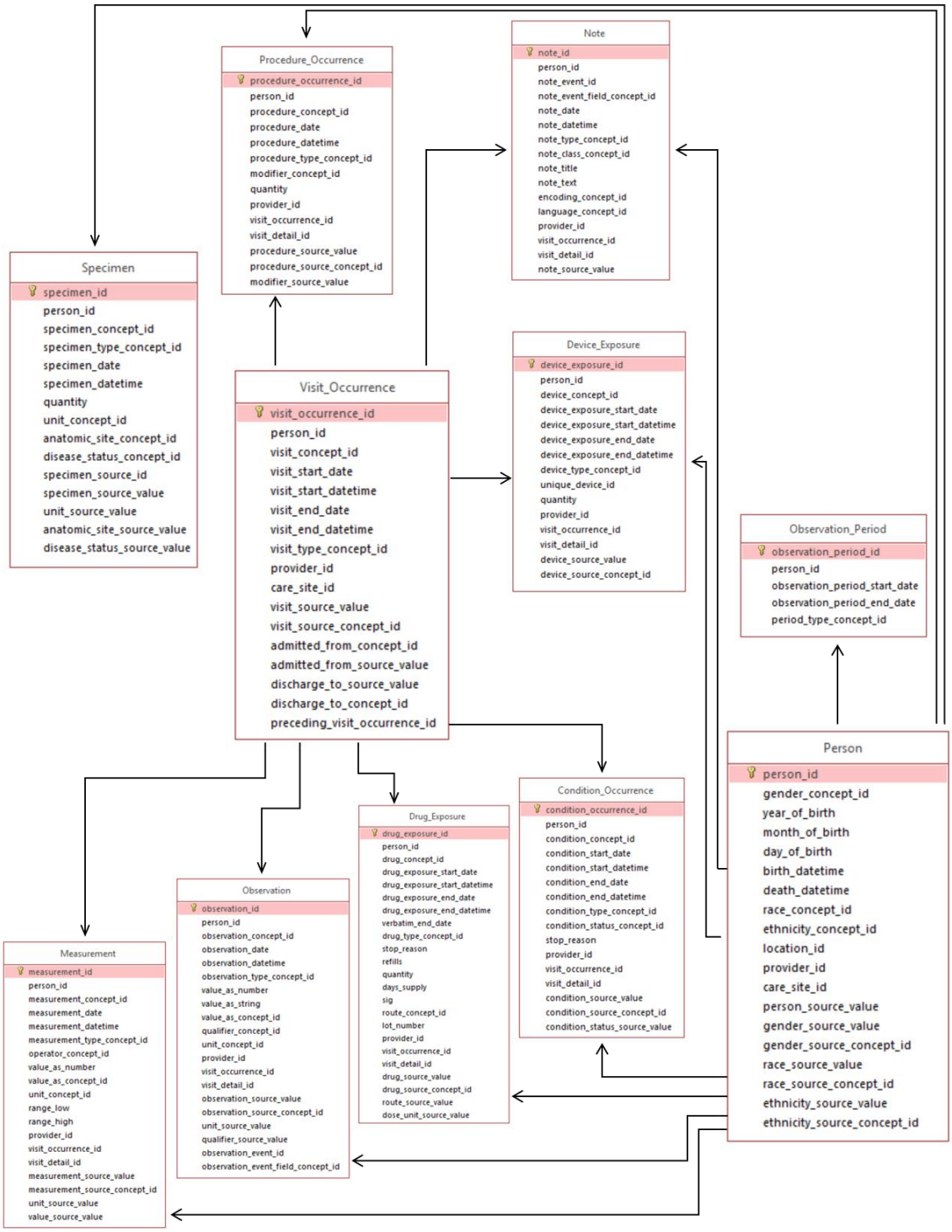

Figure 3 Diagram of OMOP's standardized clinical data tables

Notice that in the previous entity-relationship diagram it's not being shown all the tables. The complete Standardized Clinical Data Tables consists of six more tables but the showed ones are the most important.

As a matter of fact, for the purposes of this work only three tables are used, which are person, observation_period, and drug_exposure. Also, these tables store almost every detail related to its purpose, so it won't be necessary to use all their columns.

The person table contains records that uniquely identify each patient in the source data who is time at-risk to have clinical observations recorded within the source systems.

The observation_period table contains records which uniquely define the spans of time for which a Person is at-risk to have clinical events recorded within the source systems.

The drug_exposure table captures records about the utilization of a Drug when ingested or otherwise introduced into the body. A drug is a biochemical substance formulated in such a way that when administered to a person it will exert a certain physiological effect. Drugs include prescription and over-the-counter medicines, vaccines, and large-molecule biologic therapies.

These OMOP tables will be useful to design the case of study for this work.

2.3. Prior Work

Before presenting the actual proposal in the chapter 3, first it's necessary to review the work that other authors have already done about applying blockchain beyond financial services and Bitcoin. After that, we will focus on the application of blockchain to the medical context.

At [14] Zyskind et al. described a decentralized personal data management system that ensures users own and control their data. Unlike Bitcoin, the transactions in this system are not strictly financial.

Entering more to the medical context, at [15] Kish talks about why patients should own their medical data in a very interesting and not-so-technical paper. Kish talks about creating a health data resource in a much broader and more universal context, controlled by the individuals who supplied the data. Kish supports that this is a unique moment where we may be able to provide for personal control and, at the same time, create a global knowledge medical resource. Kish have coined the term "UnPatient" for their new model of data ownership as it has the double entendre of the patient subjected to medical paternalism and information asymmetries, along with the idea that it has taken far too long to become free to use our medical data as we see fit and to own it.

This way, we will be reviewing more and more specific cases where blockchain takes part in the medical context.

2.3.1. Blockchain encryption for electronic health records

The applications of blockchain in healthcare industry are several but we are interested especially on those cases related to medical records sharing.

As it was reviewed before, there are different clinical data models and protocols to exchange data. For example, an Electronic Health Records (EHR) is a systematized collection of patient and population electronically-stored health information in a digital format. Such data are generally sensitive so the biggest challenges in healthcare systems is to share medical data securely [16].

Related to EHR, OpenEHR was reviewed before but that was just one example of a standard. Healthcare centers may fully support a data model or exchange protocol. Also, there is the case where they support it partially or even the case when they don't support it at all.

Regardless the standards, often the data in EHRs remain unchanged once they are uploaded to system. In this context is where blockchain can be potentially used to facilitate the data sharing among different participating medical organizations and individuals.

At [17] Chen et al. proposed a blockchain based searchable encryption scheme for EHRs sharing. Chen's proposal is designed to facilitate different healthcare institutions to share medical records in a secure manner. It's aimed to bring convenience to both patients and researches.

The index for EHRs is constructed through complex logic expressions and stored in the blockchain so that a data user can utilize the expressions to search the index. In Chen's proposal, only the search index is added to the blockchain to facilitate the distribution of EHRs, while the real EHRs are stored in a public cloud server in an encrypted form. When users want to access these EHRs, they need to authenticate themselves to the data owner to obtain the authorization together with the decryption key. With this arrangement, the data owner has full control over who can see their data.

The smart contract used in the proposed scheme is designed to trace monetary rewards, including transactions fees, among involved parties in the multiuser setting. It ensures that the data owner is paid as long as (s)he reveals the transcript, which allows other users to search the database. At the same time, other users can obtain accurate search results as long as they make the required payments. This property guarantees fairness among the data owners and users.

Although Chen's proposal is complete, it's focused on the process of file encryption, index construction, transaction generation and search. Clearly we are not interested in things like algorithms to construct the index or any low level designs.

It would have been desirable to have Chen's proposed scheme under Hyperledger Fabric for our convenience but it is under Ethereum. Regardless that fact, Chen's paper was useful to understand the architecture to introduce blockchain to an already existing system.

2.3.2. MedRec: Data Access and Permission Management

MedRec is a concrete case of a blockchain system applied to handle electronic medical records (EMRs). It manages authentication, confidentiality, accountability and data sharing. This project incentivizes medical stakeholders to participate in the network as blockchain miners to sustain and secure the network via Proof of Work [18].

Via smart contracts on an Ethereum blockchain, MedRec system log patient-provider relationships that associate a medical record with viewing permissions and data retrieval instructions for execution on external databases.

To navigate the potentially large amount of record representations, MedRec structures them on the blockchain by implementing three types of contracts. Figure 4 illustrates the contract structures and relationships.

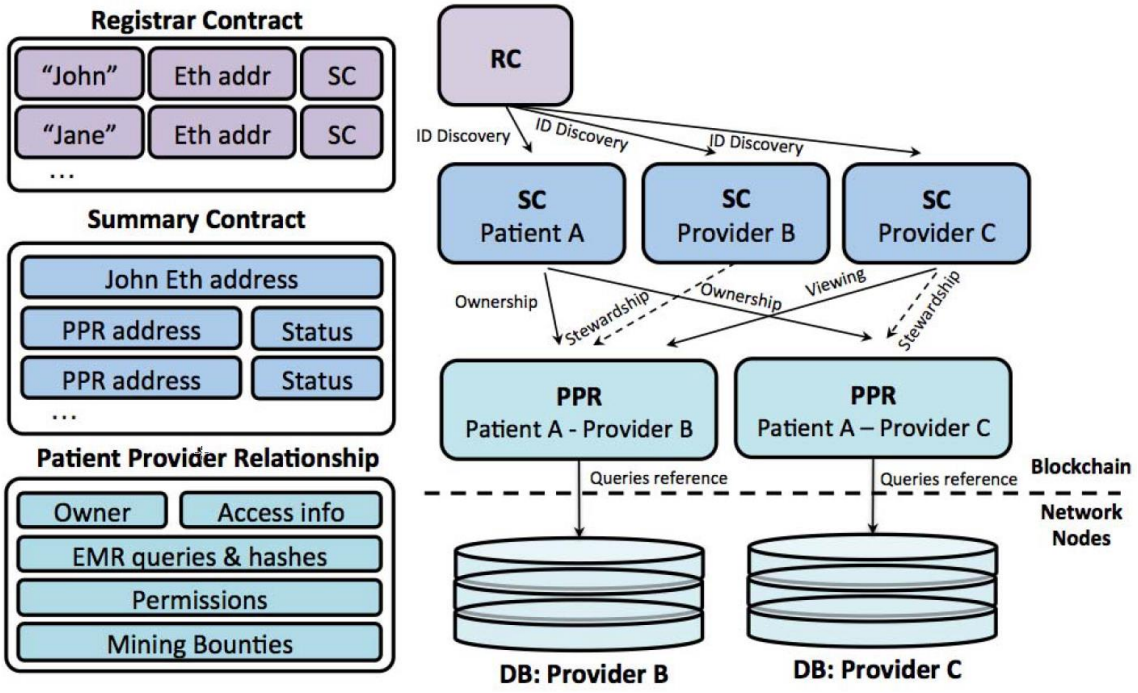

Figure 4 MedRec smart contracts on the blockchain, and data references [18]

In the previous figure, on the left there are 3 boxes which represents the 3 smart contracts and the data they contain. On the right from top to bottom, it's shown the smart contracts interactions. Notice that Eth stands for Ethereum and the acronyms RC, SC and PPR stands for the smart contracts described below. It's important to understand this figure before keep going and pay attention to the MedRec's acronyms.

- **Registrar Contract (RC)** is the global contract to map participant identification strings to their Ethereum address identity.
- **Patient-Provider Relationship Contract (PPR)** is issued between two nodes in the system when one node stores and manages medical records for the other.
- **Summary Contract (SC)** works as a trail for participants in the system to locate their medical record history.

Figure 5 illustrates MedRec's backend implementation of a scenario where a provider adds a record for a new patient. The flow of the system is given by the numbers from 1 to 9.

