SAPIENZA
UNIVERSITÀ DI ROMA

Permus: a new Blockchain exchange system

Facoltà di Ingegneria dell'informazione, informatica e statistica

Laurea Magistrale in Engineering in Computer Science

Candidate

Davide Gimondo

ID number **1650567**

Thesis Advisor

Prof. **Silvia Bonomi**

Academic Year 2018/2019

Thesis defended on July 24, 2019
in front of a Board of Examiners composed by:

Prof. Tiziana Catarci (chairman)
Prof. Silvia Bonomi
Prof. Luca Iocchi
Prof. Leonardo Lanari
Prof. Massimo Mecella
Prof. Daniele Nardi
Prof. Riccardo Rosati

Permus: a new Blockchain exchange system
Second level master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: November 23, 2019

Author's email: davegimo@gmail.com

Abstract

This thesis discusses the developement of Permus, a bartering system that makes use of the blockchain and all its cryptographic properties.

The first part of the thesis is an analysis of the minimum requirements of the application and how the architecture is structured. The second part, on the other hand, goes into detail on how the project was developed, its characteristics and which technologies were used.

The choice to call the Permus system is simply the "Latinized" translation of the Italian word "permuta", which means exchange.

Acknowledgments

I want to thank my family first of all, and then all my friends. I wouldn't have reached all the goals I set in the past two years without their support.

Contents

Chapter 1

Introduction

1.1 Bartering in modern society

In the modern society, economy is fundamentally based on the exchange of money for goods and services. Money stores value in a predictable way over time, it is not perishable and it is universally accepted as it is used as unit of account, the thing that goods and services are priced in terms of. In modern economies the unit of account is usually a currency, for example, the Euro in the European Union, but it could be a type of good instead. In the past, in absence of currencies, items would often be priced in terms of something very common, such as staple foods ('bushels of wheat') or farm animals.

Historically, the role of money as the medium of exchange has often been viewed as its most important function by economists. Adam Smith, one of the founding fathers of the discipline of economics, saw money as an essential part of moving from a subsistence economy, or autarky, to an exchange economy. In a subsistence economy, everyone consumes only what they produce. But it is far more efficient for people to specialise in production, producing greater amounts of one good than they need themselves and then trading with one another. In the absence of money, societies have to use some kind of "barter" system in order to exchange goods and services between individuals. Swapping goods or services between individuals works in very small communities, but the exchanges that people in the modern economy wish to carry out are far more complicated: when large numbers of people are involved and, perhaps more crucially, when the timing of these exchanges is not typically coincident, the use of barter systems become unpractical. Unless it is supported by modern information technologies, of course.[21] [23]

When it comes to solve simple problems, people seems to prefer using simple mechanisms instead than complicated ones. A "one to one" swap of goods or services between two individuals can be convenient and simple enough. That's why barter systems are common in small and less advanced communities. But already, when individuals need to swap goods or services in exchange of the "promise" to receive something in the future that would compensate the swap, things start to become a little bit complicated. Unless every swap is immediately resolved with a simultaneous

exchange of goods or services one has to keep track of all past exchanges where one of the exchanged objects was simply a "promise" to have something in the future. In small communities, the problem of making sure that old debt is honored is often not trivial at all. Not fulfilling an agreement is in fact an issue that could sometime raise difficult to resolve misunderstandings between otherwise friendly individuals, helping increasing conflict among the members of a community.

Modern information technology, although trough complex means, proved to possess the ability to provide simple solutions to many problems individuals seems to have in their daily life. Managing important personal data, even in a mobility environnement, appears not to be a problem any more in this first part of the 21st Century, an epoch characterized by the pervasive availability of technologies that enable individuals to create, manage, access and exchange information anytime and virtually anywhere with anyone on the planet.

This consideration lead the author to explore the possibility to create a Software System that could enable small communities to implement a simple and efficient barter system able to overcome some of the problems inherent in this method of exchange.
Such a system should be secure, because otherwise no one would trust it, and simple enough in performing the required functions, otherwise no one would choose to utilize it in the first place. It should also be accessible from modern mobile devices, perhaps as a specialized application for performing some simple operations needed "on the road".
In small communities trust is earned and maintained by personal relationships. In such environments there is no place for anonymity. For this reason, a software system that has the ambition to help people manage barter activities has to positively identify any subject involved in the exchanges and has to make sure that all transactions are publicly recognized in order for them to have some sort of binding value based on the community norms. Of course that doesn't mean that the object of all transactions have to be public. The system should also provide the means of record and maintain information about swaps whose details are known only by the involved subjects. This feature is expected for privacy reasons of course, but it is also useful to keep other users from accessing data not relevant to them.

1.2 Quantification problem for bartering

Quantifying the value of an asset can sometimes be very complicated.
Nowadays, the market does the price, in the sense that on the basis of the relationship between supply and demand it is possible to estimate the economic value so as to satisfy the majority of buyers.
In the case of bartering, the situation is slightly different. First of all, you have to think that during the exchanges the two people involved are trying to maximize their earnings while helping the other person at the same time.

Therefore, these exchanges must continue over time as it benefits both sides to continue to operate in this way, so it would make little sense for one side to try to cheat the other.
For this reason, during the development of an application that allows the exchange between different users who know each other in real life, **the agreement protocol on quantities is strictly live.**

Figure 1.1. A graphical example of "honest" bartering

Chapter 2

Requirements

2.1 Requirements elicitation

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is an important aspect of project management.

Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Energy should be directed towards ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements.[14]

For the creation of a barter service, it is therefore essential to make a list of requirements so that during development it is clear what are the needs for the service to be created efficiently:

- The service must provide an authentication system for the user in order to be able to log-in.
- The service must allow users to communicate so to create new exchange transactions.
- All the transactions are of public domain, but users may choose whether the description of the exchange to be public or private.
- All the transactions are stored by a central server.
- Confirmed transactions must be consistent: this means that they can not change during time.
- Transactions may be of different types according to the needs each user has.

- The back-end service must recognize if the user making a request is enrolled in the system and also is "trusted" (Identity management).
- User must be able to recover and check old transactions, of himself or from other people as well.
- Each user will store locally all the transactions and will update from the central service during time.
- Each user has to be able to sign a transaction with a **private key**.
- Each user has to be able to encrypt the content of a private transaction.

Furthermore, there are **additional requirements** which are less technical that are analyzed more specifically in the next sections.

2.2 Deferred Bartering

Deferred bartering can be seen as the “promise” to compensate individuals in the future, after that they have received some goods or services by somebody.
This concept, although it is not officially spread, is often used between groups of friends who organize on whose turn it is to bring food and drink to parties, or when you have to pay a bill at the restaurant.
So the idea of having non-synchronized barters, would be to keep track of past "actions" visible to all users involved, so that it is easy to determine who will be the next person in charge.
There are smartphone applications, such as "Wiebetaaltwat" in the Netherlands, that do something similar by keeping track of who paid the most in a common till. This type of app is perfect for student houses or travel groups that need to share expenses, so in this way everything is divided equally.
Introducing deferred bartering, slightly changes the target for which such a system is designed, in fact we no longer talk about small groups of people but associations of people or large communities instead.

2.2.1 Compensation

A direct consequence of defered bartering is the concept of **compensation**. When there are several exchanges between two subjects over time, there will be a time when the two will be "in even" or some of them will be in debt to the other.

Therefore, it is important that a system that supports this type of exchanges also gives users the possibility of being able to pay off old swaps.
This means - also - that the system must show old transactions a subject had with a specific user.

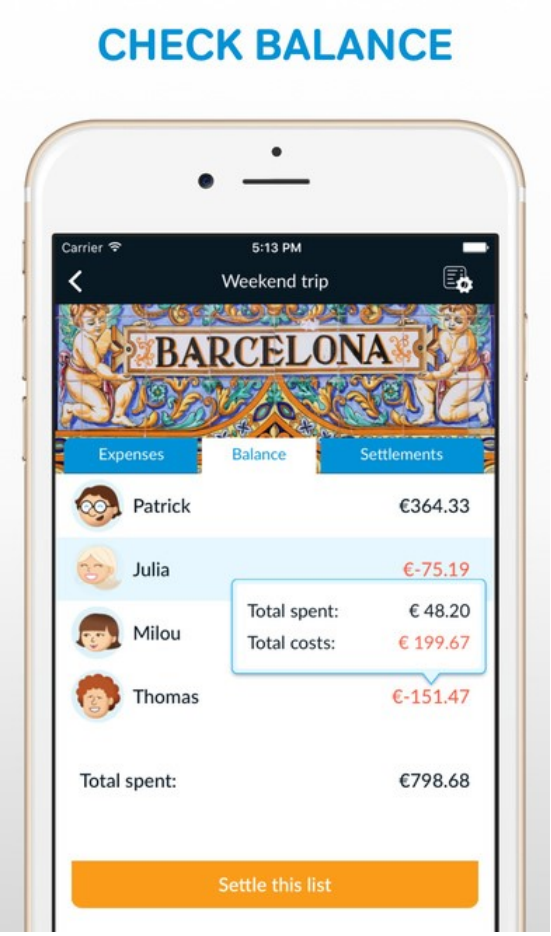

Figure 2.1. "Wiebetaaltwat" is just one of the many bill-sharing apps, it shows who has payed the most so far or who still has to pay off his debts.

2.3 Public and Private exchanges

To keep track of the exchanges made between users belonging to the system, it is essential to model them as transactions so that they can be stored. This type of exchanges can be seen as public or private.
The main difference between the two is that the information of a public exchange is visible to everyone, while in private exchanges there are information which are accessible only to the users involved.
In addition, you should consider that the user can always decide whether to make a public or private exchange, but there are cases where private exchanges are not allowed. For example, the user who represents a central institution in the system is not allowed to perform private exchanges because everyone must be given the opportunity to see what the institution did. As a result, private exchanges mainly take place between subjects (people) belonging to the system.

2.4 Internal Exchange Unit (IEU)

Many imposed limitations from using bartering for the exchange of goods and services between subjects who are members of a community, can be fixed with the adoption of transactions that make the use of an Internal Exchange Unit (IEU), a virtual coin whose value is universally accepted by the members of the community and can only be used inside of the system, giving it a local scope.
The exchange mechanism when IEU are used is very similar to barter swaps: the two parties agree on the value of what is exchanged, by taking into account the exchange unit instead of another good/service.
Just like explained in [1.2], the agreement for the quantification of the price of an object is **strictly** managed live between the two parties. The client service is just a way for the users to save a transaction and keep track of it in the future.

There is no need to explain the advantages of this approach with respect of bartering, although it is important to highlight some peculiarity that characterize an exchange system that makes use of a virtual coin in Permus:

1. All the transactions that involve IEU are public, certain and chronologically sorted.

   Public transactions, so that it is possible to prevent "double spending" and everyone is allowed to check the legitimacy of the IEU flow.

   Certain transactions. In Permus every transfer transaction is accompanied at least by a digital signature so to certificate the will of the subject to perform the operation.

   In some cases the transaction expects the digital signature of the receiver so that the will of the latter is certified as well.

   Chronologically sorted transactions, as a grant of flow consistency of the exchanged goods.

   Every transaction is provided of a timestamp that certifies the moment when the transaction has been pushed in its block.
   Every transaction is immutable and it is granted by the blockchain's features.

2. Every user has the possibility to spend only the IEU he is in posses of, which means in limit of the saldo between the IEU received and the ones spent.

   Every subject belongs to the system, without considering the one discussed in point 2, can only spend the IEU possessed. In every moment, the saldo between the received and transmitted IEU can not be lower than zero.

2.5 Non-functional requirements

In addition to the requirements set out above, the system needs non-functional requirements that improve the quality of the entire service.
These requirements are:

- **Data integrity:** the data that is stored must be consistent.
- **Security:** malicious users can not gather personal data from others.
- **Availability:** every user must be able to get access anytime.
- **Performance:** each operation must be fast and come to an end.
- **Usability:** the system must provide an easy navigation interface for the users.

2.6 Accessibility

A very important prerequisite for the system to be effectively usable, is to be accessible to all those people who want to access the service.
This means, first of all, being able to recognize which technologies are mostly used nowadays by potential users which remind an electronic payment and then do several considerations. Those technologies are:

- **Computer**
- **E-card**
- **Smartphone**

These three tools have certainly become accessible to a large part of the population, each is available to at least one of them and are in fact used every day by millions of people.
Consequently, it is necessary to make assessments of the pros and cons of each tool in order to see in the future which will be the preferred to work on.

Computer

The first tool that comes to mind for this type of technology is definitely a computer, especially because the technologies involved in the system are the blockchain and a PKI certificate.
The problem of the computer is represented by portability, although there are lightweight notebooks, this could be a disadvantage for users since not everyone goes around with a computer for the whole day.
Although it can be seen as a disadvantage for normal users belonging to the system, the computer can still be used by the central authorities of the community or by shops. For example, the first time a user wants to register into the system, he will go to the central authority to gain access, or a shop will be able to perform the transaction directly from the computer.

E-card

Unlike a computer, an e-card is extremely portable and can contain the information needed to perform a transaction. In fact, through NFC technology, it would be possible to install the certificate of each user inside the card.
In case a user wants to buy something inside a shop, it would be enough to use the electronic card to accept the transfer from the shopkeeper.
A problem could be that two users can not perform exchanges between them because a simple card containing personal information can not run anything.
This issue could be easily solved by automated teller machines (ATMs) where you have to place the card on a specific NFC reader.

Smartphone

Smartphone represents the union of the two technologies explained above, in fact now every phone supports the NFC and the screen can have a decent client application. The only problem would be to choose the best way to develop the application: at the moment operating systems for smartphones are dominated by Android and iOS, which would mean having to program two native applications, resulting to be very expensive.
An alternative to this scenario could be done using existing applications where its usage can be customized using botchat and one-time passwords, such as Telegram.

2.7 Use case diagram

Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.
They also help identify any internal or external factors that may influence the system and should be taken into consideration.
Finallyy, it provides a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented. [15]

Before implementing the whole system, it was important for the author to clearify which were the features and possible scenarios that a potential user needs while using the service.
The use case diagram is in this case the perfect choice to represent those features, because all the functional requirements shown in the diagram are the minimum attributes that make the service valuable (Minimum Valuable Product).

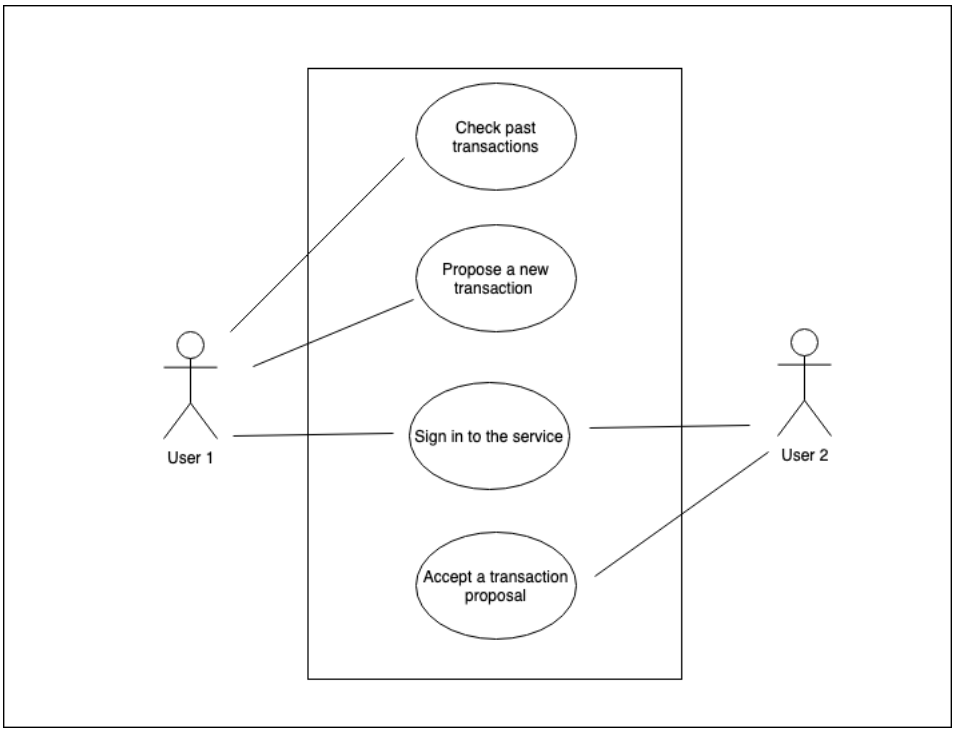

Figure 2.2. Use case diagram, showing what the service allows to do the user

As a proof of concept of the system and from a user's point of view, these are the basic actions that must be implemented in order to meet the minimum requirements. In the diagram two users are represented, both can connect to the client system and communicate through the proposal and acceptance of exchange transactions.

