Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

The Usage of Decentralized Applications for Enhancing the Donation Process

Majd Turfa

Course of Study:	Informatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Ghareeb Falazi, M.Sc.
Commenced:	March 5, 2018
Completed:	September 5, 2019

Abstract

A number of social and economic terms have been influenced and changed by the development of blockchain technology and its concepts of smart contracts, and many business processes have been affected and changed. In addition, multiple used methods of various fields, in different domains, have been reconstructed and further developed using the blockchain to solve many current problems and conflicts.

Donation is one of those places where we can notice a lack of transparency and a decreasing trust level. Therefore, in this thesis, we discuss the issues of currently used methods, and we also suggest and implement two use cases that work on the blockchain, with the aim of increasing both the transparency and efficiency of civil organizations' work. This is a step towards reestablishing trust in those organizations.

The first introduced use case is oriented toward the transparency of nonprofit organizations, and it does not replace the currently used methods but rather completes them. The other use case is built as an approach, and it is more oriented towards the donation methods, donors and the management of projects. Finally, the two implemented use cases are combined by a shared task.

Kurzfassung

Die Blockchain Technologie und die Konzepte von Smart contracts haben viele wirtschaftliche and soziale Begriffe sowie Geschäftsprozessen beeinflusst und verändert. Außerdem wurden etablierte Methoden in verschiedenen Bereiche mit Hilfe der Blockchain rekonstruiert und weiterentwickelt, um viele gegenwärtige Probleme und Konflikte zu lösen.

Spenden ist eine der wichtigsten Bereichen, in denen man spürbar unter dem Mangel an Transparenz von Informationen und Vertrauen leidet. Daher wurden im Rahmen dieser Arbeit die Schwächen der heutzutage benutzten Methoden diskutiert und schließlich wurden zwei Ansätze mit Hilfe der Ethereum Blockchain und Smart contracts konstruiert. Durch deren Nutzung soll die Transparenz und Effizienz von wohltätigen Organisationen gesteigert und das Vertrauen gestärkt bzw. wiederhergestellt werden.

Für das erste vorgestellte Konzept steht die Transparenz der Arbeit der zivilen Organisationen im Mittelpunkt, wobei bereits existierende Methoden nicht ersetzt, sondern erweitert werden. Während beim zweiten Konzept der Fokus auf dem Handeln von gespendetem Geld, den Spendern und der Verwaltung von Projekten liegt. Letztlich wurden beide Ansätze mit einer gemeinsamen Aufgabe kombiniert.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

As of May 2018, the world population was estimated to have reached 7.6 billion people. Of these people, 805 million are considered to be in extreme poverty (living at a consumption or income level below \$1.90 per day), while approximately 6.08 billion people are living on less than \$10 a day [TLCM15], and donations form an important social tool that is currently fighting against these problems.

Donations were often organized through the Church and other religious institutes; however, the number of nonprofit organizations has grown enormously over the years. Despite this growth in many countries, the levels of individual donations have dropped. One of the factors that explains the decrease in giving is the lack of public confidence in charities. Furthermore, charities are always under scrutiny about the discharge of available money, while the end receivers remain at the mercy of charities. Indeed, many studies [RFP12] [TCC16] have demonstrated that the main motivation for trust and confidence in nonprofit organizations is their ability to satisfy people's demands for transparency about their activities.

Due to its transparency, security, non-central authority and rule enforcement, blockchain technology and the related use of smart contracts could provide a better environment to develop a new donations model, which will lead to an increase in the transparency level as well as the efficiency of the donation process, a decrease in the possibility of corruption and misleading and a simultaneous increase in individual giving.

Motivated by the above-mentioned information, we present two use cases in this thesis, implemented as decentralized applications (DApps), which focus on enhancing donations and that are able to be deployed and run on the Ethereum blockchain.