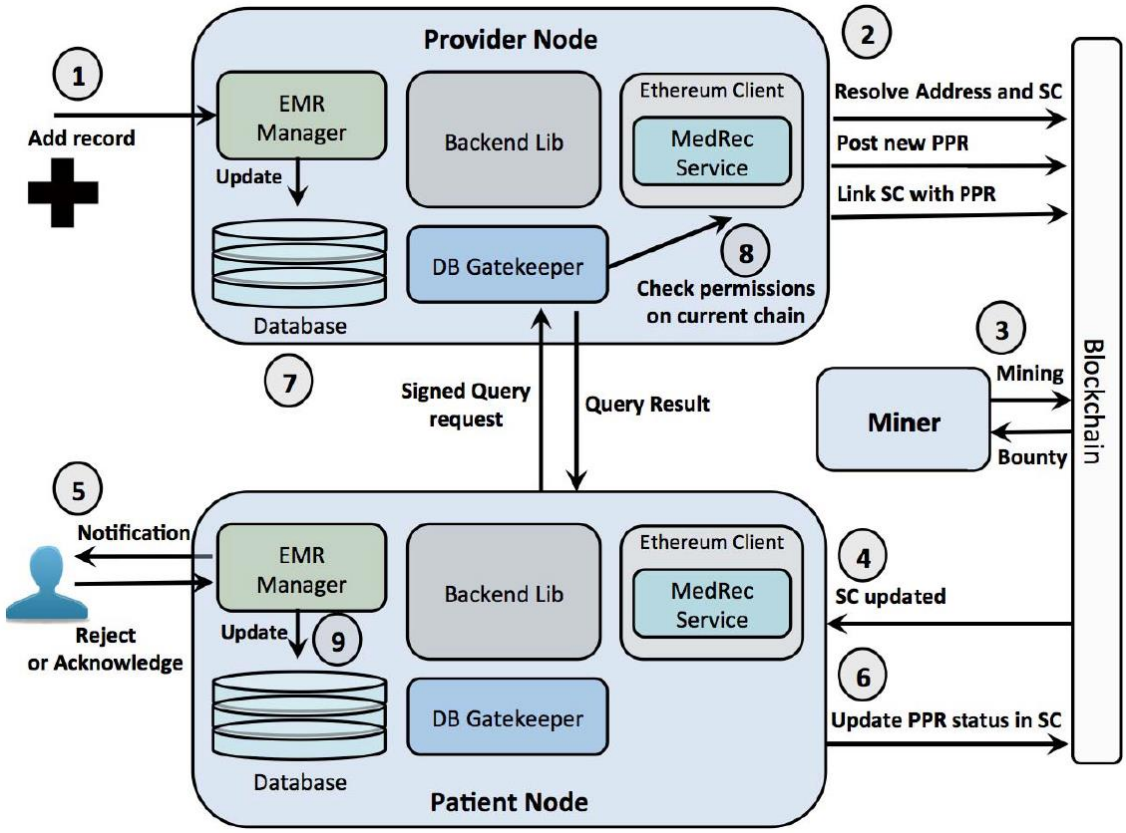

Figure 5 MedRec system orchestration when a provider adds a patient record

MedRec design introduces four software components.

- **Backend Library**: abstracts communications with the blockchain and exports a function-call API, which interacts with Ethereum.
- **Ethereum Client**: handles a broad set of tasks such as connecting to the peer-to-peer network, encoding, and sending transactions and keeping a verified local copy of the blockchain.
- **Database Gatekeeper**: implements off-chain, access interface to the node's local database.
- **EMR Manager**: is a user interface application to manage EMRs.

Steps 1 to 3 from Figure 5 illustrate MedRec's backend implementation of a scenario where a provider adds a record for a new patient. Using the RC on the blockchain, the patient's identifying information is first resolved to their matching Ethereum address and the corresponding SC is located. Next, the provider uploads a new PPR to the blockchain, indicating their stewardship of the data owned by the patient's Ethereum address. The provider node then crafts a query to reference this data

and updates the PPR accordingly. Finally, the node sends a transaction which links the new PPR to the patient's SC, allowing the patient node to later locate it on the blockchain.

Steps 4 to 6 from Figure 5 continue from the patient node perspective the use case described above. The patient's modified Ethereum client continuously monitors his SC. Once a new block is mined with the newly linked PPR, the client issues a signal which results in a user notification. The user can then acknowledge or decline her communication with the provider, updating the SC accordingly. If the communication is accepted, our prototype implementation automatically issues a query request to obtain the new medical data. It uses the information in the new PPR to locate the provider on the network and connect to its Database Gatekeeper server.

Steps 7 to 9 in Figure 2 illustrate how a patient retrieves personal data from the provider node. Note that components similarly support third-parties retrieving patient shared data: the patient selects data to share and updates the corresponding PPR with the third-party address and query string. If necessary, the patient's node can resolve the third party address using the RC on the blockchain. Then, the patient node links their existing PPR with the care provider to the third-party's SC. The third party is automatically notified of new permissions, and can follow the link to discover all information needed for retrieval. The provider's Database Gatekeeper will permit access to such a request, corroborating that it was issued by the patient on the PPR they share.

As it could be seen, MedRec integrates provider's existing data storage infrastructure. It facilitates continued use of existing systems. MedRec was designed with flexibility to support open standards for health data exchange such as FHIR and other flavors of HL7.

To recapitulate, MedRec shows how principles of decentralization might be applied to largescale data management in an EMR system. It enables patient data sharing and incentives for medical researchers (miners) to sustain the system. MedRec incentivizes miners to participate in the network and contribute their computational resources to achieve a trustworthy, gradual advancement of the chain. This provides them with access to aggregate, anonymized data as mining rewards, in return for sustaining and securing the network via Proof of Work.

With respect to the interests of this current work, MedRec is what we want to implement though something smaller and simpler. However, we cannot take MedRec as something more than a theoretical example because it is a public (permissionless) blockchain, and that's why Ethereum needs Proof of Work so that nodes can be validated

as trustworthy. In MedRec's scenario everybody is potentially able to participate in the network.

However, this is not the scenario this work intends to cover. We want to cover the scenario where an organization knows with whom other organizations they are working. This is a more common situation for enterprises, and this is why Hyperledger Fabric will be used instead of Ethereum.

2.3.3. ProvChain: Data Provenance

ProvChain is another case of successful application of blockchain. Although it is not so related to the medical context, it is of special interest to us for being about data provenance.

Before reviewing ProvChain, let's review what data provenance is. Data provenance, one kind of metadata, pertains to the derivation history of a data product starting from its original sources. It provides the history of the origins of all changes to a data object, list of components that have either forwarded or processed the object and users who have viewed and/or modified the object and has enhanced requirements for assurance [19].

Under blockchain, the data provenance is guaranteed since the data is stored on the blockchain blocks and they are immutable and cannot be tampered.

ProvChain is a decentralized and trusted cloud data provenance architecture that uses blockchain. It operates mainly in three phases: provenance data collection, provenance data storage, and provenance data validations [20]. Figure 6 illustrates its system interaction.

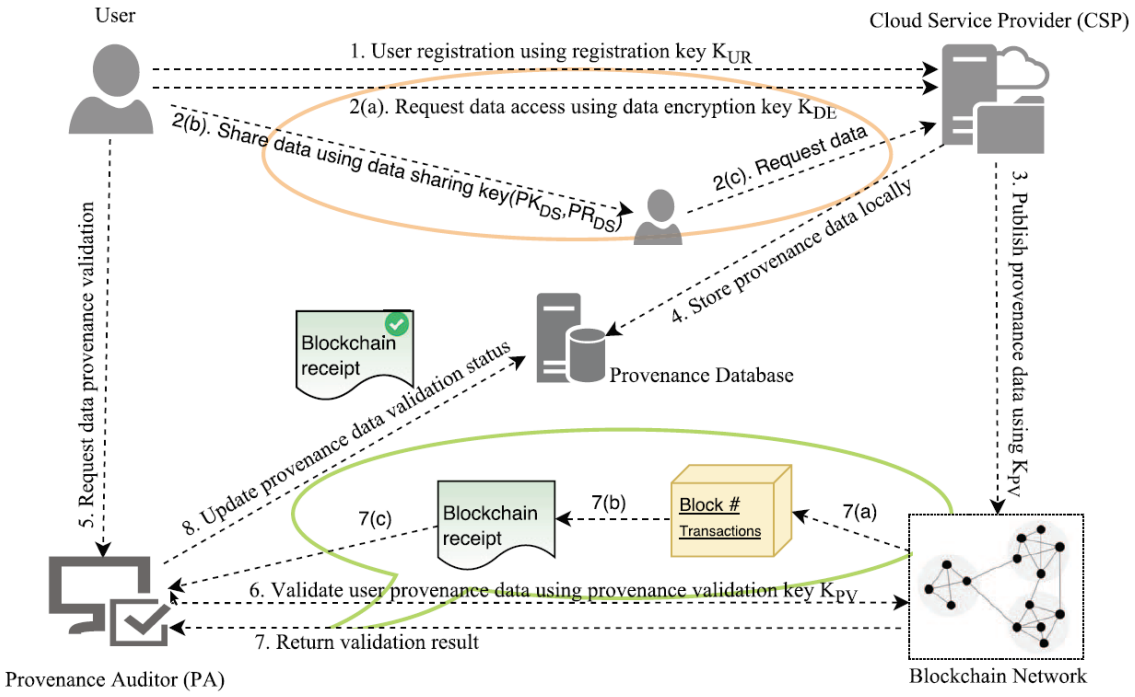

Figure 6 ProvChain system interaction [20]

The previous figure shows components flow at ProvChain starting with the user registration. These are the main components:

- **Cloud User**. A user, who owns its data and has sharing relationship on other users' data. The provenance data will not expose real user identity.
- **Cloud Service Provider (CSP)**. It offers a cloud storage service, is responsible for user registration and can audit the data changes all the time.
- **Provenance Database**. The provenance database records all provenance data on the blockchain network. All data records are anonymized.
- **Provenance Auditor (PA)**. PA can retrieve all the provenance data from the blockchain into the provenance database and validate the blockchain receipt.
- **Blockchain Network**. The blockchain network consists of globally participating nodes. All the provenance data will be recorded in form of blocks and verified by blockchain nodes.

Also in Figure 6, notice it's been used some keys. To use ProvChain, users are required to register the service and create their credentials. For cloud storage applications, users generate data encryption key pairs to encrypt their cloud data for

confidentiality. If the user wants to share a file, a data sharing key will be provided. For provenance data, cloud service provider generates key pairs to encrypt provenance data for privacy considerations, because provenance data will be further uploaded and published to the blockchain network.

ProvChain is built on top of open source applications. For the data collection and storage, ProvChain is built on top of an open source application called ownCloud which provides both web-based cloud storage services similar to Dropbox and Google Drive. For provenance data storage, ProvChain uses Tierion API to publish data records to blockchain network. Also, Tierion API is used for provenance data validation. For publishing data records to blockchain network, ProvChain adopts Chainpoint standard.

To recapitulate, ProvChain operates in 3 phases: provenance data collection, provenance data storage and provenance data validation. For each phase, it uses a different open source software component.

With respect to the interests of this current work, ProvChain is a good case of data provenance architecture. It will be taken as an example to implement data provenance, although making some changes.

For example, we won't use a Cloud Service Provider to storage or register users. Also, unlike ProvChain that uses different software components, we will use Hyperledger Fabric for everything instead.

2.4. Hyperledger Fabric

Hyperledger is the chosen blockchain framework for this work so in this section it will be studied in more detail covering key concepts and components.

In general terms, a blockchain is an immutable transaction ledger, maintained within a distributed network of peer nodes. These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that bind each block to the preceding block. Hyperledger Fabric is an implementation of distributed ledger technology that delivers enterprise-ready network security, scalability, confidentiality and performance, in a modular blockchain architecture [21].

In a permissionless blockchain such as the one underlying the Bitcoin cryptocurrency, anyone can operate a node and participate through spending CPU

cycles and demonstrating a proof of work. However, for our case of study, which will be reviewed in the next chapter, we don't need that. It makes sense for Bitcoin cryptocurrency to be so public, but for enterprises and for our case, it doesn't make sense since sensible data is managed.

On the other hand, a permissioned blockchain controls who participates in validation and in the protocol. Nodes typically establishes identities and form a consortium. Hyperledger Fabric is permissioned as well [22], so this is another reason why we are interested on it.

The following sections describe how Hyperledger Fabric (Fabric) differentiates itself from other blockchain platforms and describes some of the motivation for its architectural decisions.

2.4.1. Blockchain Network

A blockchain network is a technical infrastructure that provides ledger and smart contract services to applications. Primarily, smart contracts are used to generate transactions which are subsequently distributed to every peer node in the network where they are immutably recorded on their copy of the ledger. The users might be end users using client applications or blockchain network administrators.

Figure 7 shows most important Fabric's components by presenting an example of a blockchain network that interact with some external components. The involved components are:

- Client Application (A)
- Channel (C)
- Certificate Authorities (CA)
- Channel Configuration (CC)
- Ledger (L)
- Network (N)
- Network Configuration (NC)
- Ordering Service (O)
- Peer node (P)
- Organization (R)
- Smart Contract (S)
- Consortium (X)

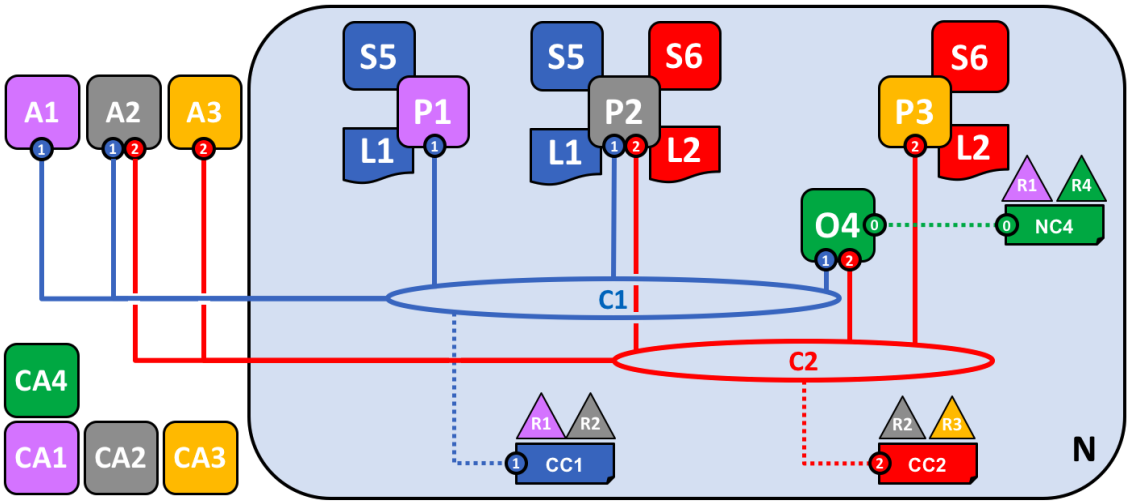

Figure 7 Sample of Fabric's blockchain network

All of these Fabric components and others will be described in the next sections following the time order that they were included at the network.