Chapter 3

System design

Before developing **Permus**, it was important to follow a pattern of design that would have helped to clear out the ideas and to follow a roadmap.
A 3-tier architecture is a type of software architecture which is composed of three "tiers" or "layers" of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Usually, it gives flexibility to development teams by allowing them to update a specific part of an application independently of the other parts.[7]

- **Data tier:** The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms.

- **Application tier:** The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing.

- **Presentation tier:** This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer which users can access directly (such as a web page, or an operating system's GUI).

This pattern was used for Permus, where each layer is taken into account is described through different techniques of description which will be explained in the next section.

3.1 System Architecture

"Software architecture description" is the set of practices for showing, communicating and analyzing software. It gives a global understanding of a complicated system showing its interaction with the environment, but also encourages the design of alternatives when the system can be improved.
Software description becomes helpful, for example, when a development team has to show to the customer the structure of a project or other important components that belong to it. The architecture of a complex system is necessary and has different purposes:

- **Communication:** it defines the global meaning of a system and gives insight on how to use the system
- **Analysis:** it answers questions about the system
- **Construction:** detailed design by developers of the technologies involved

The models used in the project as a reference point are the physical view, process view and the scenario. Those viewpoints formalize the idea that there are different ways of looking at the same system, and at the same time they enrich the set of information about it.
For Permus, it has been decided to focus on these models in order to give a clear overview to the reader on how the client and server work and communicate, and what are the main actions of an internal subject in the system. Every viewpoint is characterized by a set of concerns, a set of stakeholders, conventions and correspondence rules that link the model together. A stakeholder represents the subject whose description is addressed to, the view explains what type of description is given and, finally, the concerns point out what is the aim of the given description.

3.1.1 Physical view

The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components.[1]

The whole system is divided into two different parts: back-end and client side.
In addition there is a third party which is a Certificate Authority responsible for issuing certificates to every client that wants to join the service.

Blockchain in the system

Before getting to the heart of architecture, it is good to explain why the blockchain was chosen as the main component of this service.
Thanks to its cryptographic properties, it is possible to record countless transactions between different users with the certainty that they will be immutable over time.
In addition, as the architecture is structured, the blockchain will be maintained by each user locally so to lighten the constant updates during each session.

3-Tier Architecture

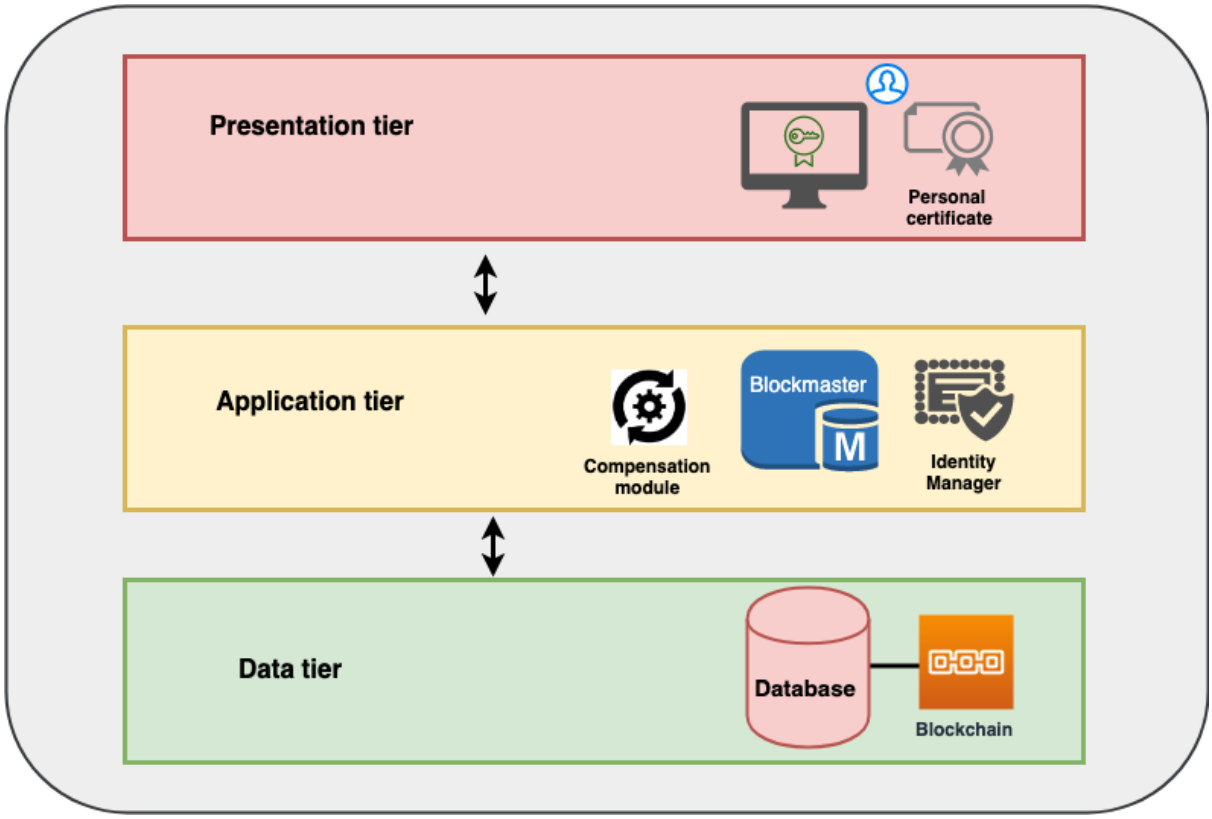

Figure 3.1. 3-Tier Architecture of the system.

Physical view

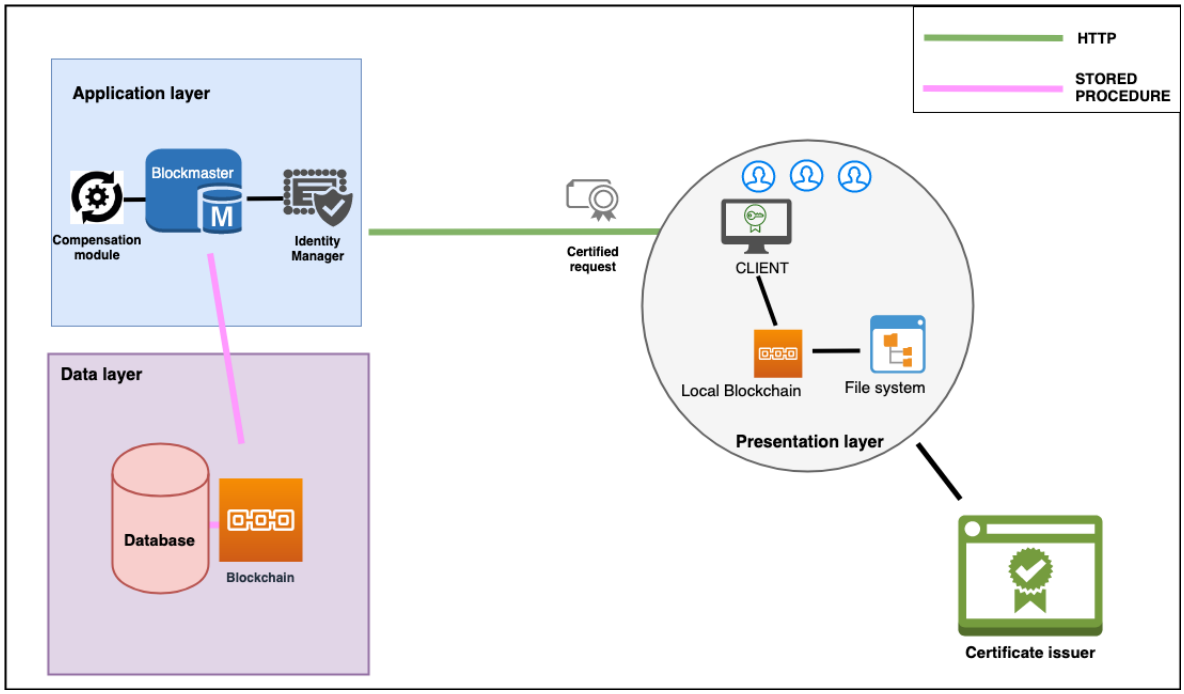

Figure 3.2. Another point of view for the architecture of the system: the physical view.

Blockmaster Back-end

In the back-end the Private Blockchain relies on the database and the **Blockmaster** is in charge for the communication with the client. The communication between the two parts is done through REST APIs making use of HTTP protocol and the client is continuously asking for an update of the blockchain.
The communication between the Blockmaster and the database containing all the blocks, works with stored procedures instead.
Unlike classic blockchain approaches, where the update of a chain is peer-to-peer and not centralized, the blockmaster is the only source for the system to get an update.

Compensation Module

The compensation module is a software component belonging to the application tier. Its task is to keep track among all users of the debts that each has to another subject, and go to scale old transactions when there are new compensations.

ID Manager

There is an ID Manager in the backend. In particular, any request that comes from a client must be validated. It is only valid if it is signed with a certificate from a

subject who has previously been enrolled in the blockchain.

During the enrolment, a new subject asks the Blockmaster to be enlisted. The blockmaster accepts the request only and only if the certificate:

- it has been issued by a specific (trusted) Certification Authority
- the subject string contains the basic elements required by the blockchain as a specification (tax code, name, email)

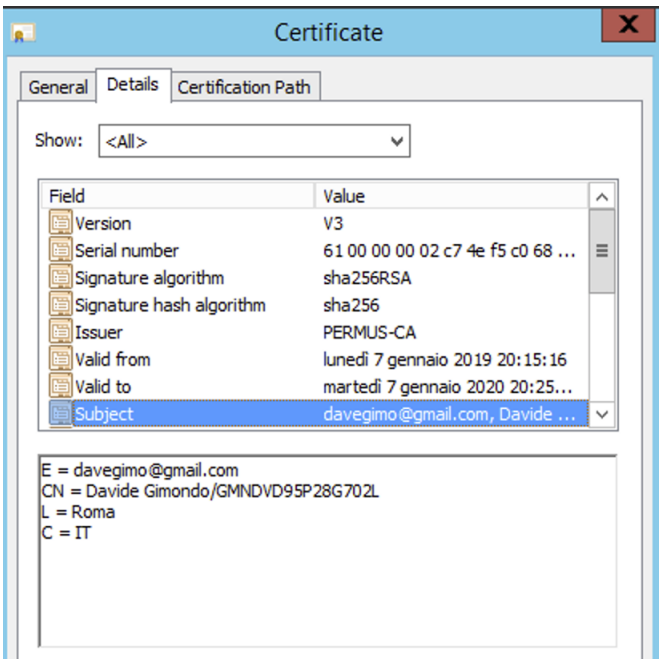

Figure 3.3. An example of a certificate issued containing the requested information.

Storage

As can be seen in Figure 4.1, the blockchain data is stored in two different ways, both online (database) and locally (file system).
The online version is nothing more than a SQL table containing all the records belonging to the blocks, which will be delivered to all users who will have to update their personal ledger.
The blockchain that instead is maintained locally by each user, is located in an ad hoc folder. It represents the reference point for each user in Permus and will be the part of information that will be consulted for the validation of the chain when accessing the system.

Client side

As anticipated, on the client side the blockchain is stored locally, the reason why it has been decided to do so, is that it is assumed that the blocks saved on the file sysyem are secure and can not be changed.
Although, in case a user wanted, he will be able to delete the current file system and make a new refresh anytime.
The organization of the file system is personal for every user that is using Permus, the public blocks will in fact be the same for all users, while the private blocks that are stored will be only those that include the subject in the transactions.

Certificate Authority - PKI

The certification authority has the task of issuing certificates to persons interested in having a personal certificate, so that they can access the service. As already mentioned above, the minimum requirements to be specified are name, surname, email and tax code.
The issuing service could be either a website that the service trusts, or it could be seen as a certificate issuing centre where the subject will have to present himself physically in order to have the issue.

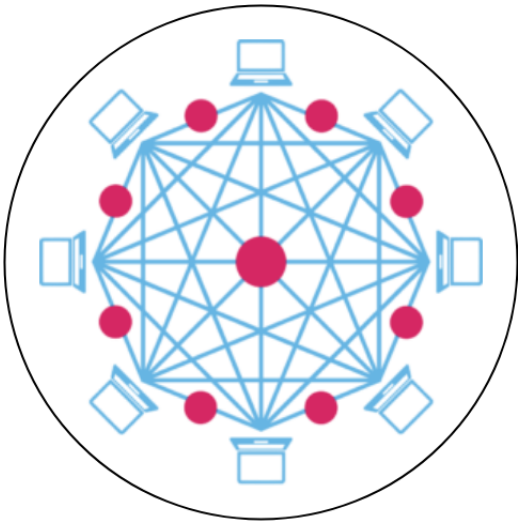

Figure 3.4. Permus network: all the nodes can communicate because of the central source, the Blockmaster.

3.1.2 Application tier: Process view

From its definition, the process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. [1] In the given models, it addresses the concerns of the developer on how the communication between the parties has to be implemented and gives an overview on how the protocol is. Every action inside the model is considered "atomic", and only after the execution of an action the process moves to the following. In this kind of model, it is assumed that all the actions have a positive outcome, leading the process to an end.
Thus, the Application tier is described using a model template that describes the interaction and the actions that a user and the blockmaster must perform so as to be able to successfully complete the basic operations of the system.

Process 1: Subject identity

The first model taken into account explains how the system deals the process of creation of a new subject: the parties involved are a subject and the blockmaster.
The first action of getting a valid certificate issued, has nothing to do with Permus itself. This means that the subject must be in possess of a valid certificate from a trusted authority and that it is not Permus' concern to issue one.
Once the blockmaster receives the request from a potential new subject, there are some constraints about the certificate that have to be respected, such as the CA that issued the certificate, the expiration date of the certificate and finally some fields such as name, surname email and user ID number.
Since all the subjects in the system need to know all the users, the blockmaster will create a "subject identity" transaction so that everyone will be updated and the new subject is finally able to access the system.