Outlook

This thesis is divided into six chapters. The first chapter contains a general introduction to the entire work, while the second chapter is a background chapter where we discuss the concepts of the blockchain and the scientific background knowledge required to understand the thesis. Different subjects, such as cryptography, Ethereum, smart contracts and DApps, are explained. In the third chapter, we explain how a donation works and why it is important; we also present our motivation and some similar works related to our thesis. In the fourth chapter, after explaining the transparency term, we present the first use case, which is related to that term. In the fifth chapter, we discuss the problems that some of the donation models have, along with the solutions, and we introduce the main approach we created and the different components we developed, such as the work of each smart contract, how they are related to each other and their roles in the system. At the end, in the sixth chapter, we present the summary of the work and suggest how the work could be improved. Finally, there is a bibliography where we cite all the articles and books we referenced in the thesis.

This thesis starts with Chapter 2
The programs can be found on github under this name :
The Usage of DApps for Enhancing the Donation Process

2 Background

Because the blockchain is a new technology and unfamiliar, we will cover in this chapter the basic concepts, Ethereum platform and decentralized applications.

2.1 Basic Concepts

The first concept and the backbone of this new technology that we are going to explain is cryptography.

2.1.1 Cryptography

Cryptography [Sch16] is the science of converting data using encryption into a particular form to share information and communicate selectively because the converted data is unreadable for an unauthorized user.

The two basic techniques to encrypt and decrypt information are the Symmetric and Asymmetric cryptography. The Symmetric methods use the same secret key for encryption and decryption of data which means that both, the sender and receiver, need to have the same secret key, which may be a disadvantage in some cases. Instead of using the same secret key for encryption and decryption, in the Asymmetric methods two different but mathematically related keys are used a public key and a private key. In this method one of the keys will be used to encrypt the information and the second key for decrypt it, that allow any sender to encrypt information with the public key of the receiver encrypt (plain, public key), and then only the receiver can decrypt it using his own private key decrypt (cipher, private key).

A hash function is a function that converts data of any size into a fixed bit-length output `h(m)= x`. There are different types of hash functions, however we will focus on cryptographic hash rather than other functions because they have an important role in the blockchain technology. Cryptographic hash functions have the following properties [Dre17]:

- Deterministic: the same data will result always the same hash.
- One-way function: it is practically impossible to invert given hash value back to its original data.
- Collision resistant function: it is practically impossible to find two not equal data sets `M != M’` with `h(M) = h(M’)`.
- Sensitive to change: any small change in the data will change the hash value dramatically.
- Quick to compute: it is easy and quick to compute and verify the hash of a given data.

An example for a hash function that fulfill these properties is the SHA3-256 which is a member of the Secure Hash Algorithm family of standards, which deliver a fixed size 256-bit (32-byte) hash.

Digital signature

Digital signature [DSS94] are the counterpart of hand signature in the digital world, digital signature uses the concept of Asymmetric cryptography and hash functions to prove not just the ownership of digital messages or documents but also the integrity of that message. A digital signature schema mostly consists of 3 algorithms which are:

1. A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
2. A signing algorithm that, given a message and a private key, produces a signature
3. A signature verifying algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

As shown in Figure 2.1 Alice creates a digital signature of the data by encrypting the hash value of it using her private key `encrypt(hash(data), private key) = signature`, after that she sends the digital signed data through an untrusted or unsafe network to Bob. Bob can verify the signature now by decrypting the signature using the public key of Alice `decrypt(signature, public key ) = X` , and then compare the result with the hash value of the data. if `X == hash(data)` , it means that Alice is the owner or sender of the data, and the data has been not altered.