2.4.2. Organization (R)

Also known as members, organizations are invited to join the blockchain network by a blockchain service provider. At Figure 7, there are four organizations which are R1, R2, R3 and R4.

In green, R4 has been assigned to be the network initiator. It has been given the power to set up the initial version of the network and has no intention to perform business transactions on the network. R1 and R2 have a need for a private communications within the overall network, as do R2 and R3.

An organization can be as large as a multi-national corporation or as small as an individual. The transaction endpoint of an organization is a peer. A collection of organizations form a consortium. While all of the organizations on a network are members, not every organization will be part of a consortium.

2.4.3. Certificate Authority (CA)

A certificate authority is used to issue certificates to administrators and network nodes. At Figure 7 there are four certificates authorities which are CA1, CA2, CA3 and CA4. CA4 is used to dispense identities to the administrators and network nodes of the R4 organization.

CA4 plays a key role in our network because it dispenses X.509 certificates that can be used to identify components as belonging to organization R4. Certificates issued by CAs can also be used to sign transactions to indicate that an organization endorses the transaction result, a precondition of it being accepted onto the ledger.

2.4.4. Network Configuration (NC)

The network is governed according to policy rules specified in network configuration. At Figure 7, also in green, there is only one network configuration NC4 that interacts with ordering service, which will be explained above.

2.4.5. Ordering Service (O)

At the time of building the network from scratch, a network is formed when an orderer is started. At Figure 7, the ordering service comprising a single node O4, is configured according to a network configuration NC4, which gives administrative rights to organization R4. At the network level, Certificate Authority CA4 is used to dispense identities to the administrators and network nodes of the R4 organization.

Many distributed blockchains, such as Ethereum and Bitcoin, are not permissioned, which means that any node can participate in the consensus process. These systems rely on probabilistic consensus algorithms which eventually guarantee ledger consistency to a high degree of probability. Fabric works differently. It features a set of nodes called an ordering nodes that does this transaction ordering. Orderers also maintain the list of organizations that are allowed to create channels, the consortium.

Ordering service nodes receive transactions from many different application clients concurrently. These nodes work together to collectively form the ordering service. Its job is to arrange batches of submitted transactions into a well-defined sequence and package them into blocks. These blocks will become the blocks of the blockchain.

It’s possible to have only one ordering node under the ordering service, which is the simplest case and is mostly used for development environments. Thus, one ordering node is enough for the purposes of this work.

Just like peers, ordering nodes belong to an organization. And similar to peers, which will be reviewed later, a separate CA should be used for each organization. Peers form the basis for a blockchain network, hosting ledgers, which can be queried and updated by applications through smart contracts. Specifically, applications that want to

update the ledger are involved in a process with three phases that ensures all of the peers in a blockchain network keep their ledgers consistent with each other.

In the first phase, a client application sends a transaction proposal to a subset of peers that will invoke a smart contract to produce a proposed ledger update and then endorse the results. The endorsing peers do not apply the proposed update to their copy of the ledger at this time. Instead, the endorsing peers return a proposal response to the client application. The endorsed transaction proposals will ultimately be ordered into blocks in phase two, and then distributed to all peers for final validation and commit in phase three.

2.4.6. Consortium (X)

At Figure 7, a network administrator defines a consortium X1 that contains two members, the organizations R1 and R2. This consortium definition is stored in the network configuration NC4, and will be used at the next stage of network development. CA1 and CA2 are the respective Certificate Authorities for these organizations.

Another present consortium is X2 that is formed by R2 and R3. These two consortiums X1 and X2 are not shown in the figure for simplicity, but it can be implied from the way organizations R1 and R2, and R2 and R3, are grouped.

2.4.7. Channel (C)

A channel is a private blockchain overlay that allows data isolation and confidentiality. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be properly authenticated to a channel in order to interact with it.

At Figure 7, there are two channels C1 and C2. C1 has been created for R1 and R2 using the consortium definition X1. The channel is governed by a channel configuration CC1, completely separate to the network configuration. CC1 is managed by R1 and R2, who have equal rights over C1. R4 has no rights in CC1 whatsoever.

Similarly, C2 was created for R2 and R3 using consortium definition X2. The channel has a channel configuration CC2, completely separate to the network configuration NC4, and the channel configuration CC1. Channel C2 is managed by R2 and R3 who have equal rights over C2 as defined by a policy in CC2. R1 and R4 have no rights defined in CC2 whatsoever.

2.4.8. Peer (P)

Peer nodes are the network components where copies of the blockchain ledger are hosted. They are maintained by organizations.

At Figure 7, there are three peers which are P1, P2 and P3. Each of them was created by its own organization. P1 has joined the channel C1. P1 physically hosts a copy of the ledger L1. P1 and O4 can communicate with each other using channel C1. Similarly happens with P2 and P3.

2.4.9. Ledger (L)

A ledger consists of two distinct, though related, parts: world state and blockchain.

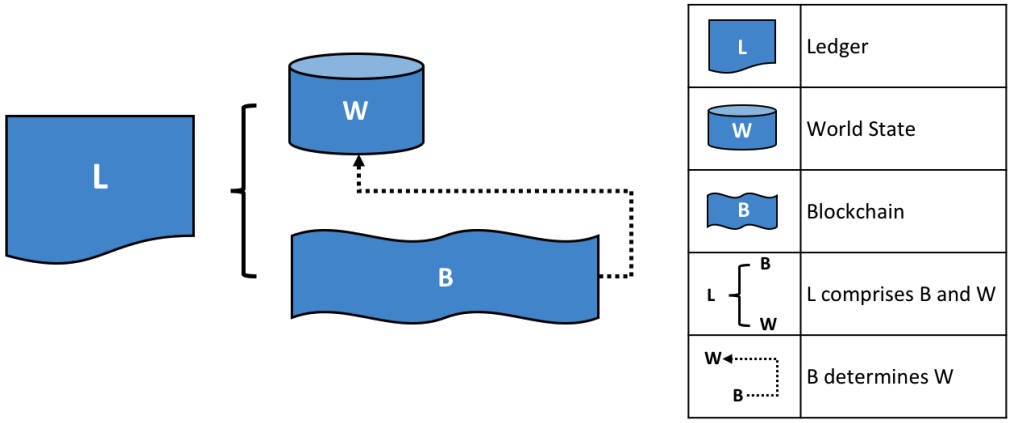

Figure 8 Fabric's ledger structure

The world state part is a database that holds a cache of the current values of a set of ledger states. The world state makes it easy for a program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log. The world state can change frequently, as states can be created, updated and deleted.

The blockchain part is a transaction log that records all the changes that have resulted in the current the world state. Transactions are collected inside blocks that are appended to the blockchain, enabling to understand the history of changes that have resulted in the current world state. The blockchain data structure is very different to the world state because once written, it cannot be modified.

Blockchains are immutable which means once a block has been added to the chain, it cannot be changed. In contrast, the world state is a database containing the current value of the set of key-value pairs that have been added, modified or deleted by the set of validated and committed transactions in the blockchain.

A ledger stores important factual information about business objects; both the current value of the attributes of the objects (world state), and the history of transactions that resulted in these current values (blockchain).

At Figure 7, there are two ledgers L1 and L2 and they have copies. As a matter of fact, even each channel has a copy of the ledger they are related to. Peer node P1 maintains a copy of the ledger L1 associated with C1. Peer node P2 maintains a copy of the ledger L1 associated with C1 and a copy of ledger L2 associated with C2. Peer node P3 maintains a copy of the ledger L2 associated with C2.

2.4.10. Application (A)

An application can interact with a blockchain network by submitting transactions to a ledger or querying ledger content. It has to follow six basic steps to submit a transaction.

1. Select an identity from a wallet
2. Connect to a gateway
3. Access the desired network
4. Construct a transaction request for a smart contract
5. Submit the transaction to the network
6. Process the response

At Figure 7, Organization R1 has a client application A1 that can perform business transactions within channel C1. Organization R2 has a client application A2 that can do similar work both in channel C1 and C2. Organization R3 has a client application A3 that can do this on channel C2.

2.4.11. Smart Contract (S)

A smart contract is code, which is invoked by a client application external to the blockchain network, that manages access and modifications to a set of key-value pairs in the World State. In Fabric, smart contracts are referred to as chaincode. A smart contract is installed onto peer nodes and instantiated to one or more channels.

At Figure 7, P1, P2, P3 have installed some smart contracts and C1 and C2 have instantiated some smart contracts.

P1 has installed the smart contract S5. Client application A1 can use S5 to access ledger via P1. S5 is instantiated on C1 so A1, P1 and O4 can all make use of the communication facilities provided by the channel.

P2 has smart contract S5 installed for channel C1 and smart contract S6 installed for channel C2. Peer node P2 is a full member of both channels at the same time via different smart contracts for different ledgers.

In a similar way, P3 has S6 installed on it. S6 is instantiated on channel C2, which is used to provide controlled access to ledger L2. Application A3 can now use channel C2 to invoke the services provided by smart contract S6 to generate transactions that can be accepted onto every copy of the ledger L2 in the network.

With smart contract, it's finished the examination of Fabric' key components from Figure 7.

2.4.12. Wallet and MSP

So far, the Fabric's key concepts were presented. Besides that, it's also important to review the concepts to develop client applications and smart contracts. In this sense, a wallet is a useful concept that helps applications to manage identifications and permissions.

A wallet contains a set of user identities. An application run by a user selects one of these identities when it connects to a channel. Access rights to channel resources, such as the ledger, are determined using this identity in combination with a Membership Service Provider (MSP).

MSP is important in the background but we won't enter to that level of detail in this Fabric review. We only need to know that MSP identifies which Root CAs and Intermediate CAs are trusted to define the members of a trust domain, e.g., an organization, either by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members, or through a combination of both.

When an application connects to a network channel, it selects a user identity to do so, for example ID1. The channel MSPs associate ID1 with a role within a particular

organization, and this role will ultimately determine the application's rights over channel resources. For example, ID1 might identify a user as a member of some organization who can read and write to the ledger, whereas another identity ID2 might identify an administrator who can add a new organization to a consortium.

2.4.13. Gateway and Connection

Gateways are also useful so that applications can avoid low level interaction. A gateway manages the network interactions on behalf of an application, allowing it to focus on business logic. Applications connect to a gateway and then all subsequent interactions are managed using that gateway's configuration.

A Fabric network channel can constantly change. The peer, orderer and CA components, contributed by the different organizations in the network, will come and go. Reasons for this include increased or reduced business demand, and both planned and unplanned outages. A gateway relieves an application of this burden, allowing to focus on the business problem it is trying to solve.

Gateways have to parameters: the connection profile, and the connection options. Using them allows an application to focus on business logic rather than network topology.

A connection profile describes a set of components, including peers, orderers and CAs in a Fabric blockchain network. It also contains channel and organization information relating to these components. A connection profile is primarily used by an application to configure a gateway that handles all network interactions. It is normally created by an administrator who understands the network topology.

Connection options are used in conjunction with a connection profile to control precisely how a gateway interacts with a network. For example, here is specified the wallet and the identity that will be used by the gateway on behalf of the application.

2.4.14. Chaincode

Fabric users often use the terms smart contract and chaincode interchangeably. In general, a smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. It is then packaged into a chaincode which is then deployed to a blockchain network.

A chaincode is a generic container for deploying code to a Fabric blockchain network. One or more related smart contracts are defined within a chaincode. Every smart contract has a name that uniquely identifies it within a chaincode. Applications access a particular smart contract within an instantiated chaincode using its contract name. When a chaincode is deployed, all smart contracts within it are made available to applications.

In most cases, a chaincode will only have one smart contract defined within it. However, it can make sense to keep related smart contracts together in a single chaincode.

2.4.15. World State

The world state holds the current value of the attributes of a business object as a unique ledger state. That's useful because programs usually require the current value of an object and it would be cumbersome to traverse the entire blockchain to calculate an object's current value. The world state is implemented as a database.

Let's review an example. Figure 9 shows a ledger world state that contains two states. The first state is: key=CAR1 and value=Audi. The second state has a more complex value: key=CAR2 and value={model:BMW, color=red, owner=Jane}. Both states are at version 0.