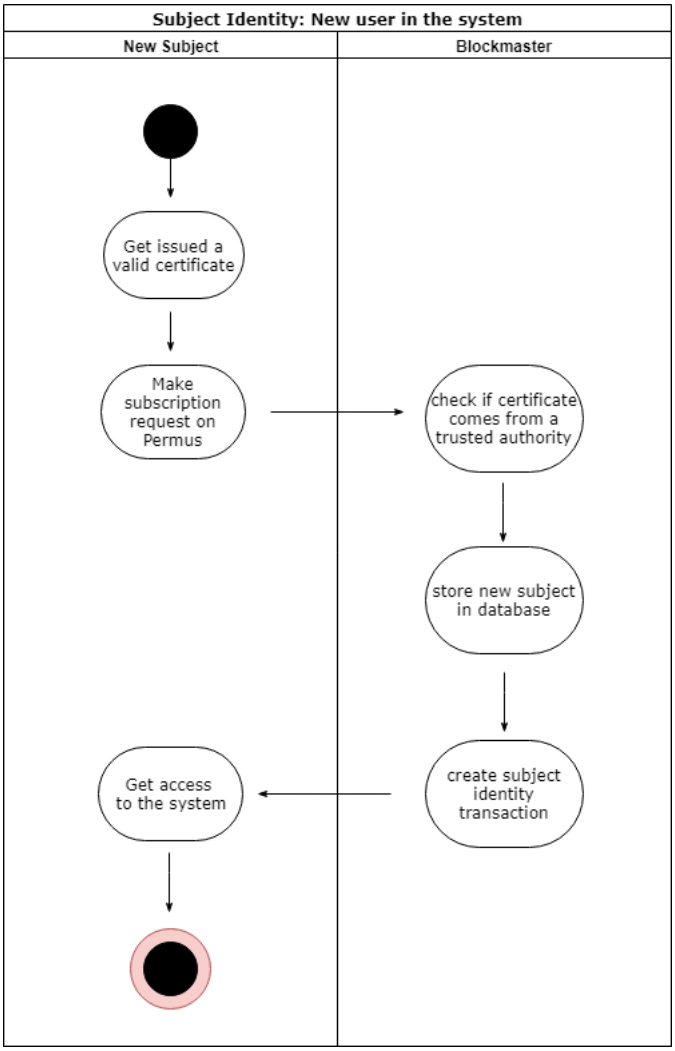

Figure 3.5. Subject identity process.

Process 2: Blockchain synchronization

While the client application is running, it is constantly updating the blockchain in background in case when new transactions have been added into the system. This means that the application has a first update when it starts running and then repeatedly while it is still on.
The way the client asks to the blockmaster an update, is to first get a "system info" update request. The response from the server contains specific information based on the subject who is asking, considering that the payload of the message will also contain the amount of coins a user is currently holding and other personal information.
When the response comes back, the client application will check that the latest serial of the block and of the transaction correspond to the ones of the local file system. In case the two values don't match, the client will forward an additional request "get block" for all the missing blocks in order to store them locally. The last thing

to do, once all the new blocks are received, is to validate the blocks by checking the hashes of blocks and transactions. If the validation is positive, the blocks will be added to the local file system and will always be considered reliable.

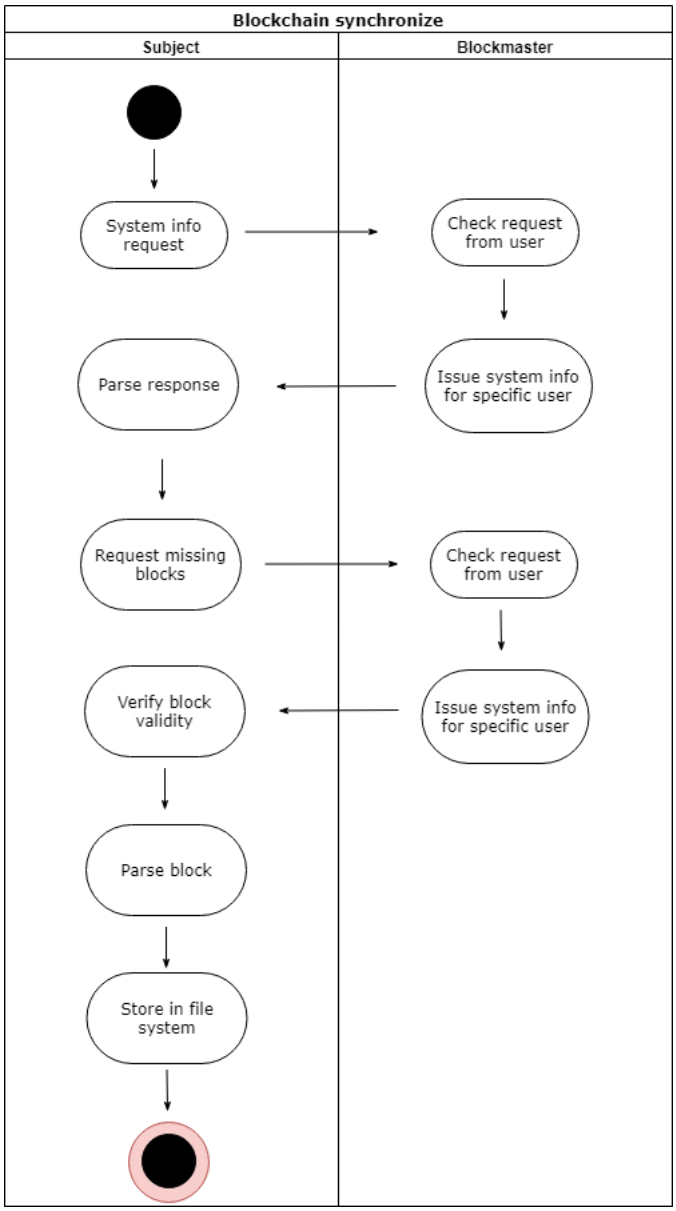

Figure 3.6. Blockchain synchronization process.

Transfer Initiation

Transfers are the main reason why Permus system has been implemented, thus it is clear that the communication process between parties is longer and gets more into detail.
All the transfers have the same initiation process and depending its type of transaction it will follow a different path in a second moment.
A transfer represents the will of a sender to give or exchange something, this means that the first action will start from him.
The initiation process begins with the sender creating a transfer transaction package (TTP), containing all the information concerning the specific transfer.
The TTP will be sent to the blockmaster in order to receive as a response a transfer id. This transfer id is needed for indexing it inside the pending transactions but also to prevent the re-use of the transaction by a malicious user.
The blockmaster will verify the validity of the transfer, generate the transfer id and send it back as a response to the initial user.
After this initiation, there is a split between signed and cosigned transactions.

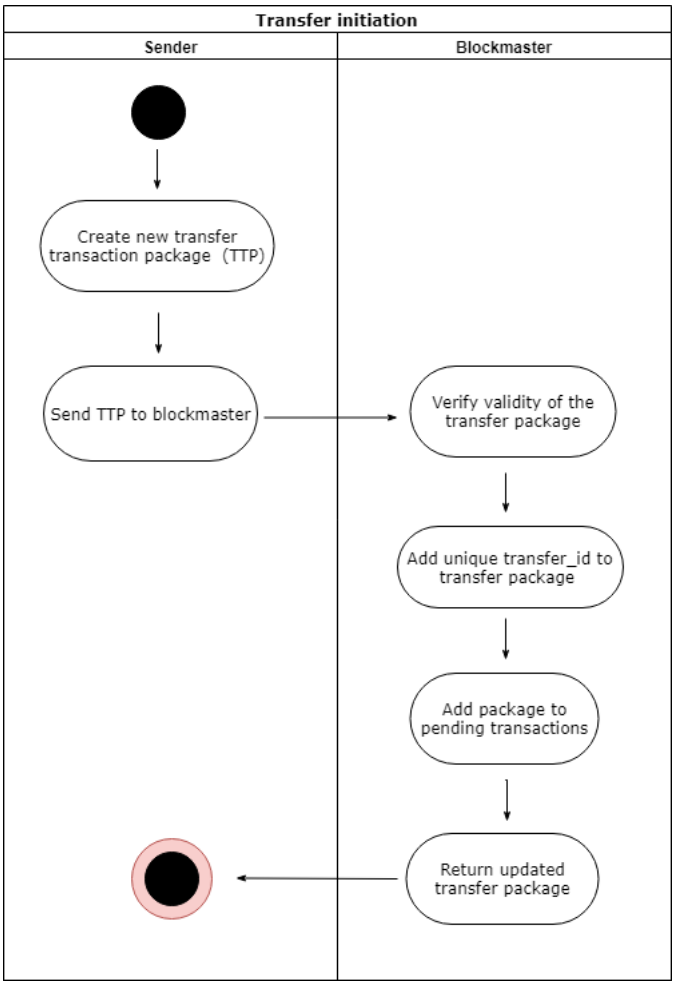

Figure 3.7. Transfer initiation process.

Signed Transactions

In signed transactions, the first part of the process consists of a transfer initiation just like shown in Fig [4.4].
Once the transfer id is received, the next step is to sign the TTP and send it again to the blockmaster.
The blockmaster will check again the transaction and, if everything is correct, the transaction will be be considered successful and added to blockchain.

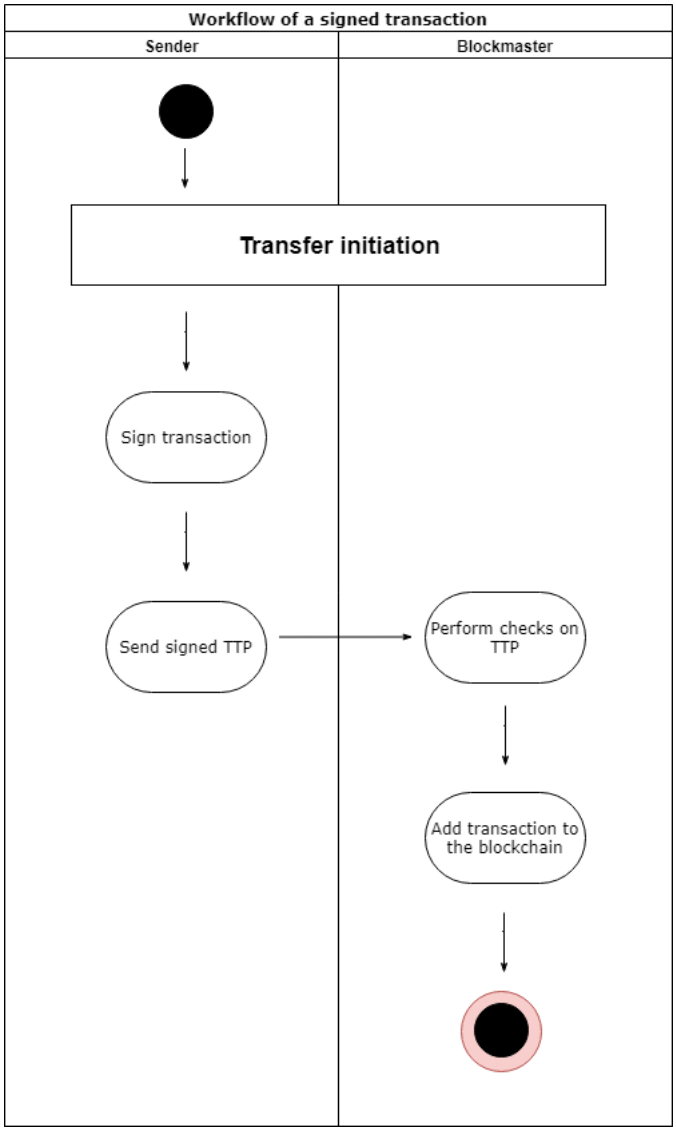

Figure 3.8. Signed transaction process.

Cosigned Transactions

Differently from signed transactions, cosigned transactions need also the signature from the receiver.
This means that after the checks computed by the blockmaster the second time, the transaction will be added into a pending transaction list.
From the receiver side, when asking for the block synchronization, he wil notice he received a transfer request. This transfer request will be checked, verified and finally examined. In case the receiver accepts the transfer, he will only have to sign it and send the TTP to the blockmaster.
Once again the blockmaster will perform all the checks and finally add the transaction in the blockchain.

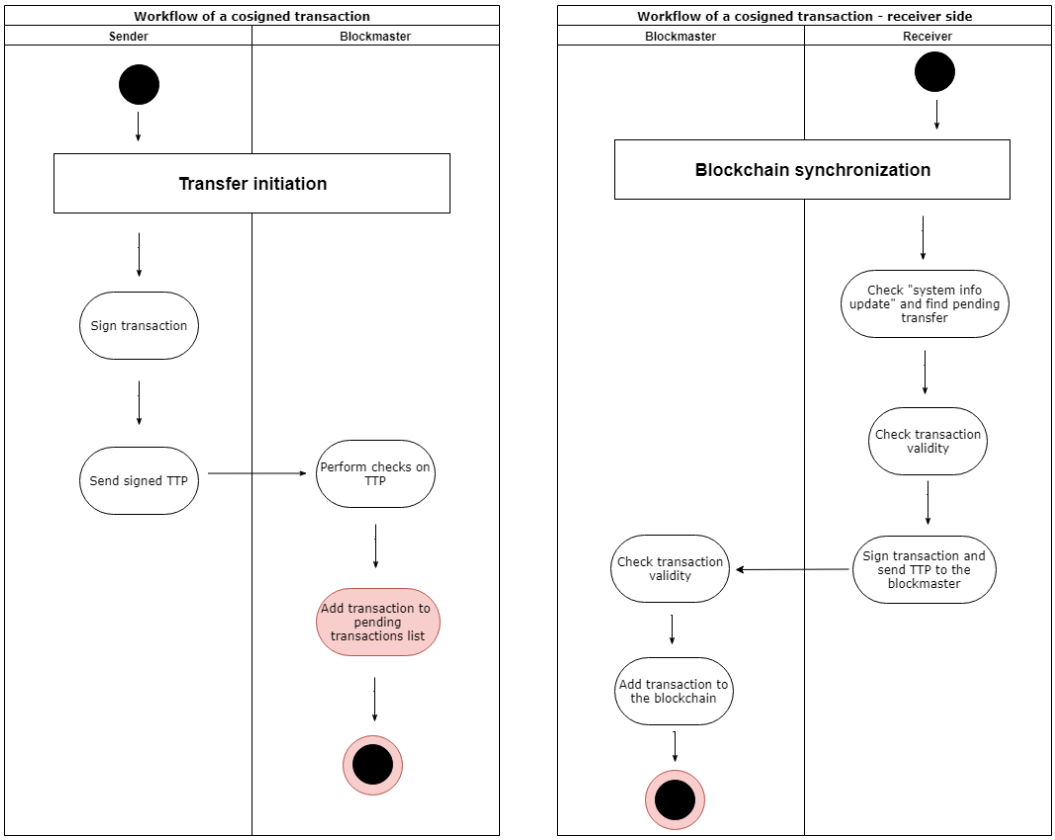

Figure 3.9. Process of a cosigned transaction with all the parties involved

3.1.3 Presentation tier: Operation list

For the presentation tier, it was decided to list all the operations that need to be implemented for a first prototype of the application.
It gives an overview of all the main actions users can perform inside the system from the client side and also gives idea to the developer what is still missing in the system in order to work properly.
In addition, the main operations of the blockmaster also have been added.

- User signs in with Permus for the first time
- User requests to the blockmaster an updated version of the blockchain
- User checks old transactions he has had with a certain person
- User checks the public blocks
- User checks validity of the updated blocks and stores them in the file system
- User signs a transaction he proposed
- User signs (accepts) a transfer he's been proposed
- Central institution creates additional IEU coins (coin creation)
- Subject transfers coin to another subject (coin transfer)
- Two subjects agree on a transfer and decide to do a public transfer transaction
- Two subjects agree on a transfer and decide to do a private transfer transaction
- Two subjects agree on a sale and decide to do a public sale transaction
- Two subjects agree on a sale and decide to do a private sale transaction
- Two subjects agree on a transfer and decide to do a private compensation transaction where the first subject pays off old transfers
- Blockmaster accepts subject's certificate and permits user to sign into Permus
- Blockmaster creates a subject identity transaction for the new user
- Blockmaster issues updated blocks belonging to the blockchain
- Blockmaster checks validity of a proposed transaction
- Blockmaster stores a new transaction in the blockchain

3.2 Why not using an existing Blockchain system?

There are many blockchain platforms out there today and most of them are free and open source. When I started thinking about Permus project, I first considered the possibility to simplify my work implementing the required ledger mechanisms just customizing those of a suitable open source blockchain platform.