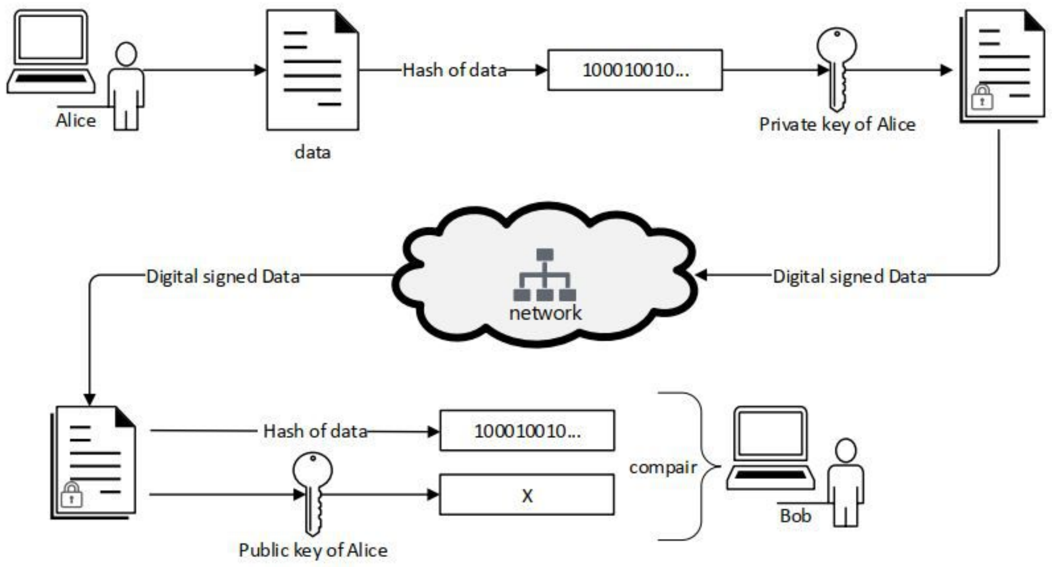


Figure 2.1: Sending and verifying the data between two entities through a network using the digital signature method

Merkle Trees

A Merkle Tree [Mer79] is a binary tree constructed by hashing paired data (the leaves: which save the hashing value of the data instead of the data itself), and each parent node is the hash value of its two children, pairing and hashing the results until a single hash remains. The Figure 2.2 shows a simple Merkle Tree constructed by four Blocks of data A, B, C and D. The four leaves h(A), h(B), h(C), h(D) are the hashes of the data, and the parent node P1 (of A and B is) equal to `h(h(A),h(B))`, and the same schema of the other parent node P2. The root P0 is equal to `h(P1,P2)` which is called the root hash.

Merkle trees are used to compare data sets using the properties of cryptographic hash functions, that mean instead of comparing a big amount of data sets, we only need to compare the hash root, and if the hash root is different, then it proves that the data sets are not equal. The obvious advantage of using the tree structure is the ability to confirm the accuracy of a data set without the necessity of downloading and verifying all other data sets. Assuming that a user need just the data set C, then it is not necessary to download the other sets A, B, D and to rehash them and comparing the value with the hash root, a user just need to get the hash values of data set D and P1 to be sure that data set C is correct and not altered.

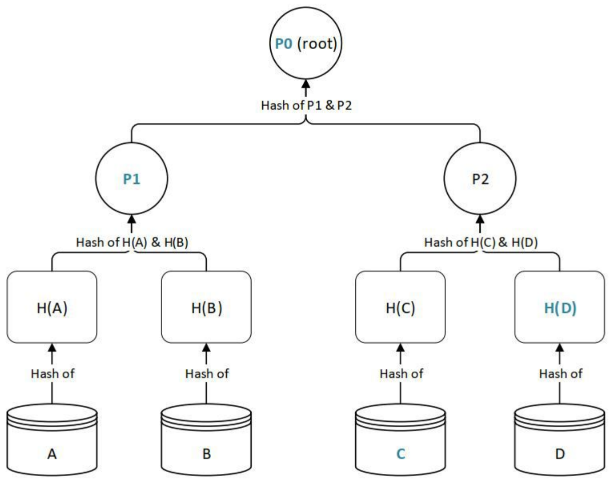


Figure 2.2: Merkel tree that contain 4 sets of data

Timestamp

The idea of Timestamp [Nak08] is based on taking the cryptographic hash of an object and publish it on a public network, the Timestamp of that object will prove that the object has been existed at the time of publishing that information. That works because of the properties (one way and Collision resistant function) of cryptographic hash functions and assuming that the network is immutable to hacks and manipulation.