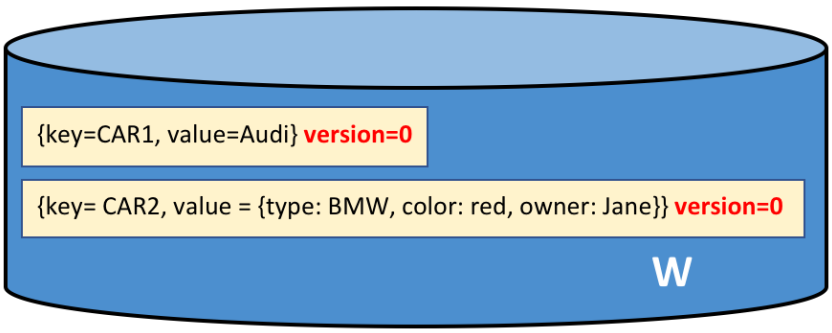

Figure 9 Example of a Fabric' ledger world state

An application program can invoke a smart contract which uses ledger APIs to get, put and delete states.

Applications submit transactions that capture changes to the world state, and these transactions end up being committed to the ledger blockchain. Applications are insulated from the details of this consensus mechanism by the Hyperledger Fabric SDK; they merely invoke a smart contract, and are notified when the transaction has been

included in the blockchain (whether valid or invalid). The key design point is that only transactions that are signed by the required set of endorsing organizations will result in an update to the world state. If a transaction is not signed by sufficient endorsers, it will not result in a change of world state.

2.4.16. Blockchain

The blockchain is structured as sequential log of interlinked blocks, where each block contains a sequence of transactions, each transaction representing a query or update to the world state.

Each block's header includes a hash of the block's transactions, as well a copy of the hash of the prior block's header. In this way, all transactions on the ledger are sequenced and cryptographically linked together. This hashing and linking makes the ledger data very secure. Even if one node hosting the ledger was tampered with, it would not be able to convince all the other nodes that it has the 'correct' blockchain because the ledger is distributed throughout a network of independent nodes.

The blockchain is implemented as a file, in contrast to the world state, which uses a database. This is a sensible design choice as the blockchain data structure is heavily biased towards a very small set of simple operations. Appending to the end of the blockchain is the primary operation, and query is currently a relatively infrequent operation.

Figure 10 shows an example of the structure of a blockchain and the parts of the belonging blocks.

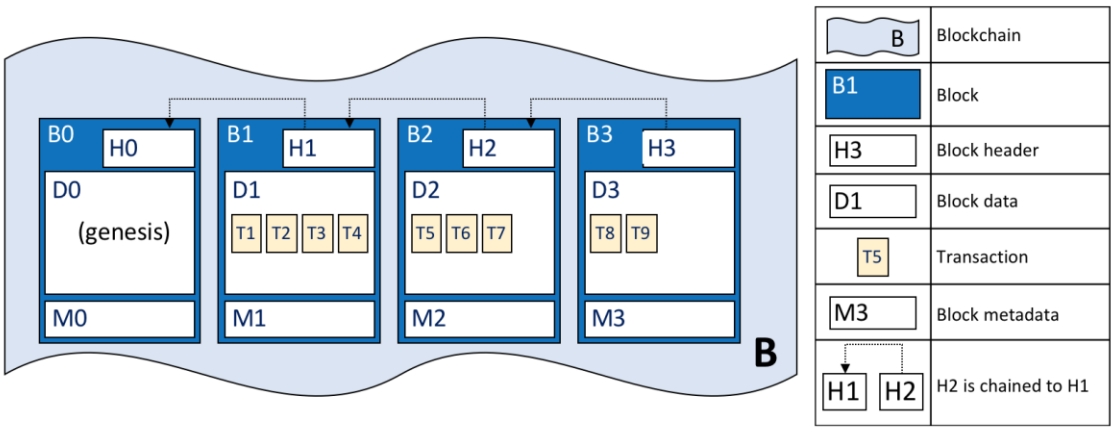

Figure 10 Fabric's structure of ledger blockchain

In the figure, blockchain B contains blocks B0, B1, B2, and B3. B0 is the first block, called genesis block. Each block consists of three sections: block header, block data, and block metadata. The block header has its current hash and a copy of the previous hash. The block data contains a list of transactions arranged in order. It is written when the block is created by the ordering service. The block metadata contains the time when the block was written, as well as the certificate, public key and signature of the block writer. Subsequently, the block committer also adds a valid/invalid indicator for every transaction, though this information is not included in the hash, as that is created when the block is created.

2.4.17. Transactions

As it was stated before on the application definition, an application can interact with a blockchain network by submitting transactions to a ledger. Transaction captures changes to the world state. Figure 11 shows the details of a transaction.

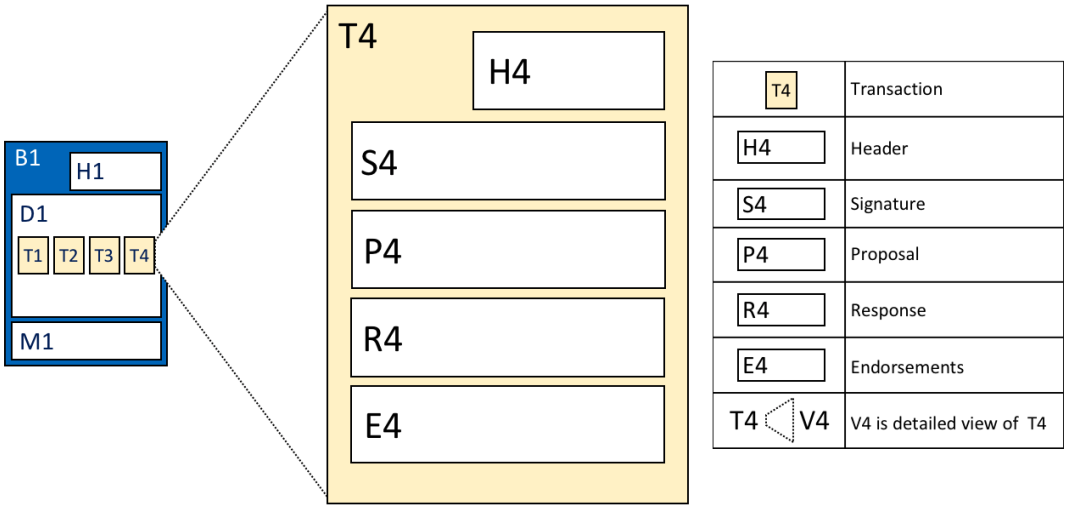

Figure 11 Transaction details inside a block data

In the figure above, transaction T4 in block data D1 of block B1 consists of transaction header H4, a transaction signature S4, a transaction proposal P4, a transaction response R4, and a list of endorsements E4.

- Header (H4): captures some essential metadata about the transaction e.g. the name of the relevant chaincode, and its version.
- Signature (S4): contains a cryptographic signature, created by the client application. This field is used to check that the transaction details have not been tampered with, as it requires the application's private key to generate it.

- Proposal (P4): encodes the input parameters supplied by an application to the smart contract which creates the proposed ledger update. When the smart contract runs, this proposal provides a set of input parameters, which, in combination with the current world state, determines the new world state.
- Response (R4): captures the before and after values of the world state, as a Read Write set. It's the output of a smart contract, and if the transaction is successfully validated, it will be applied to the ledger to update the world state.
- Endorsements (E4): is a list of signed transaction responses from each required organization sufficient to satisfy the endorsement policy.

All this transaction details will be handy when we review the implementation details.

Finally, one more aspect from transactions that is important to understand is the flow. We don't necessarily need to understand the flow at low level so Figure 12 showing a high level transaction flow will be enough.

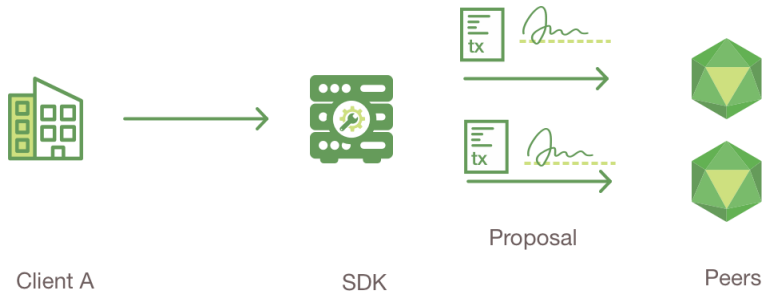

Figure 12 Fabric's transaction flow

At the figure, some client is sending a request. This request targets both peers, who are representative of different clients. Next, the transaction proposal is constructed. An application leveraging a supported SDK utilizes one of the available APIs to generate a transaction proposal to send to peers. The whole round flow is as follows:

1. Client A initiates a transaction.
2. Later, endorsing peers verify signature and execute the transaction.
3. Proposal responses are inspected.
4. Client assembles endorsements into a transaction.
5. Client assembles endorsements into a transaction.
6. Ledger is updated.

3 Implementation

3.1. Case of Study

3.1.1. Scenario

Through this chapter, we will describe the implementation of a particular case as a way to study blockchain in the medical context. This case was proposed by the author of this work and is about a simple example to apply blockchain for medical data sharing using the blockchain framework Hyperledger Fabric.

The case is about registering the medical data, especially the prescription of drugs, when a patient attends to a medical examination.

Certainly there is a lot of information to register during a medical examination and while a patient is being diagnosed, but in our scenario we will focus in drug administration and prescription.

The scenario starts when the medical examination record is being created. After that, the medical record is under observation and modification since the patient is being diagnosed. Later, after being diagnosed, the patient is prescribed and administered some drugs to heal and to be healthy. However, sometimes patient gets sick again and has to be diagnosed all over again. Finally, as inevitably happens sometimes, patient can die despite medical treatment. The Figure 13 shows the states that this process flows through.

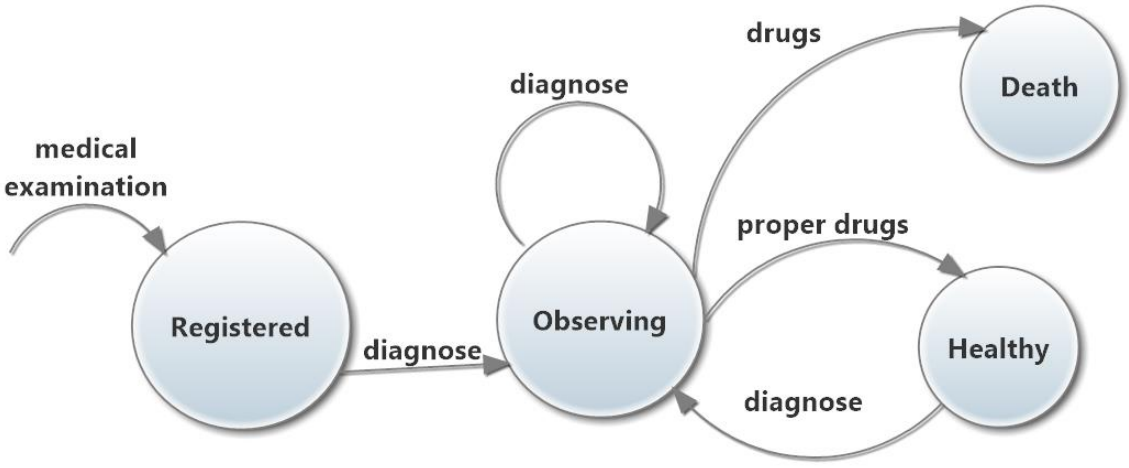

Figure 13 State diagram of a medical examination based on OMOP model

This diagram was made based on OMOP model, which was already presented previously at Figure 3. OMOP was the chosen model among the other models for being

the simplest one but yet widely used. We don't need more complexity related to medical models so OMOP will be enough.

From OMOP model only some tables are used: person, observation_period, drug_exposure, visit_ocurrence, and others indirectly. Moreover, not every column from these tables are used because these tables store every detail in more than 40 columns. We will use some columns only. From person, we will be using columns name, lastName, birthDate, ethnicity, gender and deathDate. From drug_exposure we will be using colums drugName, startDate, endDate, dosis, quantity and diagnosis. From observation_period we will be using doctor, and date.

Something implicit in this state diagram is the fact that the patient may want to get the examination with either the same doctor or another. Patients even can change hospital several times.

3.1.2. Prescription Drugs

Drugs can be classified into two categories. The first group is formed by drugs that are sold directly to a consumer without a prescription from a healthcare professional. The second group is formed by drugs that legally requires a medical prescription to be dispensed [23]. We will focus on prescription drugs.

Prescription drug abuse is the use of a prescription medication in a way not intended by the prescribing doctor. The prescription drugs most often abused include opioid painkillers, anti-anxiety medications, sedatives and stimulants. Early identification of prescription drug abuse and early intervention may prevent the problem from turning into an addiction. With the high rates of prescription drug abuse, a particularly urgent priority is the investigation of best practices for effective prevention, prescription and treatment [24] [25].