I could have chosen BigChainDB [4], for example, since it is considered one of the best blockchain platforms for building open source distributed ledger systems. Better yet BigChainDB's main purpose is not only to store a large data amount, as it is also to enable developers to deploy blockchain applications as well as proof-of-concepts. Its database provides a high speed of transaction processing, a powerful query functionality, immutability, low latency, and decentralized control. For ensuring that all participants in a network agree on a single transaction log, BigChainDB uses the Federation Consensus Model that, for my project purpose is much more suitable than mechanisms based on proof-of-work and proof-of-stake used by leading edge world level cryptocurrencies.
I also considered using Multichain [10], an open source distributed ledger system designed for processing multi-currency financial transactions. Based on Bitcoin, this platform allows the developer to enjoy a fast solution deployment and enables a wide range of permissions and levels of access control. It employs a distributed consensus mechanism between identified block validators that is close in spirit to something like PBFT (Practical Byzantine Fault Tolerance), but instead of multiple validators per block, there is one validator per block.
Corda [9] was another one of the most popular open source blockchains that I could have used. Some of best things about Corda is that it has a pluggable consensus system and it allows developing blockchains where the transactions are strictly private.
Of course, I evaluated the possibility to use Openchain [12] since it is considered by many the best blockchain platform that is open source. Openchain was specifically designed to issue and management of digital assets in a safe, scalable, and robust way. It includes the Partionned Consensus as its consensus mechanism and every transaction is digitally signed just like it happens with Bitcoin.
With Partionned Consensus every Openchain instance only has one authority validating transactions and, instead of one single central ledger, each organization controls their own Openchain instance. Instances can connect to each other.

The final choice, of course, was building **"my own"** blockckain system. Awkward as it may seem, I came to this decision because the objective of the work was purely academic, as at the end of my formal studies I wanted to challenge myself building something innovative and at the same time potentially useful.
Moreover, I also wanted to use in the project the full stack of the technical knowledge I was able to acquire in my academic career, during which, especially in the last year, I have had the opportunity to follow several courses on cyber security, which I found fascinating, especially the one on applied cryptography. In this way got very close to the blockchain concepts and all the cryptographic operations needed to build fast, safe and reliable blockchain platforms.

Also, none of the available open source blockchain platforms had exactly the characteristics I had in mind for Permus. For example, it seemed to me that none was based on PKIs or X509 digital certificates, none could accommodate public transactions along with private ones.
I have therefore decided I decided to work on a very "personal" prototype of permissioned blockchain to explore the potentiality of this kind of platforms in providing services to small communities.

After carefully reading the documentation of some mainstream opens source blockchain systems, it was clear that they way offered too many unnecessary features for the system I had in mind, at the cost of not giving the programmer the ability to customize other parts.
The thing that was most important to me was to implement different types of transactions that somehow were linked together.
Finally, as I have already hinted, having worked on several personal projects both on my own and in a team over the last few years, I really liked the idea of creating a blockchain related system as a personal challenge.

Chapter 4

Background

A system expected to offer the above mentioned features, should be designed around commonly accepted software standards and technologies. For this reason it is important to first identify the standards that best fit the development needs. It is important, in this phase, to focus open or widely accepted and commonly used standards, in order to avoid the possibility to incur in post-development validation issues.

4.1 Blockchain

Usually, when talking about the Blockchain, it is straightforward to talk about cryptocurrencies as it is the main reason of the rise of this technology. The structure is a decentralized, distributed and public digital ledger that is used to record transactions across many computers so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks through hashing the content of the block when it has become full. The database is managed autonomously using a peer-to-peer network and a distributed timestamping server. Blockchain technology can be integrated into multiple areas. The primary use of blockchains today is as a distributed ledger for cryptocurrencies, most notably bitcoin. There are a few operational products maturing from proof of concept, such as Permus. [16]

4.1.1 Blocks

Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the cryptographic hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the original genesis block.
Blockchains are typically built to add the score of new blocks onto old blocks and are given incentives to extend with new blocks rather than overwrite old blocks.
Therefore, the probability of an entry becoming superseded decreases exponentially as more blocks are built on top of it, eventually becoming very low. For example, bitcoin uses a proof-of-work system, where the chain with the most cumulative proof-of-work is considered the valid one by the network. There are a number of methods that can be used to demonstrate a sufficient level of computation. Within a

blockchain the computation is carried out redundantly rather than in the traditional segregated and parallel manner. [19]

A block can be thought of as the container for data. Each block contains data, Block Header, Block Identifies, and Merkle Trees.

- Block headers contain metadata about that particular block, such as the cryptographic hash from the block chronologically before it and a data structure to summarize the transactions in the block – also called the Merkle Tree Root.
- Block identifiers are essentially the cryptographic hash to uniquely identify the particular block.
- Merkle Trees[20] refer to the structure of transactions in the block.

[8]

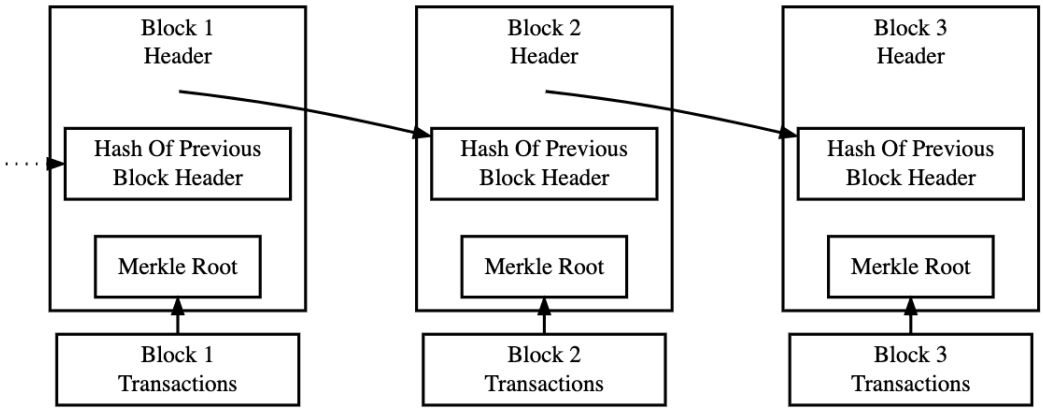

Figure 4.1. Simple example of a block structure in a blockchain. [5]

4.1.2 Transactions

The blockchain is a distributed ledger of transactions across a peer-to-peer network. It's basically a database of transactions. The transaction itself can involve cryptocurrency, contracts, records, or other information. So, a transaction is simply a record of something, normally it would be a financial transaction in the cryptocurrency context. Let's go through how it works to understand more. Imagine that someone has requested to make a transaction, for example, a payment to another person. This request will be broadcast to a network of computers that are going to verify the transaction and the user's status. The verified transaction then will be combined with other transactions to make a new block of data in the existing blockchain. From this mechanism, you can see there are many advantages of the blockchain system. The ledger or database is distributed, which means the information is never controlled by one central source and anyone can check for recorded transactions in it. After transactions are verified and recorded, they become nearly impossible to change. The hacker that try altering the record would be up against miners supercomputers.[22]

Role of hashing

In a complicated setting such as the blockchain, in order to achieve trust between different parties and to keep the transactions and block safe from modifications, hashing plays a very big role. In general, without hashing, it would be impossible to get these fundamental features. Its one-way strength is crucial for the system as for nowadays technology it is impossible to rollback to the plaintext from a given hashed string. For this reason all the blocks and all the transaction make a very big use of hashing, so that if an attacker or a malicious user tries to alter the information the chain will be invalidated.

4.1.3 Blockchain Types

A single, universal blockchain network cannot possibly serve all industries, granted the vastly different needs of businesses, and individual users. This has led to the creation of numerous blockchain networks, each with a slightly different set of protocols, while the back pillars remain the same. Despite the vast number of blockchain networks available at this moment in time, the market has two types of blockchains: permisionless (public) and permissioned (private). [18]

Permisionless Blockchains

Permisionless blockchain networks power up most of the market's digital currencies. They allow every user to create a personal address and begin interacting with the network, by submitting transactions, and hence adding entries to the ledger. Additionally, all parties have the choice of running a node on the system, or employing the mining protocols to help verify transactions. Apart from allowing anyone to get involved on the network, there are few more characteristics associated with the permisionless model. These are: Decentralization: permisionless networks need to be decentralized, which means that no central entity has the authority to edit the ledger, shut down the network, or change its protocols. Many permisionless networks are based on consensus protocols, which means that network changes of any type can be achieved as long as 50% + 1 of the users agree to it. Digital assets: Another characteristic is the presence of a financial system on the network. Most permisionless networks have some kind of user-incentivising token, which can grow or fall in value depending on the relevancy and state of the blockchain they belong to. Currently, permisionless blockchains employ either monetary or utility tokens, depending on the purpose they serve. Anonymity: granted the way blockchains operate, anonymity has become quite relevant in the industry. Many permisionless networks do not require users to submit personal information prior to being able to create an address, or submit transactions. However, in certain cases, personal information is required for legal purposes. Bitcoin, for instance, does not offer full anonymity, as user identity is indirectly tied to the addresses they have the private keys of. Transparency: blockchain networks are bound to be transparent by design. This is a required characteristic, given the fact that users who get involved must be incentivised to trust the network. Therefore, a transparent network needs to freely give users access to all information apart from the private keys – from addresses, to how transactions are processed into blocks, and the freedom to see all transactions processed by the network.

Permissioned Blockchains

Permissioned blockchains act as closed ecosystems, where users are not freely able to join the network, see the recorded history, or issue transactions of their own. Permissioned blockchains are preferred by centralized organizations, which leverage the power of the network for their own, internal business operations. Company consortiums are also likely to employ private blockchains to securely record transactions, and exchange information between one another. Transparency Anonymity: private blockchains are not required to be transparent, but they can choose to do so

freely, depending on the inner organization of the businesses. In terms of privacy, it isn't needed on a central level, and can be individually determined on a user-case basis. Many private blockchains store an extensive amount of data relating to the transactions, and operations carried out by users. Lastly, as there is no internal economy for most private blockchains, there is no need to see how monetary tokens are being sent or used.

4.2 Crypthography

In order to achieve a reasonable level of security, commonly accepted cryptography standards have to be used. Throughout the proposed system three cryptography functions are used: hashing, encryption and digital signature.

4.2.1 Encryption

Encryption is the process of encoding a message or information in such a way that only authorized parties can access it and those who are not authorized cannot. Encryption does not itself prevent interference, but denies the intelligible content to a would-be interceptor. In an encryption scheme, the intended information or message, referred to as plaintext, is encrypted using an encryption algorithm – a cipher – generating ciphertext that can be read only if decrypted.

4.2.2 Hashing

Hashing is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash) and is designed to be a one-way function, that is, a function which is infeasible to invert. The only way to recreate the input data from an ideal cryptographic hash function output is to attempt a brute-force search of possible inputs to see if they produce a match.[wiki]

4.2.3 Digital signature

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity).

4.3 PKI - Identity Management

Any subject expected to use the proposed system needs to be positively identified for obvious reasons. System users have to be sure that they are dealing with counterparts that represent “real” people or “real” entities whose existence and identity is publicly recognised in the “real life” outside cyberspace. Although there are nowadays other numerous and widespread methods of online authentication, such as oauth2 commonly used in virtually all mainstream social media, the identity management method chosen for the proposed system was a Public Key Infrastructure (PKI), for reasons detailed below.

4.3.1 Public Key Infrastructure

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store revoke digital public key certificates and manage public-key encryption. The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as e-commerce, internet banking and confidential email. It is required for activities where simple passwords are an inadequate authentication method and more rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred. In cryptography, a PKI is an arrangement that binds public keys with respective identities of entities (like people and organizations). The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA). Depending on the assurance level of the binding, this may be carried out by an automated process or under human supervision.

4.3.2 Digital Public Key Certificates

In cryptography, a public key certificate, also known as a digital certificate or identity certificate, is an electronic document used to prove the ownership of a public key. The certificate includes information about the key, information about the identity of its owner (called the subject), and the digital signature of an entity that has verified the certificate’s contents (called the issuer). If the signature is valid, and the software examining the certificate trusts the issuer, then it can use that key to communicate securely with the certificate’s subject. The most common format for public key certificates is defined by standard X.509. The digital certificate can also contain the private key enabling the owner to perform digital signatures and encryption operations.

4.3.3 Certification Authority

In a typical public-key infrastructure (PKI) scheme, the digital certificate is issued by a Certificate Authority (CA). A digital certificate certifies the ownership of a public key by the named subject of the certificate a CA acts as a trusted third party trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. A common use of a CA is in issuing identity cards by national governments for use in electronically signing documents.

4.3.4 Advantages of PKI and digital certificates

As mentioned above, a public key certificate issued by a trusted Certification Authority is a simple and effective way to positively identify subjects online. The identification is achieved by requiring the remote user to digitally sign their messages using their private key. If the digital signature is verified server side, assuming the remote subject is the only one in possession of the private key, his identity is guaranteed. At the same time, public and private key contained in the certificate can be used to perform asymmetric encryption functions.

Chapter 5

Permus system

Permus is an online bartering service that makes use of a **permissioned Blockchain**, where every user needs to be authenticated to the Blockmaster with a personal Certificate that has been previously issued by a Certification Authority.
One of the roles of the Blockmaster in this environment, is to store and provide all the existing blocks to the user during his access, so to make sure that everything is up to date and that it is secure to begin new transactions. Another necessary thing that the Blockmaster needs to do is be an intermediate between Sender and a Receiver when a transaction is proposed.

Figure 5.1. Permus logo.

5.1 Blocks in Permus

Transaction represent in Permus, as in all blockchain system, a smart contract between subjects that decide to agree on an exchange. Groups of transactions are put together in blocks one after another, which is cryptographically hashed and linked to the previous block for the validation of the chain. In Permus we find two types of block: public and private. The idea of having private blocks is to allow all the users to achieve privacy for certain types of transaction where the exchanged things are kept secret. [17]

5.1.1 Public Block

A public block is a container of transactions that all the users of the system are allowed to see and read in clear. It can contain up to 10 transactions before a new block is created.

Every public block we can find in Permus is described by the following attributes:

Version Serial: serial number inside the chain that identifies the block Timestamp: timestamp of when the block has been created Previous version: hashed fingerprint of the current transactions Transfer root: continuously updated hash using a merkle tree Transactions: list of the transactions that are in the block

5.1.2 Private block

The concept of private block in Permus is very simple, for each Private transaction there will be an additional block where its encryption will allow only the two involved subjects to read its content. For example, in a Private transfer the description of the exchanged good is not in the public chain, but in an extra block. The only reference the two users have for the private block in the public chain is the hash of the private block that is stored in the public part of the transaction. In order to ensure the privacy of the block, it is encrpyed and enveloped for the certificates of the subjects involved.