2.1.2 Peer to Peer

There are many ways to implement a software system; indeed the first fundamental step before developing any software system is to determine which kind of architecture is required for the system. There are various architecture systems, but most common are the centralized, decentralized and distributed systems. Each of those has its advantages and disadvantages, for decentralized systems is the higher reliability one of the advantages, however the higher cost of coordination and communication between the nodes is one of its problems.

Peer to Peer network is a special art of decentralized system, it refers to a group of computers acting like nodes for sharing resources directly between themselves without using an intermediate server. Commonly all nods in the P2P network has the same roles and rights. The Peer-to-peer technology has evolved through several stages. While the earliest P2P networks were mainly file-sharing applications, current uses of P2P technology are much more diverse and include the distribution of data, software media content, as well as Internet telephony and scientific computing. [RD10]

The Peer-to-Peer networks have some unique challenges which are more complicated than other networks because the lack of a trusted intermediated or Master node in the network. Some of those problems are:

- The Sybil attack occurs when a single node may be able to assign itself many identifiers, the vulnerability to a Sybil attack depends on how cheaply identities can be generated, and their influence and functionality on the network.
- The Double spending problem occurs when we move from the Internet of information to the Internet of value where a transferred massage has a value associated to it. The sender in this case can make copies of that value and use it more than one time.
- The Byzantine fault tolerance [Lam82] describes an old problem occurs in an trust-less environment like on Peer-to-Peer networks. Because peers cannot trust each other and one or more of them can lie, is the communication not reliable. Therefore, the network requires consensus mechanism which is a process of agreement between huge numbers of peers on a final state of data on the network.

2.2 The Blockchain and Bitcoin

Bitcoin was introduced by pseudonym Satoshi Nakamoto in 2008 ,in the article „Bitcoin: A Peer-to-Peer Electronic Cash System“[Nak08] and the first version v0.1 was in January 2009. Satoschi Nakamoto present in his article a new concept to solve the double spending problem in Peer-to-Peer networks using a cryptographic proof instead of a trusted third party. This concept gives the Peer-to-Peer networks the ability to send values like digital Assets and currencies and not just information like it used to be.
To solve the problem, Nakamoto, in addition to Digital signature and timestamp server, suggested the use of a data structure - called the blockchain - to record the communication between the peers in the form of transactions.

2.2.1 Blockchain

The blockchain is a data structure, similar to linked-list, but in form of a timestamped continuously growing chain of blocks in which data is stored. Each block has a set of data and linked to other blocks in a secured way using identifiers and cryptographic hash functions. The blockchain could be also explained as a distributed ledger, which works like a database, to store transactions in an invertible way which mean transactions cannot be modified and are resistant to manipulation and attacks.

The Data Structure

Transaction is the fundamental unit of a blockchain and it represents and register a transfer of a value or information from one peer to another [Bas17]. Transactions on public blockchain s like Bitcoin and Ethereum are not encrypted, publicly visible, and they could contain meta-data, inputs and outputs.

Blocks are the other fundamental unit which groups transactions together and gives blockchain its structure. The contents and size of a block may vary depending on the platform, but each block has a header which includes a Merkle-root of a set of data or transactions, nonce, timestamp and the unique identifier of the previous block header. The Genesis block is the first block on the structure and it does not reference to a previous block, furthermore any two nodes in a network can only pair with each other if they both have the same genesis block. Every new block that added to the chain is linked to the latest block using the hash value of the last as shown in Figure[1] 2.3, and therefore any change in one of the previous blocks in the chain will affect all other blocks, because it will change the hash value of all the blocks that come after the manipulated one.