From the doctor's point of view, there is the case where drugs are deliberately and illegally prescribed to patients. On the contrary and for the most part, prescription drug abuse happens accidentally. Either the doctor made a human mistake or patient is an addict that finds ways to get drugs e.g. seeking prescriptions from more than one doctor.

3.1.3. Data Provenance

Besides abuse, the management of prescription drugs is delicate in general. For example, there are some drugs that are so strong or have side effects that make the

further diagnosis difficult. The situation gets worse when patients goes to many hospitals and the track of drug information is lost or vague.

In this sense, the goal of the scenario and its implementation is to help doctors and hospital managers to track the information related to the drugs administration and prescription. In the case of patients that attend more than one hospital, it would help doctors to review the drug prescriptions from other hospitals. Rich information can be obtained from sources collaborated by many doctors and hospitals.

Since many organizations (hospitals) can contribute to the addition or modification of data, modifications should be double checked to validate that modifications are under control of the organization that created the medical record or under the control of some given administrative organizations. In this scenario, only Org1 can add new records and the other organizations can modify with Org1 approval.

In chapter two under Prior work, it was introduced the importance of data provenance for this context. Data provenance provides the history of the origins of all data changes so this will be helpful to determine which changes were made by whom and when. For example, this system can help to audit the whole drug administration process that ended up in a tragedy with the unexpected death of a patient.

This work's system can also be helpful in the phase after the diagnosis was finished. It could help to determine what mistakes were made during the diagnosis or drug administration in order to collect data for its further study.

3.2. Architecture

3.2.1. Architecture Overview

In the chapter two, it was presented Fabric and every key concept and component that this work uses and takes advantage of but in this section we will review the architecture in pieces. Figure 14 shows the architecture components from a high level perspective.

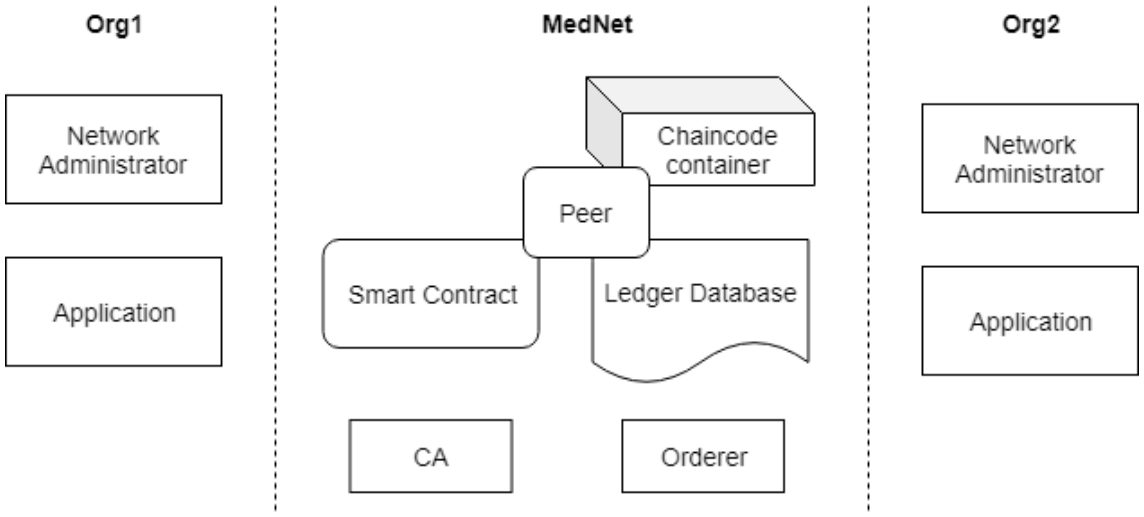

Figure 14 Interaction between MedNet network and organizations

In the figure, it's shown the two involved organizations Org1 and Org2, and the blockchain network, which is proposed for the case of study of this work, and will be called MedNet, standing for Medical Network.

On the Org1 side, we can notice the Network Administrator and Application. The network administrator configures the peers, the application and the smart contract in the network, among other tasks. The client application is the component that sends the transaction request and invokes the smart contract so ledger records the transaction.

On the Org2 side, like Org1 we have Network Administrator and Application too but are different since they belong to another organization. However, similarly to Org2, Network Administrator has to configure the network, and the Application is in charge of sending the transaction request that is defined for Org2.

On the MedNet side, we have Certificate Authority (CA), Orderer, Chaincode Container, Smart Contract, Ledger Database and Peer. CA and Orderer interact with the administrator to configure the network components. The chaincode is a container of the only one smart contract and defines the logic of the transactions with which the two organizations are going to interact. The ledger is where data and transaction logs are saved. Finally, the peer is a network node that hosts the ledger and a smart contract.

In MedNet there are actually two peers, one peer for each organization. Also the ledger have several copies along the network such as in every peer and in a channel. It happens the same for the smart contracts, which are installed on every peer node and instantiated on the channel. This level of detail will be reviewed on the next section about Network architecture.

Also, the tasks that the administrator is in charge of and how to run applications will be explained later in this chapter as well. Those tasks include the first launch of the network, configurations of peers, configuration of applications, installation of the smart contract, instantiation of smart contract, among other tasks.

3.2.2. Network

In chapter two, we already reviewed the Fabric' blockchain network in a generic way, so now it's time to take what we need from it. Our network design is a reduced version of what was presented before at Figure 7.

In the architecture overview, the main components were presented and now we will expand them into more detail imagining the network administrator already created and configured the components.

In the architecture overview's Figure 14, we presented the main components from the abstraction point of view, hiding details. The architecture is actually conformed by 2 peers, 2 applications and 2 CAs, which means one peer, one application and one CA for each organization. Smart contract and ledger also have more than one "copy" over the network and there are new components as can be seen at Figure 15 above.

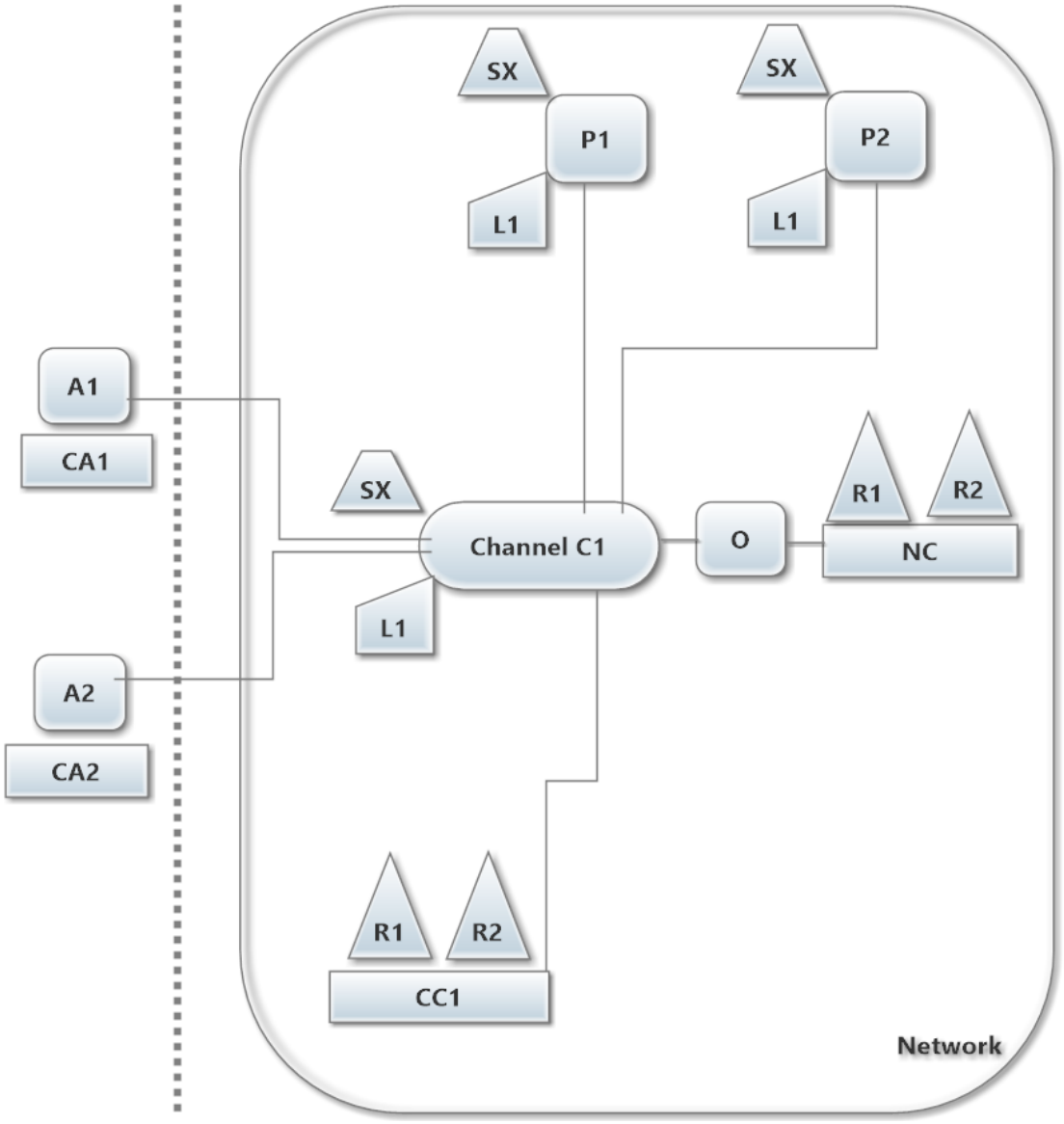

Figure 15 Blockchain network for the case of study

In this figure, the number 1 in the components means they belong to the organization Org1 and the number 2 on components means they belongs to organization Org2. In the left of the figure, we can see that there is one application A and one CA for each organization.

On the center of the figure, there is the only one orderer node. In a production environment ordering service is formed by more than one O node since that way decentralization is achieved the same way that other Fabric's components do.

On the right of the figure, there is the network configuration NC with is governed by organizations R1 and R2, which means they take part of consortium so they are able to create as many channels as they want but in this case we have only one since there

are only two organizations and it's not needed more for this case of study. The channel has its own channel configuration CC1 where R1 and R2 participate and are part of the endorsement policy but R1 was the one that created the channel.

On the top of the figure, each peer has its own copy of the ledger and its own smart contract SX installed.

Finally, again in the center, there is the channel C1, which is the primary communication mechanism among the organizations. The same smart contract SX that was installed on every peer node is instantiated on the channel. With this, all the components of our network are ready.

3.3. Development

3.3.1. Environment

The operating system used was Linux Ubuntu 18.04 LTS with latest Hyperledger Fabric available, which at the time of doing this work was v1.4. Docker was really important for the management of containers, which is for avoiding the heavy task of creating virtual machines and configuring environments.

Node JS was the chosen language to implement the client applications and the smart contract. In our proposal, there are two organization and every organization has to implement its own application. The smart contract is the same for both of them and the transactions defined inside it may be used completely or partially by the organizations.

3.3.2. Repository

The implementation of the proposal, and the development in general was done following the Fabric's recommendations. At the time this work was done, the last version of Fabric was v1.4 so that version of the Fabric repository was cloned. These are the Fabric source repository link and the customized one.

https://github.com/hyperledger/fabric-samples

https://github.com/dyave/hyperledger-fabric-tfm

3.3.3. Applications

From Figure 15, there are two applications A1 and A2, one for each organization. Application A1 composed by applications *createMed.js* and *queryMed.js*. And application A2 is composed by *updateMed.js*. A1 uses its applications to create medical examination records and to query and check the data provenance of the records. A2 uses its application to update the records and contribute with data. Below a fragment of the most important code is shown.

```
await gateway.connect(connectionProfile, connectionOptions);
const network = await gateway.getNetwork('mychannel');
const contract = await network.getContract('smartcontract');

const response = await contract.submitTransaction(transaction,
  doctor, checkId, person, drugExposure);
let med = CommercialPaper.fromBuffer(response);
```

This is just a modified fragment and it is presented omitting details for simplicity but it shows how an application uses the gateway to connect to the network and how the application invoke the smart contract when a transaction request is send. The last two lines are about the submission of a transaction request for creating the medical examination data and its processing.

The program *addToWallet.js* doesn't take part of neither of two applications but it has to be executed beforehand by the each user that runs an application. *addToWallet.js* is the program that the user will use to load his identity into his wallet, and the application will use this identity to create medical examination record on behalf of Org1 by invoking the smart contract. The following code is a fragment of *addToWallet.js*, which is similar for both Organizations.

```
const { FileSystemWallet, X509WalletMixin } =
  require('fabric-network');
const path = require('path');
const fixtures = path.resolve(__dirname, '../basic-network');
const wallet = new FileSystemWallet('../identity/user/wallet');

async function main() {
  const identityLabel = 'User1@org1.example.com';
  const identity = X509WalletMixin.createIdentity('Org1MSP',
    cert, key);
  await wallet.import(identityLabel, identity);
}
```

The *addToWallet.js* code is larger than this of course but as can be seen, firstly the fabric-network general package is being brought to scope together with basic-network, a sample network that is being adapted. Here the credentials are loaded into the wallet and some files are generated locally. Once those are created, applications don't need to run *addToWallet.js* anymore.