5.1.3 XML Block Structure

Every block belonging to Permus has the following XML structure:

```
<?xml version="1.0" encoding="utf-8"?>

<Block>
  <version>1.0</version>

  <serial>1</serial>

  <timestamp>62135596800</timestamp>

  <prev_vers>fCv3O5G4mfDXbKaUNFm=</prev_vers>

  <trans_root>nzXmFyjikt3dhDTyBru=</trans_root>

  <transactions count="10">

    ....

  </transactions>

</Block>
```

5.1.4 Hashing a block

In order to hash a block, the hash base needed from the system to output a valid hash string comes in the following order:

- version
- serial number of the block
- timestamp
- hash of the previous block in in Base16
- transaction root in Base16

```
Public ReadOnly Property hashBase() As String
    Get
        ' prima ricalcoliamo l'hash delle transazioni contenute in questo blocco
        trans_root = transactions.computeHash()
        Dim t As New StringBuilder(200)
        t.Append(version)
        t.Append(serial.ToString)
        t.Append(utility.getUnixTime(timestamp).ToString)
        t.Append(utility.bin2hex(prev_vers))
        t.Append(utility.bin2hex(trans_root))
        Return t.ToString
    End Get
End Property
```

Figure 5.2. A boat.

5.1.5 Transaction root

Transaction root is very important to the system: it is not only needed for the validation of a previous block when used as part of the hashbase of a block, if a block is still incomplete (meaning it is containing less than 10 transactions) it is still possible to verify its consistency. The way the transaction root is built is recursive, using a left-deep Merkle tree[20] where the initial root (initial seed) is given from H(000...000).
The idea is to update the transaction root for each new transaction that comes inside the block just like in the following graphical example:

The initial seed is the same for every empty block, once the first transaction is registered into the block the root will be updated in the following way:

$$TR_1 = H(TR_0||H_1) \tag{5.1}$$

Where H_1 is the hash of the new transaction. When a new transaction is saved inside Permus, the transaction root will be updated using the same pattern.

It is straightforward to see that for the **validation of a block** by checking the transaction root, the user has to start from the deepest leaves and reach the root just like when the root needs to be updated.

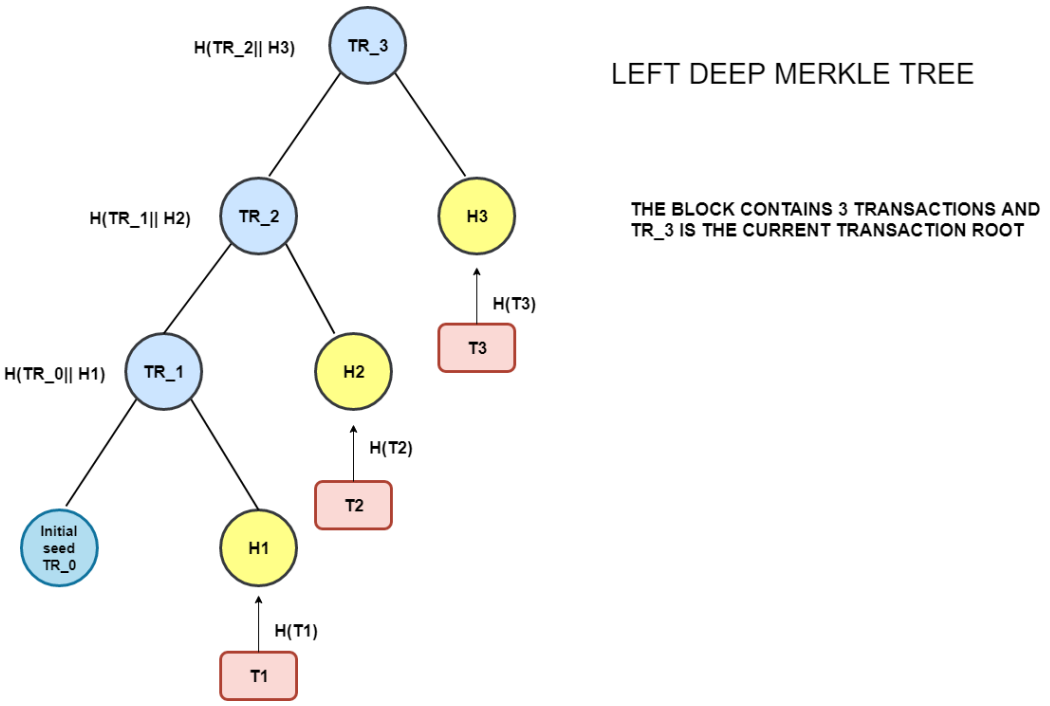

Figure 5.3. For the transaction root, the system makes use of a left-deep merkle tree.

5.2 Transactions

Every block in Permus contains a fixed number of transactions that are registered and saved by the Blockmaster. It is possible to identify various types of transactions that are necessary for the system to work properly. We can find different categories of transactions that can be seen as inheritance tree:

1. Subject identity
2. Transfer
   (a) Signed
   (b) Co-signed
   (c) Public
   (d) Private

All the different types of transaction share some attributes that are inherited from the root of the tree:

transaction type: identifies what transaction it is blockSerial: identifies the block the transaction belongs to serial: identifies the position inside the block timestamp: identifies the timestamp the transaction has been executed

There are transactions that can only be executed by the blockmaster, such as the Subject identity and the coin creation. A normal user is able to start all the other types of transactions. This is due to the fact that only the central authority (in this case the blockmaster) must be able to create new subjects inside Permus and to create new coins.

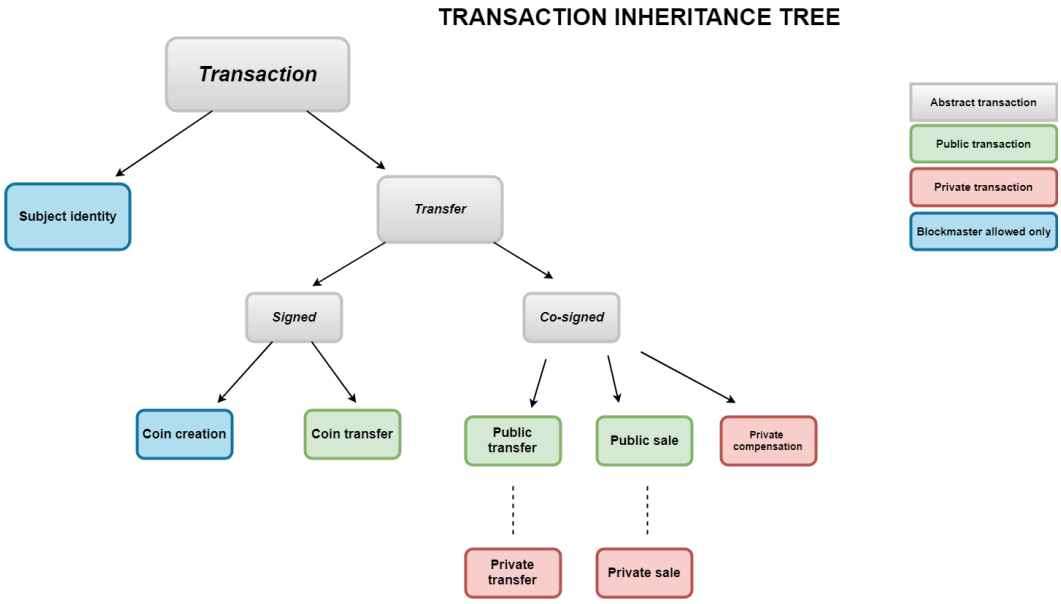

Figure 5.4. Transaction inheritance tree.

5.2.1 Subject identity

This type of transaction is fundamental for the system. When a new user is added inside of the system, in addition to the basic attributes, it will contain subject information such as name, email, social security number, and public certificate (issued by a certain Authority). The certificate the new subject wants to use for Permus has to come from an Authority that is trusted according to the blockmaster. When a subject joins Permus, he knows that some of these information can be seen by other users of the system.

5.2.2 Transfer

Transfer represents the generic category, and the additional attributes each transfer will have are the sender, receiver, a message and finally the signature from the sender. Transfer is further divided into two different categories, Signed and Co-signed. This is because we can identify transfers where the receiver is not supposed to accept or deny the transaction, and others where his signature is needed to accept the exchange.

Before getting deeper into the types of transfers that Permus supports, a distinction between public transaction and private transaction needs to be done. A transaction is public when the all the content of the transaction is in the relating block, a transaction is private when some piece of information is hidden and stored in a private block[riferimento a private block nella sezione blocchi].

Signed transactions

Signed transactions are public transactions, and only the signature of the sender is needed since, just like with bitcoins and every other cryptocurrency, it is an

advantage from the receiver to get some extra coins. Coin creation and Coin transfer are the two transactions belonging to this category.

Coin creation

Only the blockmaster is allowed to create new coins inside its environment. An inner coin value is needed since some kind of exchanges are not necessarily done with objects and goods, thus bartering is not the only type of exchange accepted in Permus. The more coins will be created inside the system, the less valuable a single coin will be.

Differently from other known systems that make use of a virtual value over the blockchain (i.e. bitcoin or ethereum), in Permus the creation of IEU is not a consequence of "mining", an efficient method to limit and control the amount of circulating currency but which generates inside the system an unfair distribution of "wealth", preserving during the initial phase those subjects who are able to use large computation quantities for the extraction of the currency unit.

In the system, there exist a unique subject who is conventionally authorized to create IEU (which implies the possibility to transfer to other subjects IEU). This unique subject represents the figure of the community and its activities in the system should be the consequence of board decisions.

Coin transfer

A coin transfer from a sender to a receiver, as previously mentioned, does not need the receiver to accept the amount. The blockmaster will check that the sender is able to send the amount and will confirm the transaction.

Co-signed transfers

Differently from signed transactions, co-signed transactions need the signature of both parties involved during the transfer. The protocol of communication between the two subjects involved is very simple, the sender will forward the signed proposal to the receiver, the transaction can be considered successfully ended only in case of signature of the receiver. Otherwise, the transaction will be dismissed.

Public Transfer

Public transfer is one of the first transactions that came up when developing the project, it consists in a simple proposal that the sender proposes to the receiver. The proposed object is called "transfer object" and it contains an id of the object (e.g. a product from a market), its quantity and a brief description.

Public Sale

A public sale can be seen as the union of a coin transfer and a public transfer, where the sender is giving an object/good in exchange for a certain amount of coins.

Private transfer

A private transfer can be seen as a public transfer where the object exchanged is kept private between the two parties, who will be the only ones able to see the content. This private information is kept in a private "shared" block between sender and receiver. Just like other simple users of Permus, the blockmaster will not be able to see the object that has been exchanged.

Private sale

A private sale can be seen as public sale, with the difference that the object description is hidden to the public chain for privacy. The object description will contain the referred hash. Although the content of the sale is kept private, just like in a private transfer, it is important to leave the exchanged coin amount in clear, so that the blockmaster is able to validate the transaction.

Private compensation

Private compensation is the core transaction present in Permus and the reason why the whole system has been developed. As the name reminds, with this type of transfer, the sender is able to pay back old transactions he had with the receiver. The XML description, in terms of attributes, looks just like a private transfer. The difference between the two transactions is hidden in the private block where, in case of agreement, a private compensation will contain the new object that has been exchanged with a list containing the past exchanged transactions (either public or private ones).
Private compensation do not necessarily have to completely reward an old transaction, as users may decide to agree on a consistent percentage to cover the exchange.

As you can see in the picture, a list of transactions: some that still need to be rewarded and others that have been 100% compensated.

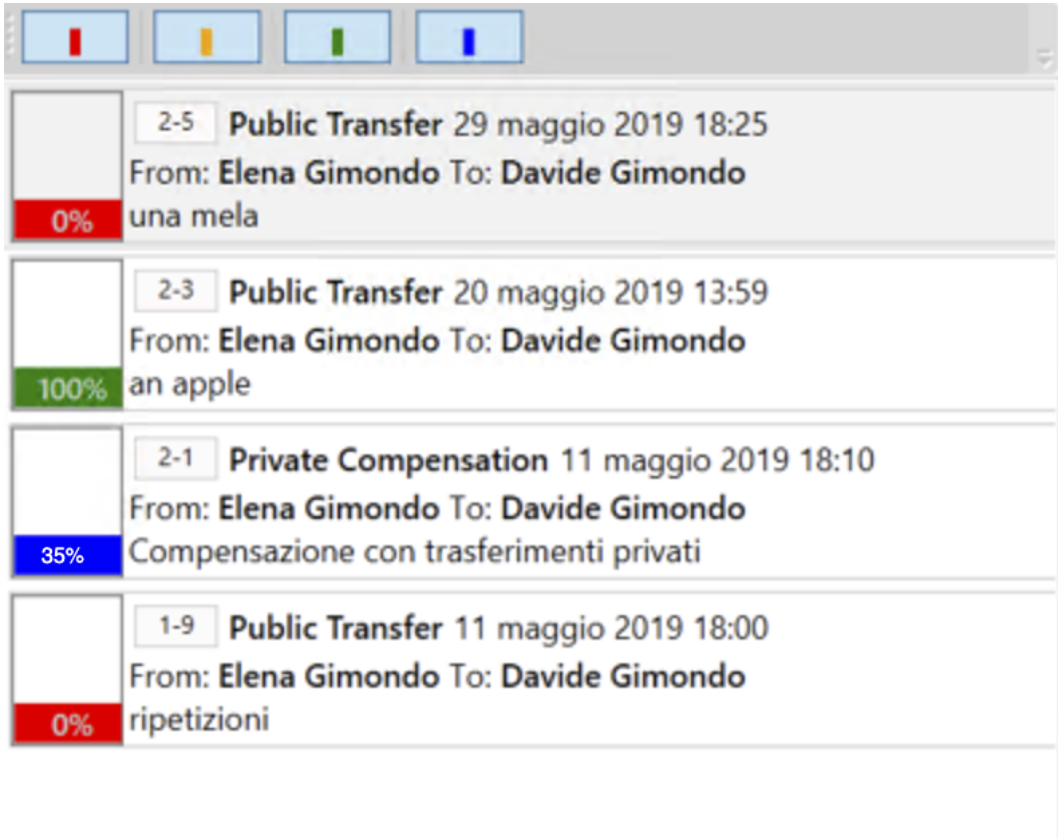

Figure 5.5. List of transaction between two users with the relative compensation percentage.

5.2.3 Hash base of a Transaction

In order to hash a transaction, the hash base needed from the system to output a valid hash string comes in the following order:

- transaction type
- serial number of the transaction
- serial number of the belonging block
- timestamp

5.2.4 Transaction Consent Diagram

The Consent Diagram of a transaction represents broadly and abstractly the order of the actions that take place within Permus for the creation and confirmation of a generic transaction.
First of all, there is a verbal agreement between the two parties, then the rest is executed within the system with the client interface made available to users.