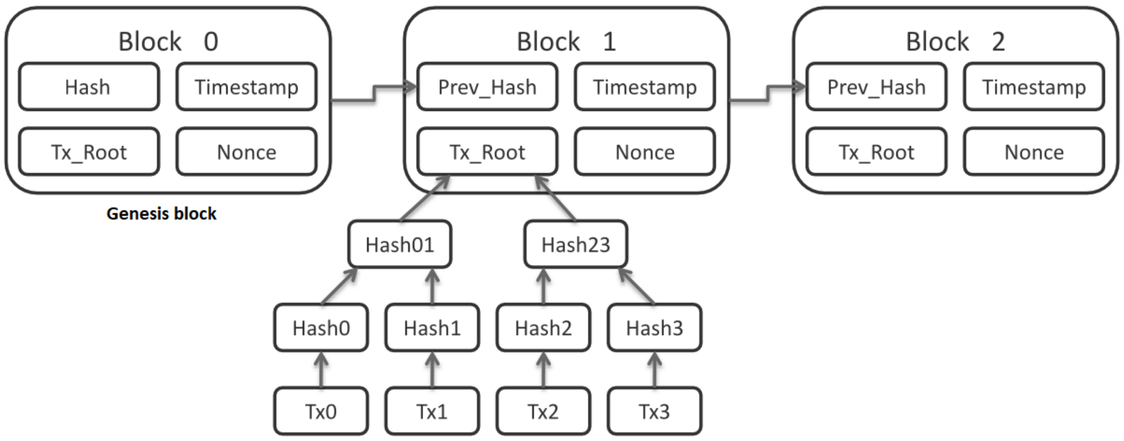


Figure 2.3: The structure of blockchain and the contains of a block header

[1]https://de.wikipedia.org/wiki/Bitcoin

CAP Theorem

The CAP theorem [CAPB00] states that each distributed system has three Properties which are Consistency, Availability and Partition Tolerance, and only two of those properties can be achieved at the same time. Mostly there is always a trade-off between consistency and Availability. In Bank systems the consistency is chosen over the availability. The blockchain cannot beat the CAP theorem and to achieve a higher availability it chooses to use eventual consistency.

Mining

The process of adding blocks into the chain is called Mining, but first in order to create a block, miners must choose transactions from a queue, with respect to the allowed block size, and then validate them. As an example, in case of Bitcoin, the maximum block size is 1 MB, and therefore depending on the size of each transaction a maximum number of transactions that fill up to 1 MB can be added in one block. To ensure the consensus of the blockchain and to prevent attacker from adding invalid blocks to the chain, the miners compete each other in a competition and strategy that based on the proof of work Protocol. Proof of work protocol (POW) is developed as a mechanism to achieve the necessary consensus in networks, it depends on a mathematical hash problem, which requires a huge amount of hash power and it is hard to solve. Solving this puzzle works as a proof that the miner, who add the block, has used some heavy-duty computing resources. It is important to know that in case of a conflict between two versions of the chain the longest chain will be accepted by the nodes because more computing power is related to it.

Advantages

The major advantages of the blockchain Technology are [BM17]:

- Transferring value: until 2009 the Internet was just limited to, what currently called as, the Internet of information which mean that it is only possible to transfer information is a securely and effectively way, but the blockchain upgraded that term by making the transfer of value between peers almost as easy as transferring information, and therefore we moved to new age of the Internet of Value.
- Immutability: transactions once stored on the blockchain are immutable, however there is the possibility of rolling back the changes and double spend, but it will require an unaffordable amount of computing resources to achieve the creating of the longest chain before all other miners and that what is called as 51% Attack.
- No central authority: a blockchain network is a truly peer-to-peer network where all nodes are equals which mean there is no master nodes who could control the network.
- Secure and Transparent: the transactions on the blockchain are public and verified by all nodes and additionally each block content a cryptographic hash of itself and the hash of the previous block which make any modifying on any transaction change the hash completely.

2.3 Ethereum

Bitcoin was the first generation blockchain network, however the active developer community proposed several enhancements to the protocol of Bitcoin and has come up with alternative types of coins and applications which had its own public or permission blockchain network. Vitalik Buterin introduce the second generation of the blockchain network called Ethereum in 2013. V.Buterin, in his whitepaper [But13], proposed that instead of having a separate blockchain for each types of application, a single programmable blockchain network where developers can write programs and develop decentralized Applications. The formal verification of Ethereum proposed by Dr. Gavin Wood in the yellow paper [Woo14].