3.3.4. Smart Contract

The smart contract defines the rules between the two organizations R1 and R2. The transactions that applications are requesting are defined in the smart contract. Those are for adding a new medical examination records, for updating a record, for reviewing the existing data in the records.

Applications invoke the smart contract to generate transactions that are recorded on the ledger. So among the code of Fabric SDK to implement the logic, there is the API to communicate with the ledger's world state and blockchain. Figure 16 shows the class diagram for handling the ledger and to implement the smart contract logic.

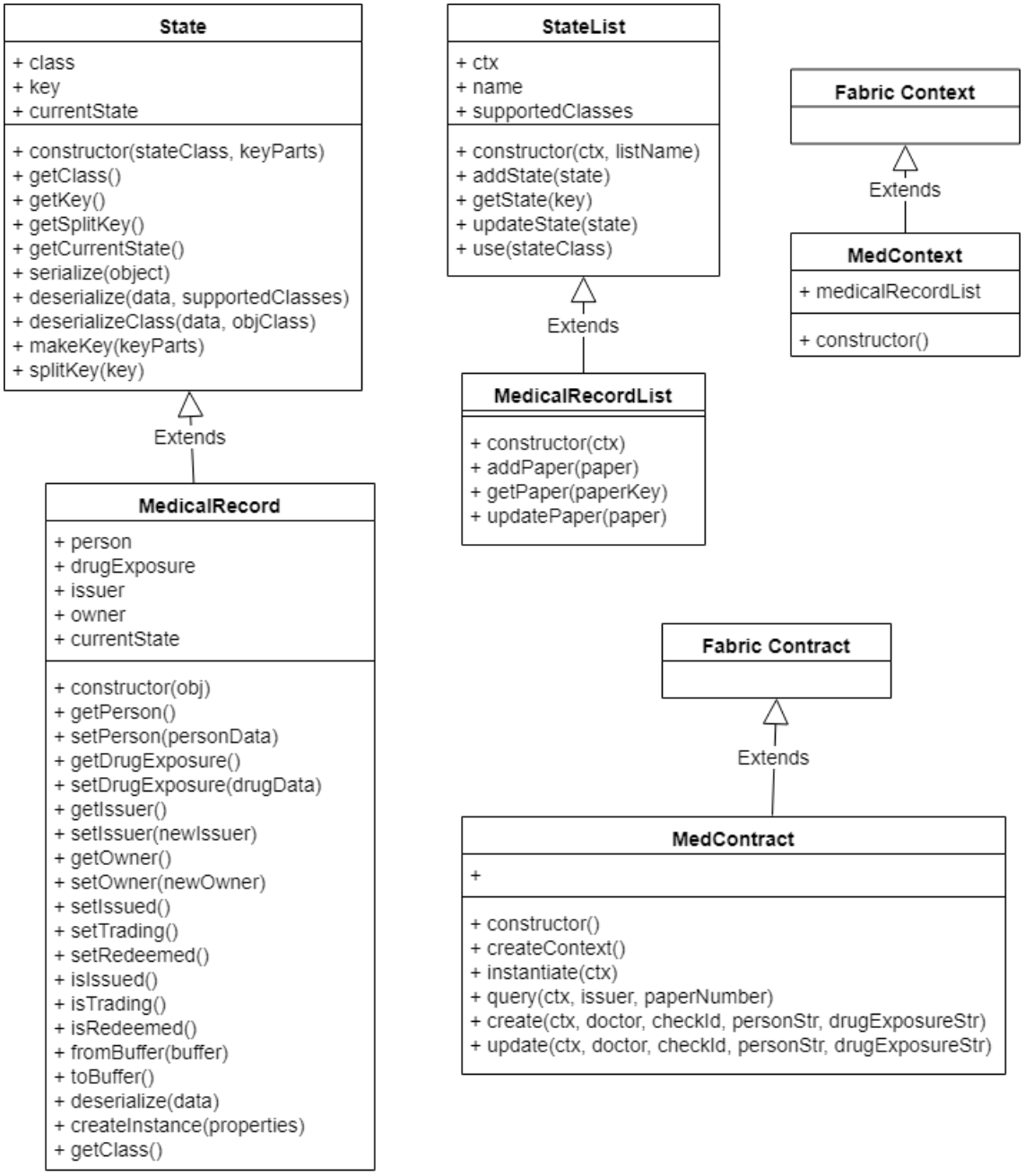

Figure 16 Class diagram for smart contract

From this diagram, notice that State and StateList are super classes for the interaction with the ledger. These super classes conform the ledger API so that the child classes adapt and override them. To highlight, MedicalRecord has the properties person and drugExposure and the respective getters and setters. MedicalRecordList is the list of medical records. Also, notice the Fabric Context and Fabric Contract, which comes with the Fabric's smart contract API, are necessary to implement the smart contract. At MedContract is defined the available transactions which are create, query and update a medical record.

Below is shown a fragment of the smart contract code. This fragments is about the transaction definition for the creation of a medical examination records.

```
async create(ctx, doctor, checkId, person, drugExposure) {
  let med = CommercialPaper.createInstance(doctor, checkId,
    person, drugExposure);
  med.setRegistered();
  await ctx.paperList.addPaper(med);
  return med.toBuffer();
}
```

The transaction create has a parameters the context, which is for interacting with ledger, and the parameters related to the medical data such as doctor, checkId, person, drugExposure. From there, and object that represent a record (in Fabric examples is called a paper) is created and its stated is being setting as registered so that it can finally be registered on the ledger. This is the way how the create transaction is implemented, but this is the basis and the other transactions are implemented similarly.

3.4. Configuration and Execution

To configure our blockchain network it's needed to configure the Fabric's components. Organizations and main components were already presented at Figure 14. Since there are at least two involved organizations, Hospital 1 and Hospital 2, many roles and users are expected. To keep it simple, we will reduce the roles to five, which are the most important: Network Admin, Org1 Admin, Org1 User, Org2 Admin, and Org2 User.

First, the network needs to be configured, that means launching the network and configuring the peers. Network admins are the responsible to do this.

After that, it's necessary to generate a wallet for each user so that it can be used as a way of identification each time an application is run. Finally, once everything is set up, users can execute applications as many times as they want. In the next sections, the necessary shell commands for each of the roles are shown.

The following network configuration was taken from a Fabric example called basic-network, and the peer configuration was taken from example PaperNet.

3.4.1. Network Admin

The following commands assume that Fabric's installation was already done. Here the admin network launches the network by executing start.sh. This script will create containers for Fabric's Orderer, CA, CouchDB, Peer, Logspout.

mkdir -p $GOPATH/src/github.com/hyperledger/

cd $GOPATH/src/github.com/hyperledger/

git clone https://github.com/hyperledger/fabric-samples.git

cd fabric-samples/basic-network

./start.sh

docker ps

docker network inspect net_basic

3.4.2. Org1 Admin

Now it's admin's turn to launch a logger and to configure the peer so that the Org1 User can be a member of the network. Otherwise, user won't be able to make any operation with the network.

cd $GOPATH/src/github.com/hyperledger/fabric-samples

cd commercial-paper/organization/magnetocorp/configuration/cli/

./monitordocker.sh net_basic

docker-compose -f docker-compose.yml up -d cliMagnetoCorp

docker ps

For a smart contract to work, peer needs to install the smart contract, and after that the smart contract has to be instantiated on the channel. At the time of instantiating, the endorsement policy is also specified. In this case, it's specified that Org1 has to endorse the transaction.

docker exec cliMagnetoCorp peer chaincode install -n papercontract -v 0 -p /opt/gopath/src/github.com/contract -l node

docker exec cliMagnetoCorp peer chaincode instantiate -n papercontract -v 0 -l node -c '{"Args":["org.papernet.commercialpaper:instantiate"]}' -C mychannel -P "AND ('Org1MSP.member')"

docker ps

Notice that by the time instantiating has finished, a new container was created for the channel. In this channel, lots of communication with the network happen so this is why it needs a separate container.

3.4.3. Org1 User

A user sends the transaction requests to the blockchain network by means of the application. And application is separated from everything so as a normal application it's needed to install its dependencies. In this case, the application was developed in Nodejs. After that, the program *addToWallet.js* has to be run so that the identification of the user can be added to the wallet, which is a one-time task.

cd $GOPATH/src/github.com/hyperledger/fabric-samples

cd commercial-paper/organization/magnetocorp/application/

npm install

node addToWallet.js

ls ../identity/user/isabella/wallet/

node createMed.js

After the wallet was generated locally, the peer has been identified properly on the network so the applications can be executed as many times as the user wants. Notice that the last command is running *createMed.js*, which is the actual application.

The application *createMed.js* has the medical data from a patient and his medical examination, so the application builds the transaction that is requesting to create a new medical examination record on the blockchain network. The application *createMed.js* invokes the smart contract because there is where transactions are defined

3.4.4. Org2 Admin

Similar to what happen with Org1 Admin, the Org2 Admin has to configure the peer. However, the smart contract doesn't need to since it's enough with the configuration that the Org1 Admin already did before.

cd $GOPATH/src/github.com/hyperledger/fabric-samples

cd commercial-paper/organization/digibank/configuration/cli/

docker-compose -f docker-compose.yml up -d cliDigiBank

3.4.5. Org2 User

Again, user needs to install the required packages so that the application works. After that, one-time *addToWaller.js* needs to be executed. After that, user will be able to execute any application.

cd $GOPATH/src/github.com/hyperledger/fabric-samples

cd commercial-paper/organization/digibank/application/

npm install

node addToWallet.js

node updateMed.js

Having finished with these commands for Org2, we are done with the whole configuration and our network and applications are good to go. Notice the last command *updateMed.js.* It is the application run by Org2 user, which is sending a transaction request to update a medical record.

3.4.6. Removing containers to reconfigure

When applications are modified, they just need to be rerun in order to take effect. However, when it's about a change in the smart contract, even if we stop and re run, it won't take effect.

Fabric can have many peers, smart contracts and organizations involved in its network so updating the chaincode can be a task carried out by someone specialized

like an administrator. Especially if it's required to modify a chaincode without stopping working or losing the already existing data in the ledger.

These considerations are going to be important in a production environment, but it's not our case. Therefore, the simplest way to have the chaincode changes take effect will be enough. To achieve this, we can remove all the containers and re run everything again. The Docker commands are the following.

docker rm -f $(docker ps -aq)

docker images | grep <name of image of chaincode instantiation>

docker rmi <image id>

The first one is for removing all Docker containers. The second one is for looking for the image id of the chaincode instantiation since we need to know its id for removing it. Finally, we accomplish the actual removal with the third Docker command by specifying the image id obtained from the previous command.

With the practice, one will notice that only the shell commands that generate the containers are needed to be rerun because the installations of packages from the applications don't need to be removed because they don't have nothing to do with smart contract changes. For example, the npm installation of application packages or the wallet installation don't need to be done again.

4 Results and Discussion

4.1. Results

4.1.1. Ubuntu Packages and Hyperledger Installation

The operating system used for Hyperledger Fabric v1.4 was Linux Ubuntu 18.04 LTS. On the other hand, the operating system for Hyperledger Composer was Linux Ubuntu 16.04 LTS. The installation on windows is discouraged.

Basically, both Hyperledger Fabric and Hyperledger Composer require to have installed the following software packages: cURL, Docker, Docker Compose, Go programming language, Node.js, NPM, and Python. However, it can gets complicated to install some intermediate software packages that are not compatible or need other extra dependencies according to the operating system. Also, it is required to change Linux configurations files such as the .profile file or define global terminal variables.

For this reason, it was really useful to take snapshots to the virtual machines in Virtual Box. For example, Figure 17 shows the installation history of Hyperledger Composer at Linux Ubuntu 16.04.

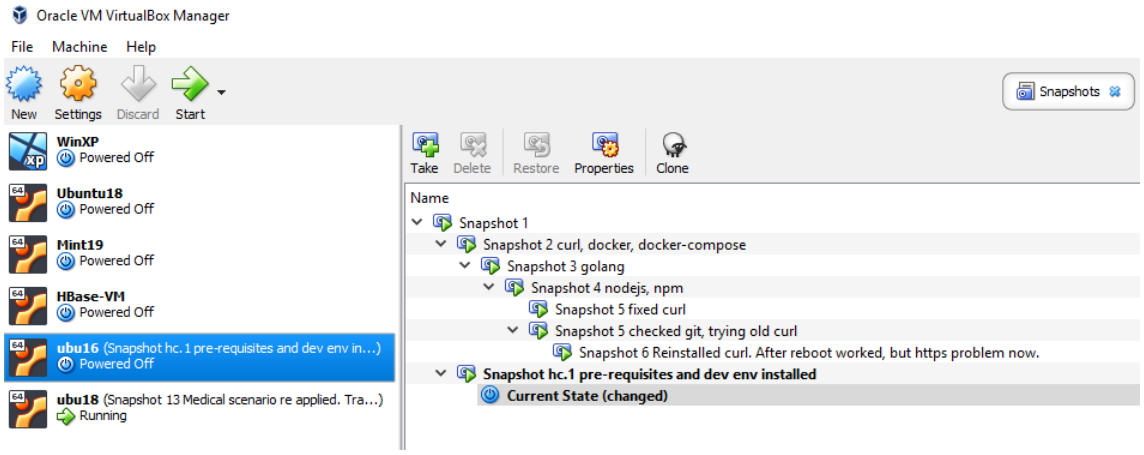

Figure 17 Installation history of Hyperledger Composer

On the figure, at the list of snapshots, can be seen that each snapshot was labeled with a message that explains its purpose, like a Git commit. This was useful for rolling back changes or to save a stable state of the virtual machine.