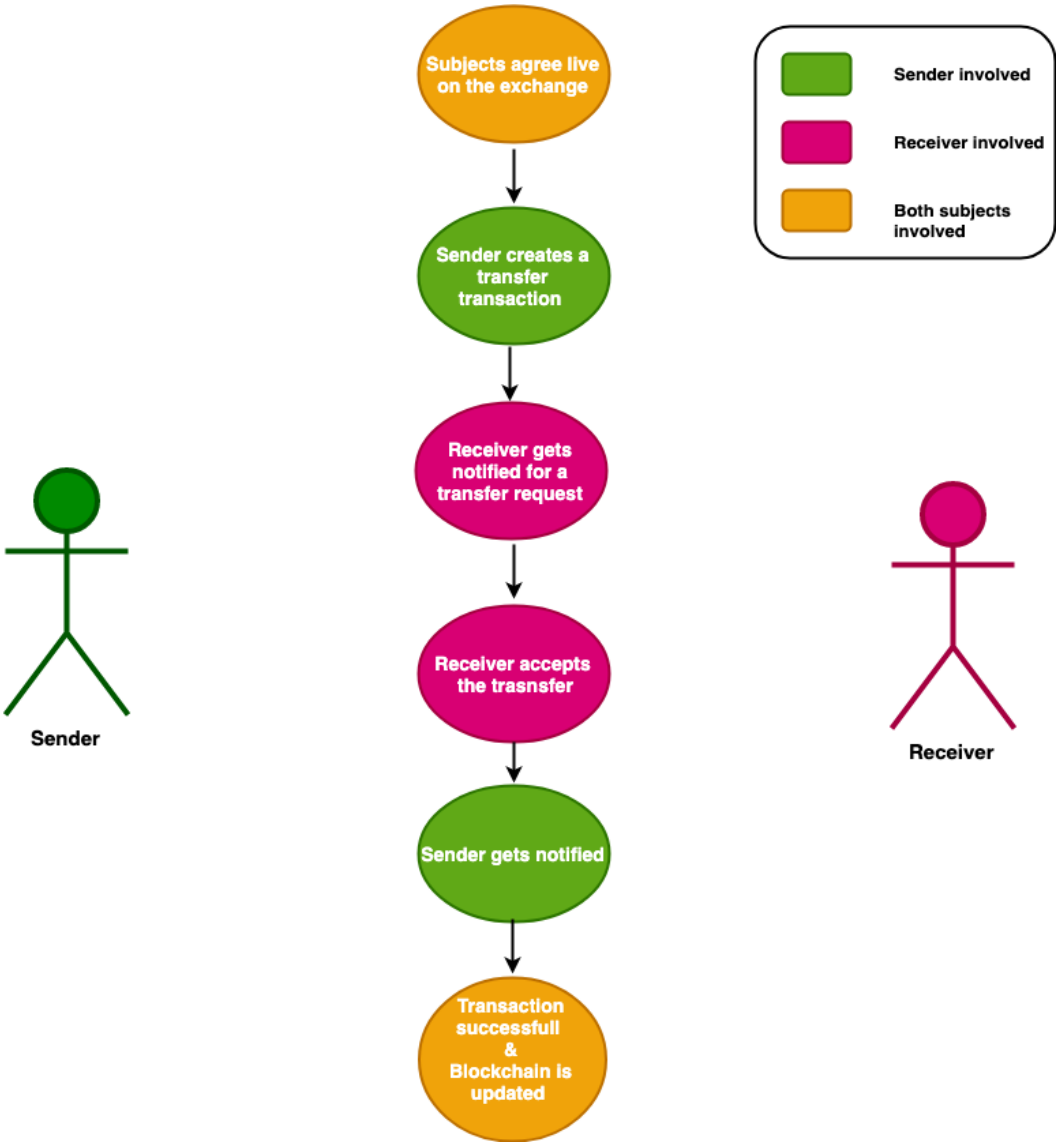

Figure 5.6. Transaction Consent Diagram

5.3 Blockmaster

The blockmaster represents in Permus the central source of the system and manages different tasks which are necessary in order to get a correct workflow of the application. The following tasks are described in detail.

5.3.1 Roles

Create new users

When a new user decides to join Permus, he has to submit his credentials with a certificate that is trusted by the blockmaster. If the certificate is valid, a new transaction of the type "Subject identity" will be created by the blockmaster and the new subject will be able to make use of all the features allowed in Permus.

Block storage

All the blocks are stored by the blockmaster so that when a user needs them he will be able to be updated. The public blocks are stored in clear, the private ones are encrypted with the two involved subject's certificates. It's clear that an outsider is not able or allowed to see the content of these blocks.

Block issuer

Since all the information about the blocks and transactions are stored by the the central authority, when a user logs in into the system the blockmaster will provide an update by sending the missing blocks/transactions from the user's last login. At the same time, when a new transaction is successfully computed by two subjects, they both will get an updated version of the current block where their transaction has been insert.

Message broker between subjects

For co-signed transactions, the blockmaster has a role as an intermediate between the two parties. The sender will let know the the blockmaster that he has a transfer offer for a certain receiver, the blockmaster will thus forward the proposal to the receiver and wait for his response.

Chapter 6

Implementation

6.1 Development

6.1.1 .NET framework

There are numerous important choices to be made during the planning phase of software development projects. Among them, one of the most critical is choosing the correct framework.
In essence, a software framework is a universal and reusable environment in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework provides a standard way to build and deploy applications. Software frameworks usually include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or system.
the criteria to consider when choosing the right framework for a given software project usually depend on the nature of the project itself. Licenses, for example are important because can have a significant impact on the distribution of the applications. For example, an application developed using a GPL-licensed framework will necessarily be subject to GPL. On the other hand, this is not the case for an MIT-licensed framework. Security is also very important as any application is potentially vulnerable. To minimize risk it is always better to select a framework capable of ensuring security functions. Long living projects are better suited by well-known and recognized the frameworks that are expected to last for decades. Widely used frameworks also have excellent documentation and support, available from the publisher in the first place, but also from communities online. It is also easier to find resources on the market when looking for skills related with the most common and widespread software frameworks.
Permus project implementation is based on .NET (dot net) framework, a software framework developed by Microsoft that runs primarily on Microsoft Windows Operating Systems. It includes a massive class library named as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET

Framework execute in a software environment (in contrast to a hardware environment) named the Common Language Runtime (CLR). The CLR is an application virtual machine that provides services such as security, memory management, and exception handling. As such, computer code written using .NET Framework is called "managed code". FCL and CLR together constitute the .NET Framework.
FCL provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their source code written in one of the many supported languages, with the framework FCL and other libraries. The framework is intended to be used by most new applications created for the Windows platform.
Although the .NET Framework began its life as proprietary software, nowadays Microsoft has changed .NET development to more closely follow a contemporary model of a community-developed software project addressing the concerns on software patents expressed by the developers, especially those in the free and open-source software communities.

6.1.2 Visual Studio

Microsoft also produces a very powerful and easy to use integrated development environment (IDE), largely intended for .NET software, called Visual Studio.
Microsoft Visual Studio is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation and many others. It can produce both native code and managed code.
As most IDEs currently available, Visual Studio includes a powerful and versatile code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer.
Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C, JavaScript, TypeScript, XML, XSLT, HTML, and CSS.
The ability to write reliable code rapidly and predictably depends mainly from the programmers skills and experience, but a good IDE helps a lot, this is why Visual Studio was a obvious choice for developing Permus. Moreover the most basic edition of Visual Studio, the Community edition, is available free of charge, at the same time it sports all those nice features a honest programmer needs to be productive nowadays.

6.1.3 Github

Visual Studio accepts plug-ins that enhance the functionality at almost every level including adding support for source control systems like Git. Source code control

or version control (also known as distributed revision control) is a form of version control where the complete codebase - including its full history - is maintained on a repository.
Distributed version control systems (VCS) maintain a mirrored copy of the project's codebase on every developer's computer. This allows branching and merging to be managed automatically, increases speeds of most operations and improves the ability to work offline.
Git is a very popular distributed version-control system created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. GitHub is a web-based hosting service for version control using Git that is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. Although GitHub was recently acquired by Microsoft, it will continue to operate independently as a community, platform and business, offering plans for enterprise, team, pro and free accounts which are commonly used to host open-source software projects. As of January 2019, GitHub offers unlimited private repositories to all plans, including free accounts.
With more than 28 million users and 57 million repositories (as of june 2018), Github is by far largest host of source code in the world. It was unthinkable not to choose it to host Permus source code as it was important to track changes in source code during development. I also plan to leave Permus source code on GitHub in the future to be used by anyone would find it useful for future projects.

6.2 Permus Object Model

Modern object oriented Frameworks, such as .NET Framework offer the possibility to define objects or classes through which a program can examine and manipulate some specific parts of its world. In other words, the object-oriented interface to some service or system. Such an interface is said to be the object model of the represented service or system.
Objects built on the Framework programming language usually map entities defined in the system. For example in the Permus System, objects such as Block, Transaction, Blockchain are defined in order to properly handle each entity properties and behaviours.
Object Oriented Programming (OOP) paradigm implemented in virtually all modern Framework support and encourages the use of inheritance. This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships. For example in Permus, class “Public Transfer” is a specialized version of class “Transaction” and inherit from it. All the data and methods available to the parent class also appear in the child class with the same names. In this example, class Transaction defines the properties “timestamp” and “serial”. These will also be available in class “Public Transfer”, which might add the properties "transferId" and "fromSubject" and “message”.
Subclasses can override the methods defined by superclasses. For example in Permus, class Blockchain define method “obtainBlock” intended to gain access to a specific Block in the Blockchain. In class Blockchain this method returns nothing, because it is specifically intended to be overridden by derived classes such as class ClientBlockChain and class BlockMasterBlockChain. Each of them has a different strategy on how to obtain a block: ClientBlockChain looks for the required block in the file system first, if it cannot find it locally it requests it to the blockmaster service.
BlockMasterBlockChain on the other hand, must assume that either the required block is in the server database or the block doesn’t exists.
In order to minimize the amount of code required to implement Permus system, a fairly complex Object Model was designed.
In order to maximize the reusability of code throughout the applications of the Architecture, Permus Object Model was packaged in a specific software component called Permus.dll. It contains almost all classes needed for creating a Permus client, only user interface related elements are not provided for obvious reasons.
Also provided by Permus.dll is a wide set of general purpose utilities closely related with the functions required by Permus processes. Among them, the most useful is class BlockMasterWebApiClient. This class exposes to the developer all Web Api methods available on the BlockMaster as they were local methods. It is up to the BlockMasterWebApiClient to handle all details needed in order to properly call BlockMaster Web Services.
For maximum flexibility, all methods in this class are designed in a way that they can be called either synchronously and asynchronously. When the programming logic requires the Api method result to be available before it can continue execution, the synchronous calling mode needs to be used. In this case the Api result is the result of the class method. otherwise , specifically when the calling thread doesn’t

need the Api method result to continue, using the asynchronous calling method is more convenient performance wise, since the call does not affects the execution flow of the thread. When a BlockMasterWebApiClient class method is called in asynchronous mode, it returns a null value immediately in order to allow the calling thread to continue execution. When the the Web Api call result is available, the system automatically calls the dataReady method of the class implementing the iWebApiAsyncReceiver interface passed as a parameter to the function in order to trigger the asynchronous mode.

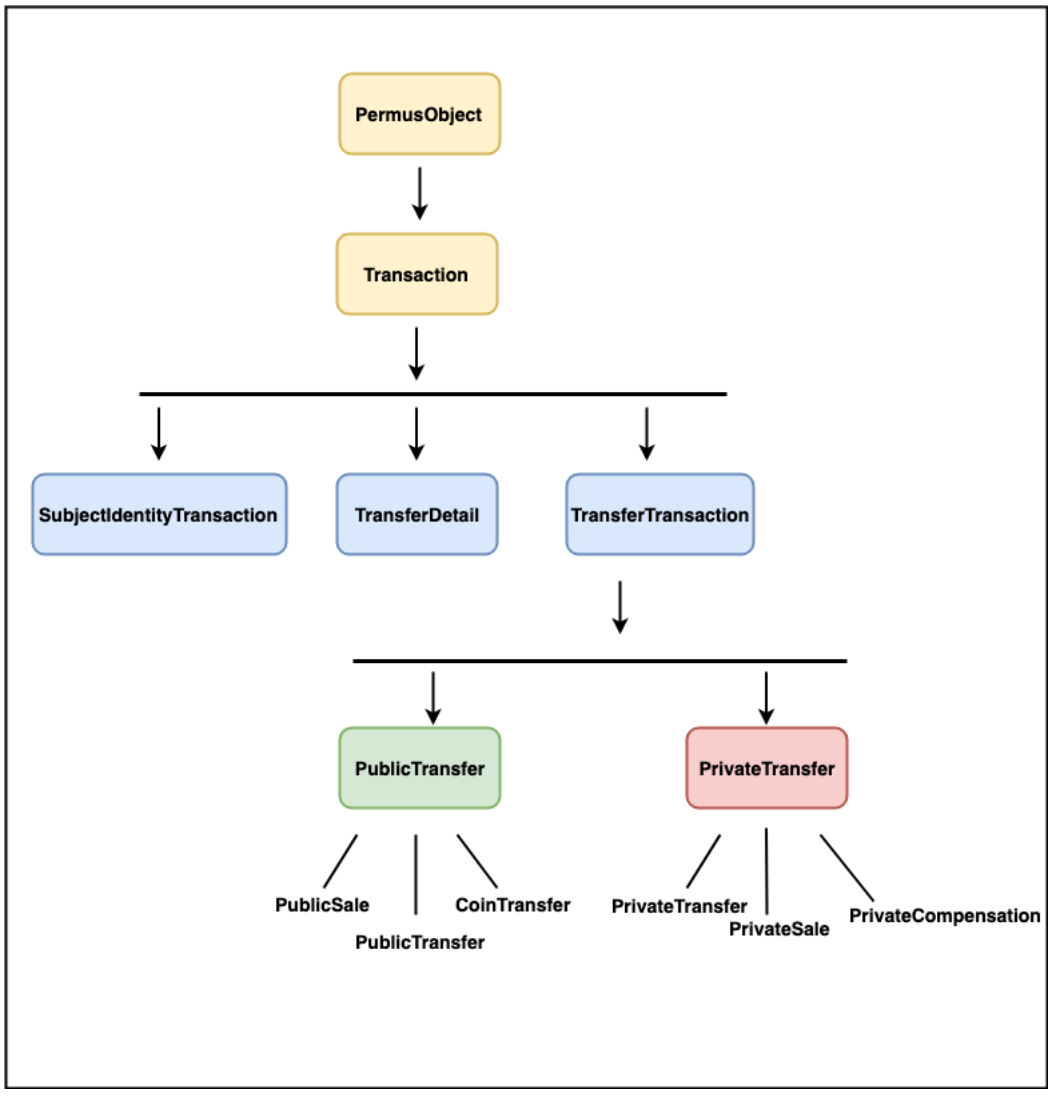

Figure 6.1. Inheritance tree for any type of transaction, from class PermusObject to all the specific classes.