2.3.1 The Data Structure

Ethereum is an Open Source Protocol which use blockchain technology and P2P network to build a public platform that featuring Smart contract which are basically programs, their code is stored on blockchain and it will be executed using an Ethereum virtual machine (EVM) which is implemented by all clients.

At the core of the Ethereum network, there is an Ethereum blockchain running on the P2P network and has its own currency called Ether. Secondly, there is an Ethereum client which is developed using different languages but most popular one is go-Ethereum (geth), geth has been developed using Golang, it runs on the nodes and connects them to the peer-to-peer Ethereum network, from where the blockchain is downloaded, and stores it locally in simple Database, a fast key-value storage that supports Put / Get / Delete, called levelDb. Meanwhile geth provides various functions, such as account management, mining and even running a private Ethereum blockchain. Another component is the web3 library that provides methods for the interaction with the blockchain via the Remote Procedure Call (RPC) interface.

The first release of Ethereum was known as Frontier, the second and third Ethereum releases are called homestead and metropolis.

Turing Completeness

Bitcoin has a script programming language, but it was limited because it does not support loops and conditional jump, however to achieve the view of V.Buterin of building a programmable blockchain, a Turing complete programming language was necessary. A programming language called Turing complete if it can do anything that an universal Turing machine can do, which mean the ability of doing any computational calculable function, given enough time and memory. Because the programming language now supports loops, there is the issue of find existing infinity loops which is not solvable even by a Turing machine, this simply means - a program that could check if another program will determinate and give a result back does not exist [2].

[2]Book: Schöning, Uwe, Theoretische Informatik - kurz gefasst

Infinity loops could have bad consequences on this kind of networks because the functions should run on each of its nodes, and if there is an infinity loop in a program, then all nodes will stuck and as mentioned there is no way to find out if a program will determinate or not, however Ethereum came with a solution to avoid this vulnerability by using "Gas-system". In this system each operation that will run on the blockchain has a cost and there is a gas-limit that specifies the maximum number of steps that are allowed to run on a single block.

Accounts

There are two types of accounts in Ethereum. The first type is Externally Owned Account (EOA), which is owned and controlled by users, have Balances, represented by an address and it is generated through an asymmetric complex encryption method called the elliptic curve cryptography (ECC). Each account is a pair key of public and private key, the address of an account is the hash value of the public key. An EOA has also a nonce that works as a counter of how many transactions have been already executed by this account, nonce are helpful to determinate the sequence of created transactions.

The second types of accounts are contract-accounts which have additional data associated to its balance and address. It has no private key, however the data storage and balance are controlled by the logic of the contract.

Transaction

Transactions are signed data messages, that could hold value, sent by an EOA to another EOA or to contract-account. But unlike Bitcoin, transactions in Ethereum platform change the state of the network (S, t) –> S' as shown in Figure 2.4 the initial state S represents the state before the transaction execution and the final state S' is the modified state.