Hyperledger Composer was installed and tested but it was left behind for many reasons. Firstly, it is not really useful for the purposes of this work. Hyperledger Composer uses a different terminology than Hyperledger Fabric and seems to have a

more administrative business approach. For this reason, it wasn't worth investing more time with Hyperledger Composer. Secondly, Hyperledger Composer project started to be discontinued. In an announcement, developers let the community know that they will stop maintaining and bringing new features to Composer because they will focus on bringing the features straight to Fabric.

On the Hyperledger Fabric side, Figure 18 shows the installation history of Fabric at Linux Ubuntu 18.04.

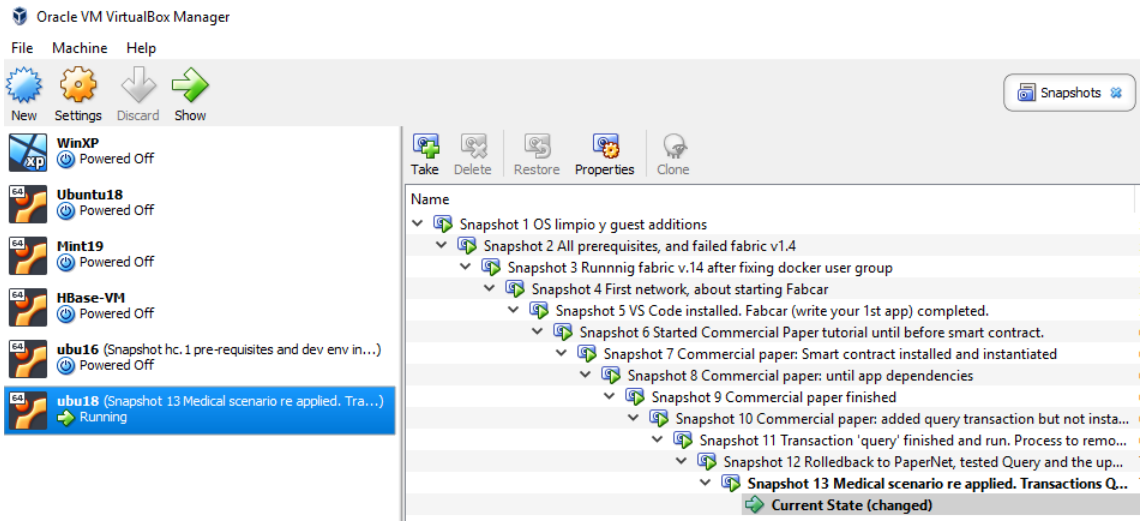

Figure 18 Installation history of Hyperledger Fabric

Taking snapshots enables the possibility to go back to a previous virtual machine state in case something goes wrong with the installation of packages or with the execution of Fabric binaries. This is something very usual because there are some commands that cannot be undone easily because they modify system configurations files. Thus, taking snapshots basically is like versioning with Git in Virtual Box. The only thing to consider is that each snapshot may take from 3 to 10 GB at the hard drive.

4.1.2. Network Creation

The network administrators' configuration tasks from both organizations have to be completed successfully before using the network. Docker easies these tasks and as a successful result Docker containers are created. We can think of a Docker container as a virtual machine. Table 1 shows the created Docker containers that represent a physical machines that take part on the network.

Table 1 Docker containers representing physical machines on the network

Name	MAC Address	IPv4
couchdb	02:42:c0:a8:50:03	192.168.80.3/20
orderer.example.com	02:42:c0:a8:50:04	192.168.80.4/20
cliMagnetoCorp	02:42:c0:a8:50:07	192.168.80.7/20
logspout	02:42:c0:a8:50:06	192.168.80.6/20
cliDigiBank	02:42:c0:a8:50:05	192.168.80.5/20
dev-peer0.org1.example.com-papercontract-0	02:42:c0:a8:50:08	192.168.80.8/20
ca.example.com	02:42:c0:a8:50:02	192.168.80.2/20

These network nodes are the components that conform out blockchain network. In our case, each node is actually not a physical machine but a Docker container, which is easier to handle and it's ok for a development environment.

The purpose of the container is based on the image it was created from. Figure 19 shows a terminal output with the generated Docker containers and the images they were based from.

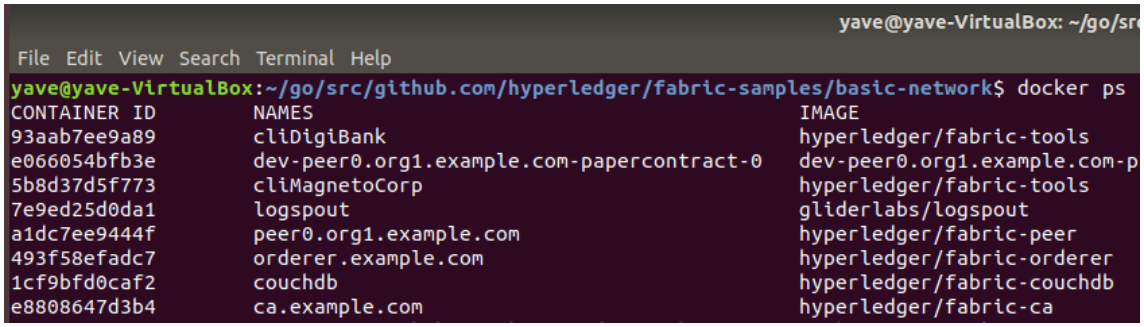

Figure 19 Docker containers after network configuration

As the terminal screenshot depicts, there are 8 containers that were created based on Docker images, which are pulled from DockerHub, a repository of images. These images have the most up-to-date version of the software for these Fabric components.

From the previous figure, the last four containers were actually the first ones to be created and they are ca.example.com, counchdb, orderer.example.com, and peer0.org1.example.com. All of them form the basic network.

The container logspout is for monitoring organization components. It collects the different output streams into one place, making it easy to see what's happening from a single window. This is really helpful for administrators when installing smart contracts or for developers when invoking smart contracts.

The containers cliMagnetoCorp and cliDigiBank are the peers for the organization Org1 and Org2 respectively. These were created based on a pre-built image in the hyperledger/fabric-tools Docker image.

Finally, the container dev-peer0.org1.example.com-papercontract-0 represents the instantiation of the smart contract on the channel so with this container the smart contract is available on the network.

4.1.3. Application

Our system proposal, MedNet, doesn't count with graphical user interface because it was not so necessary for the purposes of this work. The efforts were focused on Fabric itself than creating a web page, which shouldn't be difficult because Node JS is great to do that but it demands some extra time.

Basically, besides network configuration, for running the applications that invoke the smart contracts, it's necessary two more terminals. Figure 20 shows all the necessary terminals. The two from the top left are for launching a basic network, and for reviewing logs. The top right terminals represents the network administrator and user client from Org1, and the two from the bottom represent the same for Org2.

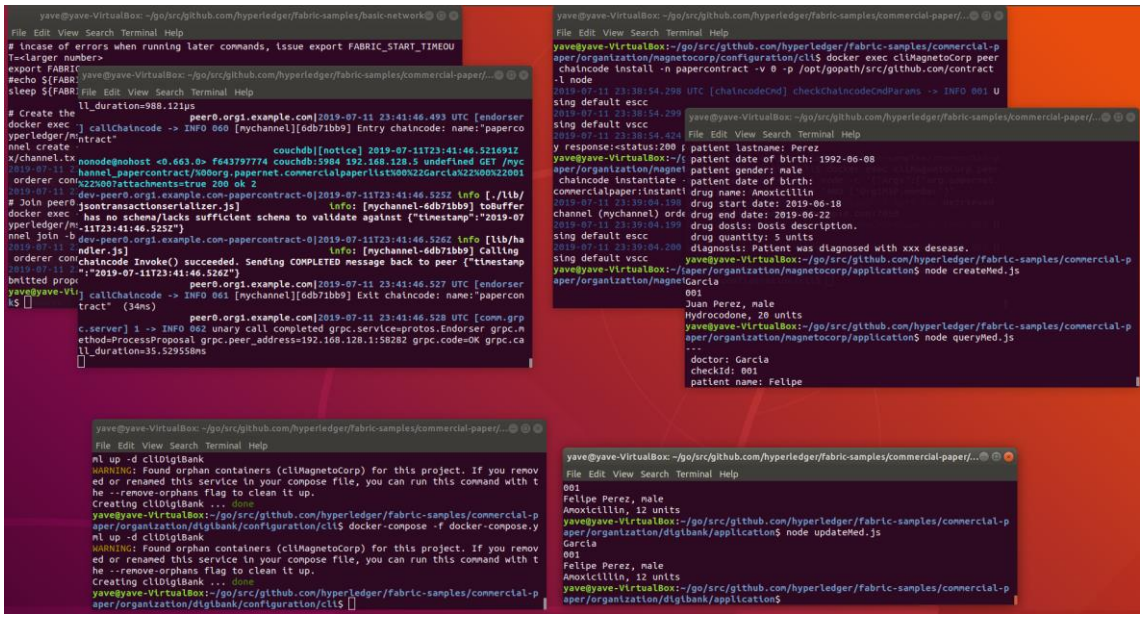

Figure 20 Terminals to configure network and run applications

This figure shows the required terminals, which potentially means the number of machines. If it weren't for Docker, running and testing the network would be a very heavy task.

What the terminal that represents Org1 user is doing is running *createMed.js* so that a transaction request to create a medical record can be sent. The sample input data to test the application and the smart contract is the following medical examination data.

Personal Data

- Doctor: Mario Garcia
- Patient Juan Perez
- Date of birth: 1992-06-08
- Ethnicity: White
- Gender: Male
- Allergies: None

Drug Exposure

- Drug Name: Hydrocodone.
- Dosis: 20 pills, 2 pills per day or in extreme pain starting 2019-06-18 and finishing 2019-06-22
- Diagnosis: Injured muscles from the back due to a car crash

This medical record was created by the Org1 (Hospital 1). However, due to a suspect case of drug abuse, the Org2 (Hospital 2), which took care of the same patient too, is recording the reduction of the quantity of pills. Figure 21 shows this scenario where Org1 creates the records, Org2 updates the dose information, and at Figure 22 Org1 query the up-to-date data.