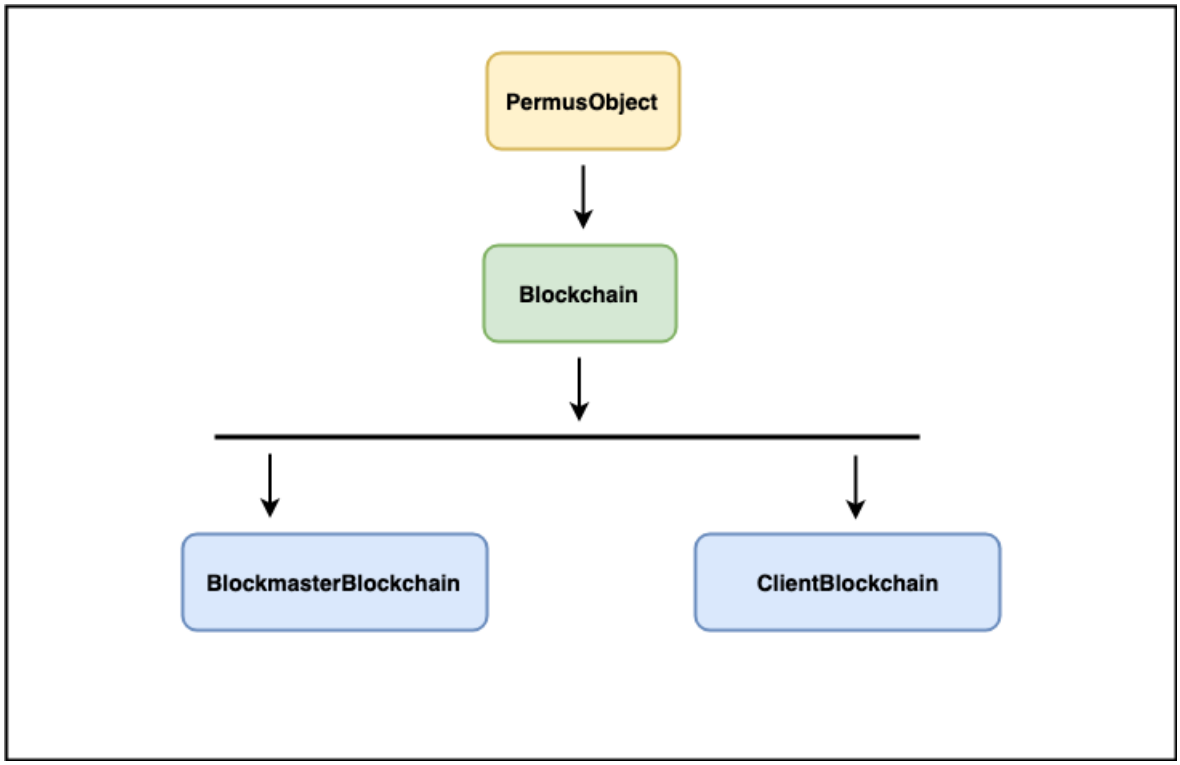

Figure 6.2. Inheritance tree for the different types of blockchain classes, from class PermusObject to the specific ones.

6.3 Blockmaster back-end

Although Permus Blockchain is essentially a distributed ledger in which every node maintains its own copy of the chain, it still requires the presence of a central node, the blockmaster, in order to operate, since there are some important services that need to be centralized for a non peer to peer blockchain to work properly. All authorized nodes of the Blockchain network have to be able to access those services, for this reason the logical choice was to implement the Webmaster as a collection of HTTP web services published on a web site.

6.3.1 Web API

The Microsoft .NET Framework offers many methods for implementing web services. The most recent and useful method is ASP.NET Web Api, a technology that allows to use the same framework and patterns to build both web pages and services, side-by-side in the same project. An ASP.NET web site can be published on the internet from any server based on a Windows OS running Internet Information Services, the Microsoft way of publishing HTTP web pages and services on the net. Web API is a programmatic interface consisting of one or more publicly exposed endpoints to a defined request–response message system, typically expressed in JSON or XML, which is exposed via the web most commonly by means of an HTTP-based web server.
In Web API, a controller is an object that handles HTTP requests. Methods in the controller object can be "connected" to the exposed endpoints and usually return objects defined in the Application Model serialized either in JSON or XML.
Blockmaster web application stores its own copy of the blockchain in a Database Management System (SQL Server Express in Permus case). A DBMS is better suited for storing large amounts of data server side because data access is fast and secure. A DBMS also simplify data maintenance such as backups.
For example, the most basic function of the BlockMaster, serving a public block to a remote node, is implemented by few simple steps:

- Web API automatically passes to the controller the required block id (called serial) as a parameter, using info obtained from the endpoint url.
- Using a specialized function of the class BlockMasterBlockchain, specialized version of class Blockchain, the controller obtains a copy of the required block from the local DB.
- The Controller ends returning the required block as it is. The serialization process is automatically handled by Web Api

Private blocks are also stored in the BlockMaster database. They are accessed and exposed to calling nodes in the same way as the regular blocks, with some important differences. First of all, Permus system identifies a private node solely by its hash code (the one that is recorded in the relative transfer transaction). The second and most important feature of a private block is that, in order to maintain privacy, private blocks stored (and served) by the BlockMaster are encrypted in a way that only those subjects involved in the related private transactions can access

them in clear since they are the only ones to possess the private key needed to decrypt them.

6.3.2 Client Authentication

Permus System is based on a private Blockchain concept. It means that only authorized users should be able to access its services. In Permus the only authorized users are the subjects member of the blockchain. Since Permus membership is based on X509 certificates, it was logical to implement a client authentication method also based on certificates.
Api methods that enforce this type authentication require the caller to include a special parameter, essentially a base64 encoded string containing a current system timestamp digitally signed with the client subject's certificate and encoded in p7m standard.
After decoding the p7m object contained in the string, the Api method code is able to verify the identity of the caller, since a digital signature can only be created by the owner of a certificate containing the related private key. Checking the signed timestamp also helps preventing the possibility that the same signed object is used by malicious code more than once in some "man in the middle" types of cyber attacks.
Decoding and validating digital signatures is a time consuming process. For this reason, when the identity of the caller subject is not really important, a token based access method is used. The first step a Client application is expected to do when it connects to Permus, is to call check_in Api method of the blockmaster. This method requires the already mentioned signed parameter to ensure the identity of the caller and returns a SystemInfo object, a very important piece of information containing, among other things, all the information needed by the client to decide if a synchronization procedure is required in order to update its own copy of the Blockchain. The SystemInfo object contains also a random string, called token, that the client can use for identification purposes in future calls of same non critical api methods.

6.3.3 Client Synchronization

Clients periodically call Blockmaster "info" API method, typically every few seconds, in order to check the current status of the blockchain. the method returns an instance of a SystemInfo object, also populated with some useful informations about the caller such as the presence of pending transfer transactions. The info method requires only the client token for authentication purposes and is the primary method for the client to assess the need for blockchain update.
When the client Blockchain obtains by the Blockmaster a SystemInfo object showing that new transactions or new blocks were issued in the distributed blockchain, the Client BlockChain synchronization process is automatically triggered.
The process simply consists in obtaining one by one all missing or updated blocks by the Blockmaster. Since it is impossible to rule out the possibility that the Blockmaster could give false blockchain informations, for example in case it is hijacked or should it falls under control of malicious users, every block obtained by the client

from an outside source must be checked and validated against all Permus blockchain rules, before it can receive the fully trusted status and subsequently parsed in order to extract useful information to be used in the client application and finally stored locally.
Validation and parsing is performed by the blockchain object and consists in the systematic validation of all digital signatures contained in the transactions, and check of all appropriate hash codes.

6.3.4 Transfer Transaction Brokering

BlockMaster plays the brokering role in Permus system in Transfer Transactions processes. In order to be valid, some Transfer transactions require the presence of only the originating subject digital signature. For example, a Coin Transfer Transaction only requires the signature of the subject that send coins, as throughout the system it is assumed that any receiving subject would accept donated coins unconditionally.
Other transfer transactions, however, require the signature of both subjects (internally named subjectFrom, the sender and subjectTo, the receiver) to be considered valid. This transfer transactions descend from the abstract class CosignedTransferTransaction and could exist in different valid states before their completion. For example the could be signed by the originating subject but not yet approved (and signed) by the receiving one. In this case, the transaction is said to be pending.

The Blockmaster is used to store, maintain and make available only to to the affected subjects a copy of pending transfer transactions until they can be inserted in the last block of the blockchain when they they eventually reach the “complete” status after the appropriate subjects interaction.
As we have already seen, clients periodically check if the BlockMaster has news for them, new Transactions or Blocks to be included in the local version of the blockchain, and pending transactions affecting the subject they represent. If the client detects one or more pending transfer transactions directed to its subject it notify the user using the User Interface or other means such as notifications using instant messaging systems (for example Telegram) that could be implemented in future to expand the system usability.
Since some transfer transactions are private e thus contain a reference to a private block, pending transactions must also contain a copy of the private block when appropriate. This is why the dialog between clients and Blockmaster intended to implement the transfer transaction completion process is performed using an intermediate object, an instance of TransferTransactionPackage class, that contains not only the pending transaction but eventually also the private block in its encrypted form. The private block is encrypted with the standard [PKCS-7], also called “Enveloping” using the public keys of the two subjects involved in the transactions.

6.4 Permus Client

Although in the real world a system like Permus would have little or no sense if not operated from client applications installed in mobile devices, during the development phase of the system seemed more practical starting developing a windows application based client to test the system.
This was required in order to minimize coding effort, since a windows application can be easily developed on Microsoft .NET framework, it can natively reuse all the code and functionalities contained in Permus.dll, the core component of the system. In this way it was possible to write a fully usable client application using just a few thousand lines of code, mostly used to define a simple but perfectly usable interface and implement some local processes, mainly used to handle data presentation and user interaction.

Microsoft .Net framework offers different technologies on building windows applications, among them Windows Presentation Foundation (or WPF), seemed the most effective for coding the reference client for Permus System. WPF separates the user interface from business logic, and employs XAML, an XML-based language, to define and link various interface elements. These elements can then be linked and manipulated based on various events, user interactions, and data bindings.
When it starts, the client looks for the X509Certificate of the subject to represent. The Certificate must be already installed and the Certificate Authority that issued it fully trusted by the OS. If the selected certificate is not yet enrolled in the blockchain the client will negotiate with the BlockMaster its enrollment. It is useful to remind that the blockmaster will enroll only certificates that fulfill expected requirements, such as been issued by a specific Certification Authority.
After establishing the BlockMaster connection, the client tries to synchronize its local copy of the blockchain. This could be a lengthy process the first time the client connects with a blockchain already containing hundred of thousands blocks. However, for the purpose of the system, it is essential that every block of the blockchain is obtained, parsed and stored locally before the client can be operated. Any block contains useful information for the client, for example information about other subjects, such as their certificate, and their coin balance calculated from the cumulative effect of their transfensfer transactions containing coins.
In order to speed up the Client application start process, at the end of every meaningful synchronization operation, the client stores locally a snapshot of the pertinent data previously parsed from the blockchain. In this way, subsequent client starts will be fast and virtually not affected by the size of the Blockchain.

6.4.1 Client overview

In this section a first prototype designed for Permus is presented.
The basic idea is that each user has access to the system through a personal certificate and is able to interact with other users, both to control old transactions and to create new ones.
The information of other users is mostly in the public domain, only what is concerning private transactions between third parties is not accessible.
A number of screenshots is shown to give an overview of the application as it has been described in the Process View [4.1.2]. The screenshots below show very basic features a user can see when launching the program but more complicated things that can be done inside the system as well:

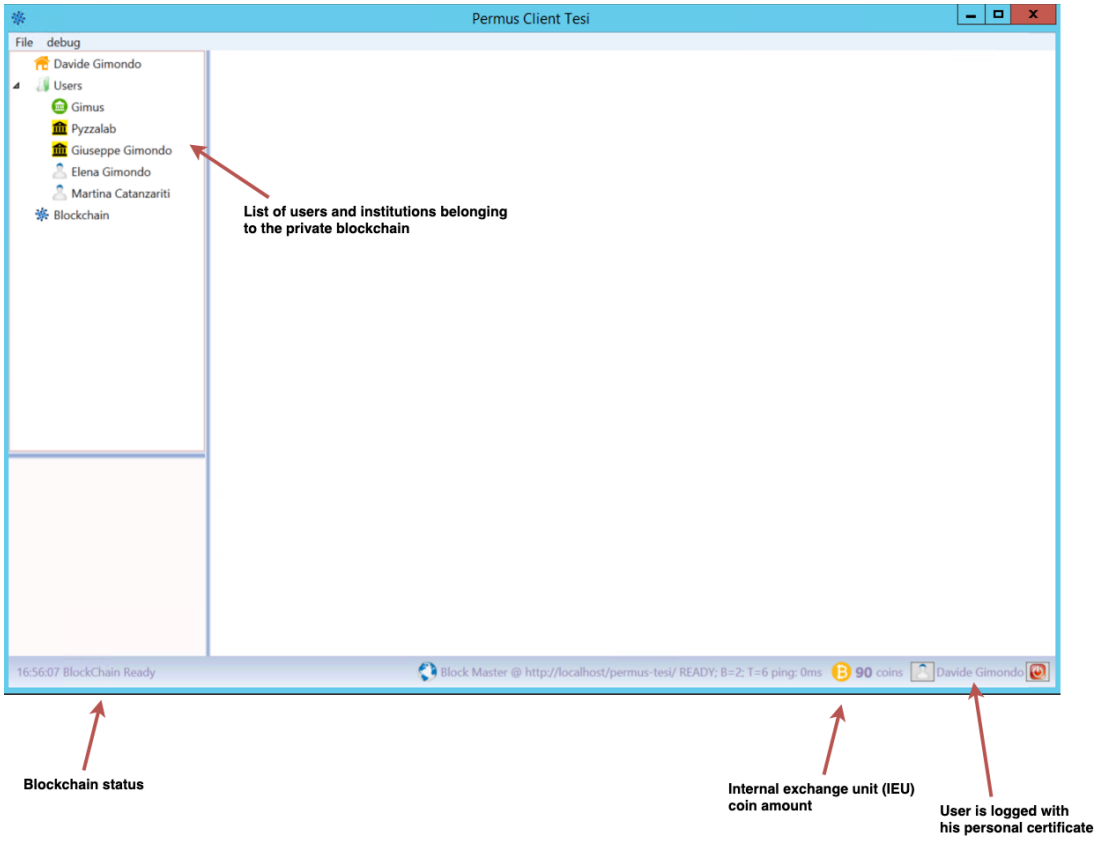

Figure 6.3. Initial screen when launching Permus, several information like the coin amount, blockchain status and users belonging to the system are displayed.