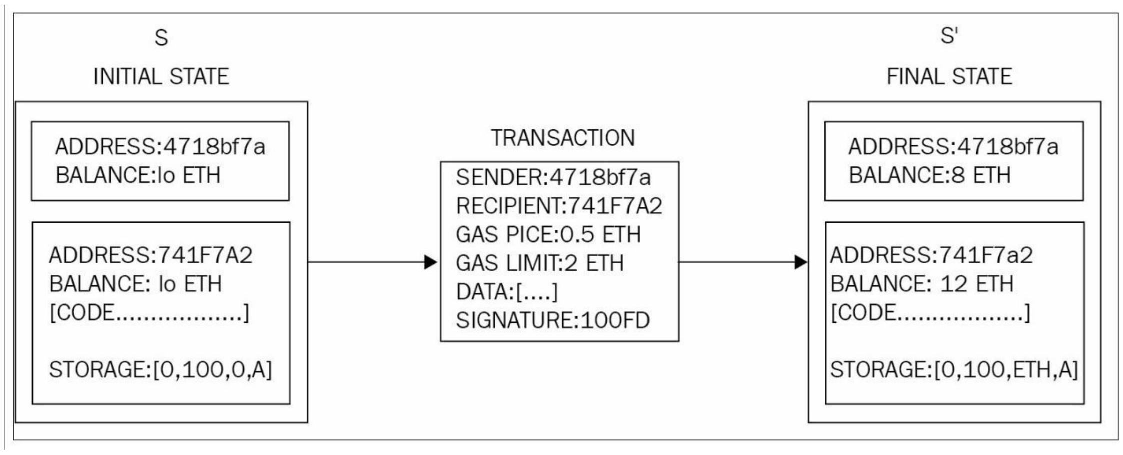


Figure 2.4: The genesis state is transformed into a final state by executing transactions [Bas17]

Transactions are signed by the sender's private key, and it include the recipient of the message, a signature identifying the sender and prove its ownership, the amount of ether to transfer (could be zero), the gas limit, the gas price, which is the cost the sender of the transaction is willing to pay for each computational step, and additional data is optional. Transactions could be used to transfer value between accounts, to invoke a method in a contract, or even to deploy a new contract by providing a valid contract code in the data.

The main different between a transaction and a message in Ethereum is determinant by who is the sender, accordingly transactions can only be triggered by EOAs, while messages are triggered by contracts to interact with other already deployed contracts or EOAs.

Blocks and Mining

Like Bitcoin transaction are chosen by Miners and grouped together after the validation process into one Block, which will be added to the chain after solving the mathematical puzzle. However, Ethereum developed its own protocol that called Greedy Heaviest-Observed Sub-Tree (GHOST) it uses Ethash and the time between two Block is decreased to about 15 seconds and to avoid corresponding problems caused by that decreasing of time such as centralization, it counts the stale block and reward the second and third block producers. Those blocks which are called uncle blocks are added in calculations to figure out the longest and heaviest chain of blocks.

A block in Ethereum is limited to 2KB which is smaller in compare to the 1 MB block in Bitcoin. The Block Header is the most important part and contain the Merkel root of all related Information as shown in (Figure 2.5).

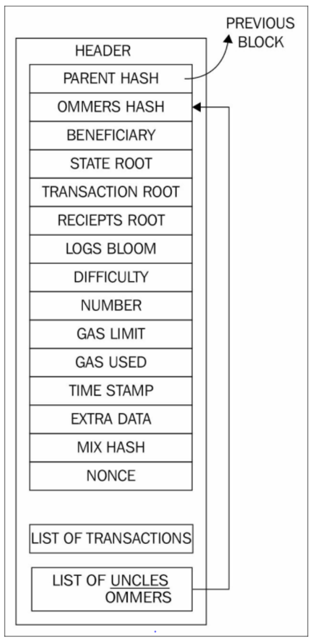


Figure 2.5: The content of a block header in Ethereum [Bas17]

- Parent hash: is the hash of the previous block.
- State root: contains the hash of the root node of the state tree after processing and finalizing all transaction.
- Transactions root: is the hash of the root node of the transaction tree, which represents the list of transactions included in the block.

- Logs bloom: is a bloom filter that is composed of the logger address and log topics from the log entry of each transaction receipt of the included transaction list in the block.
- Timestamp: is the epoch Unix time of the time of block initialization.
- Nonce: is a 64-bit hash (a number) that is used to solve the hash puzzle.

Ether and Gas

As we mentioned before, Ethereum uses Ether as its own currency with an atomic unit called Wei; 1-Ether = 1×10^{18} Wei. Ether is the basic unit of transfer of value and has been used to reward miner for their efforts and by users to interact and pay to DApps whose do not have their own coins.