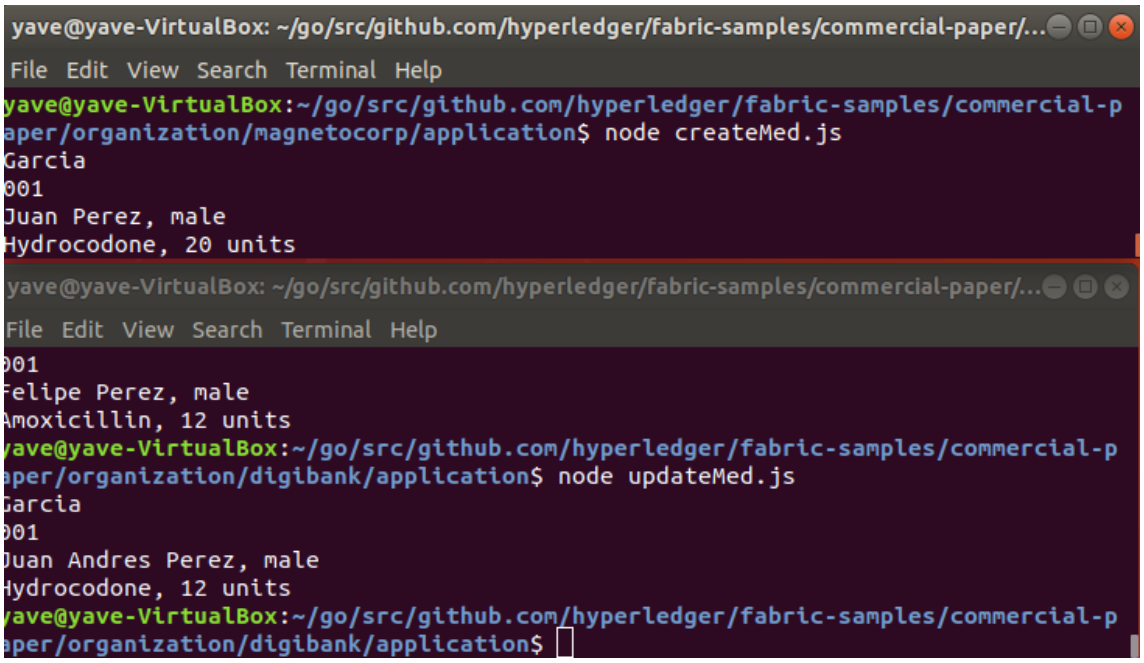

Figure 21 Execution of the transaction Create and Update medical record

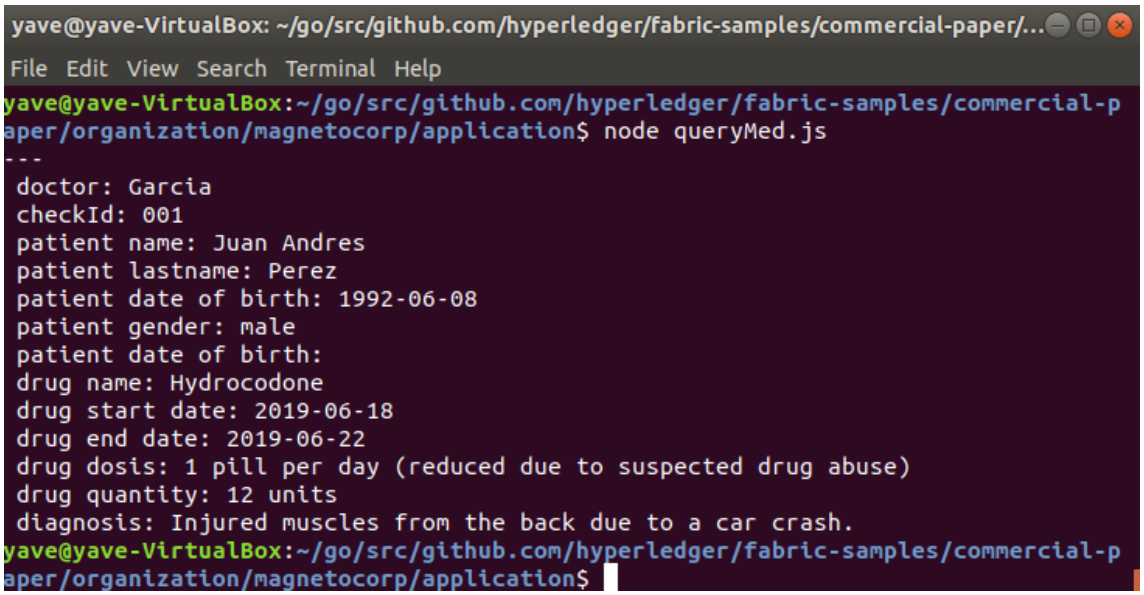

Figure 22 Execution of the transaction Query medical record

So as it was shown, in this scenario, Org1 creates the record, but Org2 contributes by updating it. In this case the endorsement policy enables every

organization that belongs to the channel to make modifications, so Org1 and Org2 can modify but only Org1 can create records. Endorsement policies can change over the time e.g. when the number of organizations increase or for privacy issues. Also as the figure shows, once *queryMed.js* application was run, Org1 can review the up-to-date records.

Similar to the query, data provenance had to be implemented as well so that the organization can query about the history changes. This data provenance was supposed to be implemented by getting the transaction log from the ledger's blockchain and filter the data related to the modification of medical fields. However, because of a matter of time, data provenance was not achieved in this work.

4.1.4. Smart Contract

Regarding the smart contract, the results were already presented together with applications. The smart contract works on the background so Applications are responsible to invoke them.

There is a way to test smart contracts by using Docker commands directly without the need of an application, but this level of automatized testing and development wasn't reviewed in this work.

Currently the smart contract offers the transactions for creating a medical record, to update and to query. The data provenance was not successfully completed because the Ledger's blockchain access to the transaction logs need some extra configuration and the interaction with Fabric Shim. In this work it was used the standard Fabric SDK only for developing the transactions.

4.2. Discussion and Recommendations

Good skills of Docker and Docker Compose are demanded since the configuration of images, paths and containers is achieved by modifying or creating .yml files, which are Docker configuration files. Also, the shell scripts to launch the containers related to the network can be modified and automated if the developer/operator has good devOp skills. There is a Fabric example called chaincode docker devmode related to this topic, which was not reviewed because of a matter of time, but it's highly recommended.

Regarding to the implementation, although Node, Go, Java and Python are the available languages to implement smart contracts and applications, there are some

recommendations to remark. First, Python and Java are not recommended since the code samples that are available on Fabric's repository don't contain Java or Python version. Second, Node seems to be the quick alternative to start playing around with smart contracts because there are Node versions in the Fabric's implemented samples and the initial documentation is under Node. Finally, for more advanced implementation of smart contracts, language Go is recommended. Fabric's advanced documentation, and more complex code are under Go so any advanced task will end up needing Golang.

Hyperledger Composer is a toolset built on the basis of Hyperledger Fabric. It was released to make the integration of blockchain applications easier with the existing business systems. Composer is as popular as Fabric, but it doesn't necessarily come with Fabric. Composer is supposed to help to rapidly develop use cases and deploy blockchain solution. However, the terminology that Composer manages is different than Fabric's. Thus, knowing Composer won't be so helpful for using Fabric. Besides the different terminology, Composer project was stopped and it seems it won't longer be supported because developers let the community know that it is becoming difficult to maintain Composer regarding to the constant changes of Fabric. For these reasons it's not recommended Composer.

In this work, a Virtual Box virtual machine with Linux Ubuntu 18.04 was used to have Fabric installed and configured. Despite managing Virtual Box snapshots and using Docker and Docker Compose, the troubles with the installation and configurations were almost inevitable since every environment is different. This difficulties leave between seeing that the installation and deployment of Fabric in a production environment would be many times harder. In a real environment, it will be required smart contracts developers, client applications developers, solution architects, networks operators and business professionals. Thus, it wouldn't be worth using Fabric in small scenarios.

5 Conclusions and Further Work

5.1. Conclusions

Cryptographically, blockchain is a structure that consists of consecutive chained blocks where each block contains a hash as identifier, a copy of previous block's hash, a timestamp and some data. It was originally presented as a part of the cryptocurrency Bitcoin, and from there it has been evolving. Combining it in a network, a blockchain network is a distributed public ledger where any transaction is witnessed and verified by network nodes. Its decentralization and security features have made researchers to develop various applications, infrastructures, and frameworks. Throughout this work, it was reviewed blockchain going from its origins to frameworks. Deciding which framework depends on the business necessities but choosing to have either permissionless or permissioned blockchain helps to decide. Permissionless blockchain allows unknown identities to participate in the network via Proof of Work, but for enterprise usage blockchain networks need to be permissioned instead for confidentiality. Thus, for this work, a permissioned one was used.

The case of study is about registering the medical data, especially the prescription of drugs, when a patient attends to medical examinations. It involves the data sharing of two organizations (hospitals) and is a simple design so that the implementation were more feasible. For designing it, related prior work and clinical models were reviewed but OMOP data tables were taken as sample of data to define the scenario and the state diagram.

Hyperledger Fabric, an enterprise-grade permissioned distributed ledger framework, was chosen. The Fabric's blockchain network have several components and the configuration can be complex. For the case of study, the launch of the network with all necessary components was successfully achieved. The implemented applications don't have a graphical interface since they are console applications but they are complete and ready to send transaction requests. About the smart contract, it currently define the transactions that allow to create, query and update medical records.

The data provenance for the case of study was taken into account and is about retrieving the history of changes in the medical records committed by organizations' members (doctors). The full implementation of data provenance wasn't successful. It's partially implemented since the transaction for querying only works for the world state and not for the blockchain transaction log.

To recapitulate, in this work blockchain was introduced into the medical context in a particular way. Firstly, Hyperledger Fabric was chosen as the blockchain framework. Secondly, taking into account a widely-used clinical data model, a medical case of study was designed so that it can be implemented with Hyperledger Fabric.

Nowadays, Hyperledger Fabric is one of the most important blockchain platforms so any serious evaluation of blockchain platforms should include it. The capabilities of Fabric make it stable and complete to address the resiliency, flexibility, scalability, performance and confidentiality challenges.

Although Fabric has an enterprise approach, which easies its application, it is a complex blockchain infrastructure, not only from the view point of learning it, but also from the view point of applying it. In this proposal it was implemented a simple example with Fabric. The roles of developer, operator and architect were covered by one person. However, for a more real example, it will be required smart contracts developers, client applications developers, solution architects, networks operators and business professionals. Thus, it wouldn't be worth using Fabric or any other framework in small scenarios. For applying a blockchain framework, it's expected a network where the business data is shared and consumed among many organizations and departments.

5.2. Further work

In this work, a simple example was implemented. So, a more complex example should be the next task to accomplish. For example, it could be about the mining of a patient data being several hospitals (organizations) the miners to contribute to medical record and all of them have to be validated by a consensus to decide the acceptance or rejection. For more complexity, more than two organizations, smart contracts and channels should be used.

At the time of installation, the Fabric's recommended steps were followed, which are the more appropriate for the development. However, for a production environment or for a performance study, the network components, peers, channels, among others, will have to be installed manually in physical machines. This task is not trivial and will involve network configuration, certificates and environment configurations. For this proposal, this kind of installation wasn't needed. As a matter of fact, these kind of tasks are made by a separate operator, which has nothing to do with development. Deploying in a production environment was something pendent to do.

The data provenance is pending to be finished. Although a progress was made with that, it may get more complex since it involves querying the history of logs on the Ledger's blockchain. In this work, standard Fabric SDK was used only but the data provenance implementation may imply to use Fabric Shim, which provides a chaincode interface and a lower level API to implement smart contracts. Thus, to implement data provenance and more complex smart contracts, the further work is about getting to know and use Fabric Shim.

6 References

[1] "Health Information and the Law," Who owns medial records: 50 state comparison, 2015. [Online]. Available: http://www.healthinfolaw.org.

[2] K. Mandl, P. Szolovits and I. Kohane, "Public standards and patients' control: how to keep electronic medical records accessible but private," BMJ, 2001.

[3] U.S. Department of HSS, "HIPPA Administrative Simplification: Regulation Text," 2006.

[4] The Office of the National Coordinator for Health Information Technology, "Report on health information blocking," 2015.

[5] M. Mettler, "Blockchain Technology in Healthcare," 2016.

[6] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System".

[7] X. Olleros and M. Zhegu, "Blockchain Technology: Principles and Applications," in *Research Handbook on Digital Transformations*, Cheltenham, Edward Elgar, 2016, pp. 225-235.

[8] M. Crosby, P. Pradan, V. Sanjeev and K. Vignesh, "Blockchain technology: Beyond Bitcoin," Applied Innovation 2, 2016.

[9] T. Swanson, "Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger," 2015.

[10] M. Valenta and P. Sandner, "Comparison of Ethereum, Hyperledger Fabric and Corda," Frankfurt School Blockchain Center, Frankfurt, 2017.

[11] G. Wood, "Ethereum: A Secure Decentralized Generalized Transaction Ledger".

[12] V. Dhillon, D. Metcalf and M. Hooper, "The Hyperledger Project," in *Blockchain Enabled Applications*, Berkeley, CA, Apress, 2017, pp. 139-149.

[13] S. Paraíso Medina, "Método de Normalización de Datos y Abstracción de Consultas Basado en Estándares Médicos," Madrid, 2018.

[14] G. Zyskind, O. Nathan and A. Pentland, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," IEEE CS Security and Privacy Workshops, 2015.

[15] L. Kish and T. Topol, "Unpatients — why patients should own their medical data," Nature Biotechnology, 2015.

[16] T. Gunter and N. Terry, "The Emergence of National Electronic Health Record Architectures in the United States and Australia: Models, Costs, and Questions," Journal of Medical Internet Research, St. Louis, MO, 2005.

[17] L. Chen, W.-K. Lee, C.-C. Chang, K.-K. R. Cho and N. Zhang, "Blockchain Based Searchable Encryption for Electronic Health Record Sharing," Future Generation Computer Systems, 2018.

[18] A. Azaria, A. Ekblaw, T. Vieira and A. Lippman, "MedRec: Using Blockchain for Medical Data Access and Permission Management," 2016.

[19] Y. Simmhan, B. Plale and D. Gannon, "A survey of data provenance in e-science," ACM, Ney York, 2005.

[20] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat and L. Njilla, "ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability," IEEE, 2017.

[21] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. Weed Cocco and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," Eurosys, Porto, 2018.

[22] C. Cachin, "Architecture of the Hyperledger Blockchain Fabric," 2016.

[23] A. Millar, "Defining Drugs," Pharma Technology Focus, 2018.

[24] D. M. Qato, J. Wilder, L. P. Schumm, V. Gillet and G. C. Alexander, "Changes in Prescription and Over-the-Counter Medication and Dietary Supplement Use Among Older Adults in the United States, 2005 vs 2011," JAMA Internal Medicine, 2016.

[25] W. Compton and N. Volkow, "Abuse of prescription drugs and the risk of addiction," Elsevier, 2006.