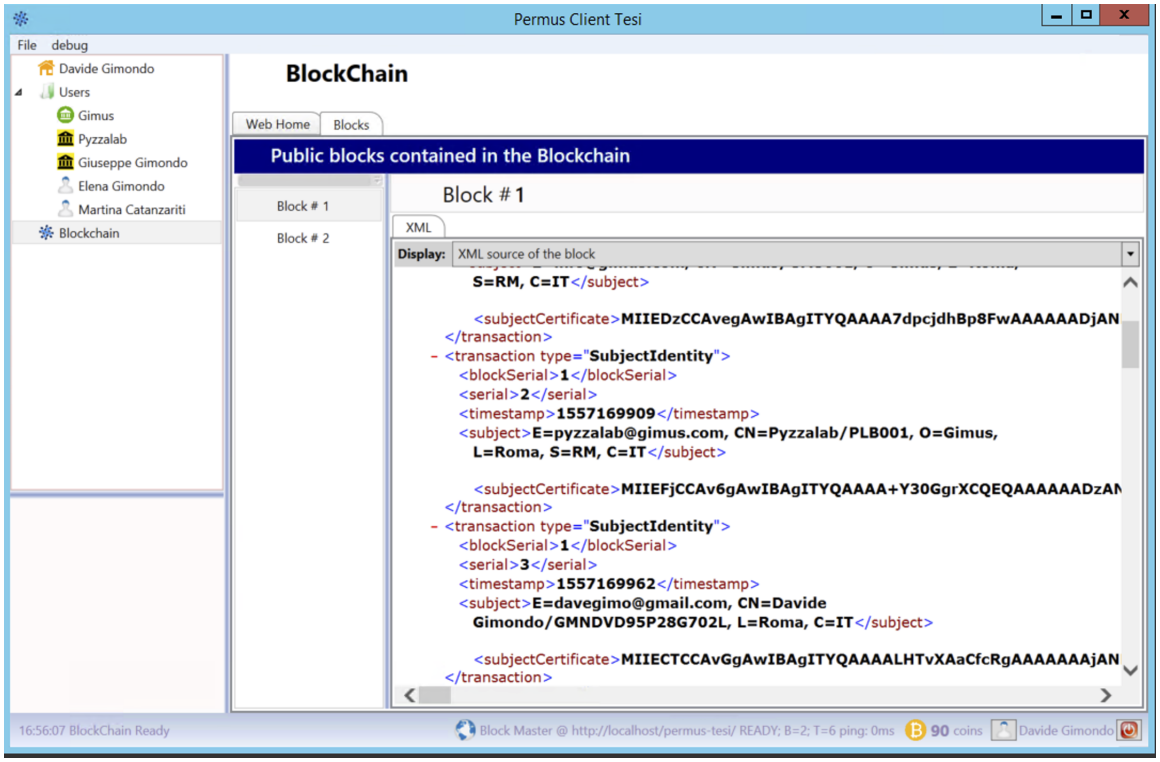

Figure 6.4. User can check all the blocks and transactions of the blockchain

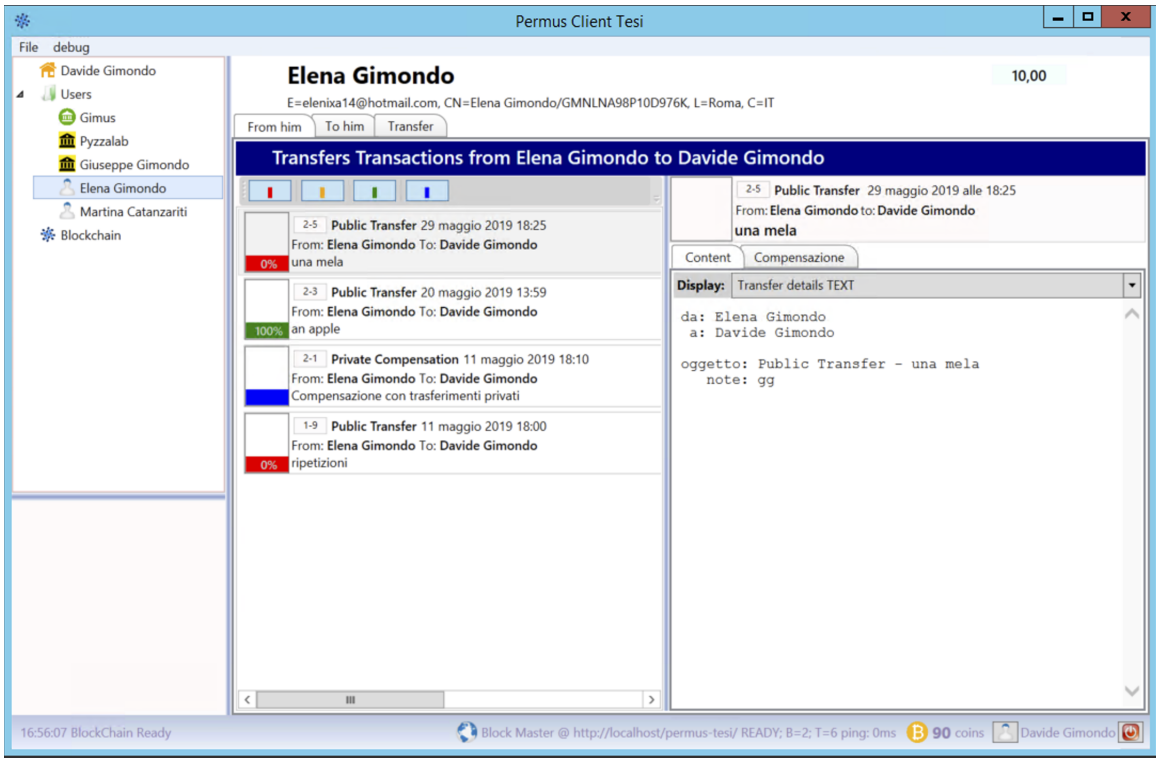

Figure 6.5. User can check personal information about a specific user and

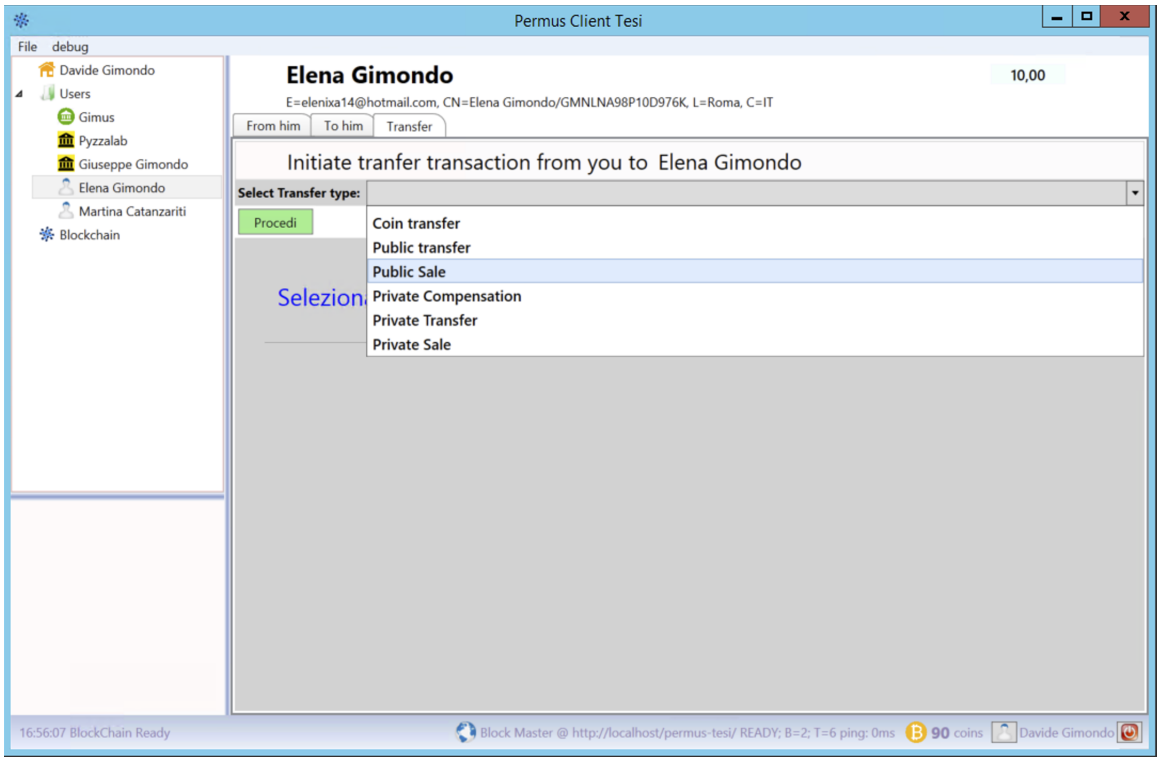

Figure 6.6. Screen where to choose what type of transfer wants to be done with a specific user

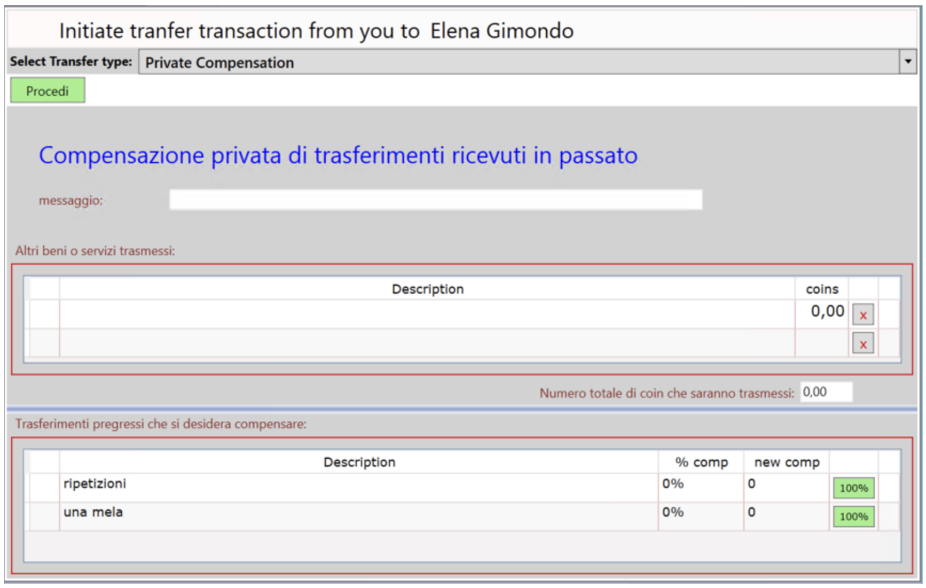

Figure 6.7. Private compensation screen: on the upper side it is possible to add the new transfers, on the other it is possible to compensate previous transactions

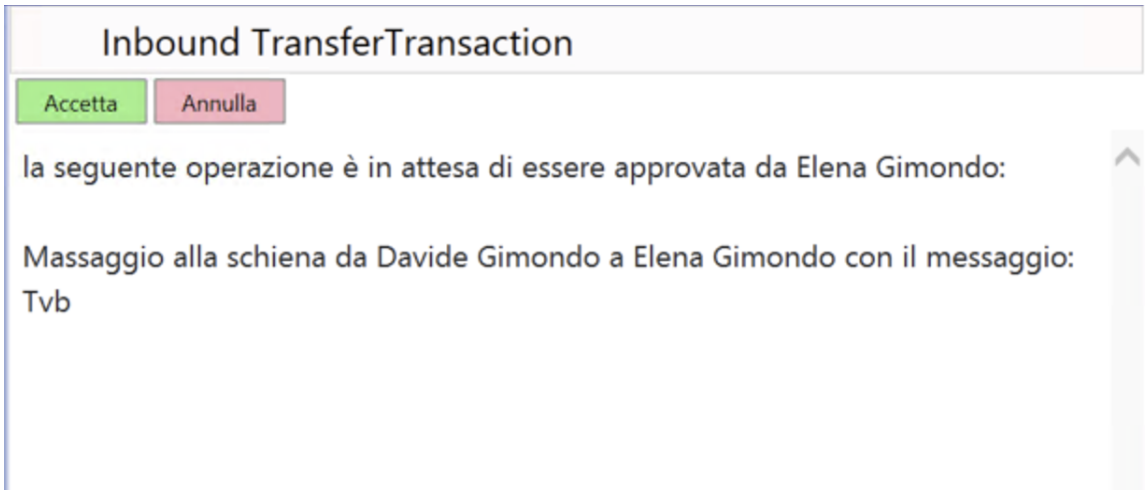

Figure 6.8. When a transfer is proposed, the receiver will be shown this window and can whether accept or deny the proposal

Chapter 7

Conclusion

Permus is my final project for the master's degree, it represents the end of a journey that began in September 2014 when I chose to attend Computer Engineering after high school.
The first time I heard about blockchain was three years ago, the thing that fascinated me was the versatility and the various scenarios in which it could be used: so it was not difficult for me to find the specific topic when I had to approach the master project.
Surely what has made this project interesting are the various areas it covers, from the design of the software to the security behind it and finally the communication between the various parties involved.
I feel lucky i was given the opportunity to get to know and touch with my own hands an extremely powerful technology that is gradually expanding more and more.

Similiar projects

Bartering involves the exchange of goods and services for other goods and services of perceived equal value. Today, money is a generally acceptable means of exchange, but before the creation of money, bartering was the primary means. People could barter exchange things like clothes for food, and services like a shoe repair in exchange for a cleaning service and so on.'
This article [2] explains in a few words what is the main idea behind Permus, although it has been written for a similiar Blockchain technology, called Mytradetoken (MYTC).
After some research, the projects that are the closest to Permus are the following:

- Mytradetoken [11]
- Bartsome [3]

According the description of their own websites, these services have the same idea of sharing economy like in Permus. Unfortunately, both projects seem to be incomplete and probably the creators decided to switch on other fields.
Although these websites were not giving all the information needed in order to give a

better "competitors overview", there are several features that point out how Permus is unique confronted to the other services in general:

- Closed environment: the system is designed to work for small communities where everyone knows each other
- Identity is certified: through the use of digital certificates inside of the system

Another reason why Permus is relatively different from these two services is that instead of aiming to revolutionize the entire economy by offering a worldwide system, the idea in Permus is to offer a service to small groups of people instead.
This means that the two economies can coexist, and users can decide which one is the most appropriate mode of exchange according to the situation.

Blockchain for social goods

The European Union is encouraging several projects to be developed through the Blockchain technology by investing on the five best projects, demonstrating that this technology could be integrated in different sectors.

> The challenge is to develop scalable, efficient and high-impact decentralised solutions to social innovation challenges leveraging Distributed Ledger Technology (DLTs), such as the one used in blockchains.
>
> DLT in its public, open and permissionless forms is widely considered as a ground-breaking digital technology supporting decentralised methods for consensus reaching as well as sharing, storing and securing transactions and other data with fewer to no central intermediaries. [6]

Future plans

Being able to work on a personal project about the Blockchain system, especially at the moment, may open up to several opportunities as it is nowadays a field in expansion.
The creation of Permus for my thesis represents only the beginning of this project, as I mentioned in the Implementation chapter [ch. 6] the code of the project is available on Github [13] and everyone will be able to fork the project and customize it according to his needs, using of course a personal server and everything else needed to set up the project.

The roadmap is pretty clear and the future plans for the projects are:

- Create a technical documentation of the project
- Create a smartphone version for Permus able to support personal certificates
- Improve UI/UX of the current prototype
- Present the idea as a proof of concept to Blockchain experts of Italy or at Blockchain conventions
- Find investors and possibly experts in management that would contribute to the project

Bibliography

[1] Architectural view model. https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

[2] Bartering issues solved by Blockchain. https://www.cryptocurrencywire.com/bartering-issues-solved-by-blockchain/.

[3] Bartsome. http://www.bartsome.com/.

[4] BigChainDB. https://www.bigchaindb.com/.

[5] Block picture example. https://bitcoin.org/en/blockchain-guide.

[6] Blockchain for social goods EU. https://ec.europa.eu/research/eic/pdf/infographics/eic_horizon-prize-blockchains.pdf.

[7] Blockchain Transaction. https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/.

[8] Blocks in Blockchain. https://www.blockchaintechnologies.com/blockchain-technology/.

[9] Corda. https://www.corda.net/.

[10] MultiChain. https://www.multichain.com/.

[11] Mytradetoken. https://thetokener.com/ico/mytradetoken.

[12] Openchain. https://www.openchain.org/.

[13] Permus on Github. https://github.com/davegimo/Permus.

[14] Requirements Analysis. https://searchsoftwarequality.techtarget.com/definition/requirements-analysis.

[15] Use case diagram. https://www.smartdraw.com/use-case-diagram/.

[16] Blockchain. https://en.wikipedia.org/wiki/Blockchain (2018 10 2).

[17] Cryptographic hash function. https://archive.is/20121208212741/http://wiki.crypto.rub.de/Buch/movies.php (2018 10 2).

[18] Dob, D. Blockchain Transaction. https://blockonomi.com/permissioned-vs-permissionless-blockchains (2018 17 7).

[19] Economist. Blockchain Blocks. https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things (2018 10 2).

[20] Merkle, R. C. Merkle Tree. https://patents.google.com/patent/US4309569 (2018 10 2).

[21] Michael McLeay, R. T., Amar Radia. Money in the modern economy, an introdction. https://www.bankofengland.co.uk/-/media/boe/files/quarterly-bulletin/2014/money-in-the-modern-economy-an-introduction.pdf?la=en&hash=E43CDFDBB5A23D672F4D09B13DF135E6715EEDAC (2014).

[22] Pitzalis, J. Blockchain Transaction. https://www.quora.com/What-is-a-Blockchain-transaction (2019 30 1).

[23] Smith, A. An inquiry into the nature and causes of the wealth of nations, Vol 2 (1776).