Gas is the name of the crypto-fuel of Ethereum, it is used as a measurement of the number and kind of atomic instructions running by the EVM for performing an operation. While some instructions like "call"take a small amount of gas just 700, other operations which store data on the blockchain are more expensive such as "contract-creating"which take 32000 and ßtore"20000. The Gas price is unstable but it is fixed per operation, and each transaction should charge a certain amount of gas the user willing to give to the miners as fee. The cost of a transaction is calculated using this formal:
`gasCost(tx) = gasPrice * gasUsed`

2.3.2 Popular Usage

The currently popular Usage of the Ethereum blockchain is the creation of tokens and securing them which enabled the new token economy.

Tokens

Tokens are a simple way to store values of an underlying real-world asset on the blockchain. Tokens are the counterpart of things as money, coins, or stocks in the real world; furthermore, they can be used for different purposes such as ownership prove, ticket to pass, Crowdfunding or even as a vote in a system of voting.

The usage of Token has been increased rapidly after the lunching of Ethereum, which provided a simple environment to deploy Smart contracts that can handle the creating of tokens and managing them. And due the growing involvement of people in the Token's economy and to make sure that the tokens are available and usable easily through wallets exchanges and other later suggested protocols, the Ethereum community defines some specific standards that a Token contract should implement, such as the following functions and events [VBERC2016]:

- totalSupply(): Get the total token supply.
- balanceOf(address owner) constant returns (uint256 balance): Get the account balance of another account with address owner.
- transfer(address to, uint256 value) returns (bool success): Send tokens to address to.

- transferFrom(address from, address to, uint256 value) returns (bool success): Send tokens from address from to address to.
- approve(address spender, uint256 value) returns (bool success): Allow spender to withdraw from your account, multiple times, up to the value amount.
- allowance(address owner, address spender) constant returns (uint256 remaining): Returns the amount which spender is still allowed to withdraw from owner.
- Event Transfer(address indexed from, address indexed to, uint256 value): Triggered when tokens are transferred.
- Event Approval(address indexed owner, address indexed spender, uint256 value): Triggered whenever approve(address spender, uint256 value) is called.

Currently the ERC20 Tokens are the most used standard, however there is some advance tokens like ERC721 which mostly inheritance the basic functionality from the ERC20, but it add some additional functionality to make each token different from the other.

However, this usage is suffering from one of the problem that the Ethereum and similar blockchains have of which is the scalability issues.

Scalability

Scalability is one of the important Problem which is suffered by all of the blockchains that based on Proof-of-work, and as mentioned in the last section the size of Blocks in - the Ethereum, and also in other blockchain network - is limited, and considering the time needed to process and validate a transaction by a normal user - because all nodes should run every transaction - the Ethereum blockchain can only support roughly 12 transactions per second which is a small number compared to the real world applications, such as the 45,000 processed by VISA.

For this reason, scalability is the biggest issue that need to be solved in the blockchain technology. Some of the introduced approaches for solving the problem are proof of stake [3] and Sharding[4] which promised to scale Ethereum, Plasma[5] which is an off-chain solution and Raiden[6] a protocol that support micropayment.

2.4 Decentralized Applications

Blockchain technology enables the development of DApps that are based on smart contracts, which we will explain in this section.

[3]https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs
[4]https://github.com/ethereum/wiki/wiki/Sharding-FAQs
[5]Plasma: Scalable Autonomous Smart Contracts.
[6]https://raiden.network/

2.4.1 Smart Contract

A smart contract describes an immutable program or protocol that lives on a network, manages **value** and information and enforces the rules on the participant.

A smart contract does not necessarily need the blockchain to process his purpose, and even the idea of smart contracts was introduced many years before the appearance of the blockchain - in 1997 by N.Szabo [Sza97] - but with no real use cases because of the lack of a trusted platform. However, the blockchain seemed to be the perfect environment for running and realizing this idea due to its trust-less structure, security and immutability, and with Ethereum, it became one of the main cornerstones of this technology.

Structure and Life Cycle

A smart contract describes the logical functionality of the program, and it can be written in several high-level languages, such as Solidity and Serpent. After writing the program, a special compiler is used to compile the code into a byte code, which will be stored on the blockchain and which can be executed on each node on the network with the EVM.

A smart contract on Ethereum includes the following:

- A unique identifier: the address of the contract on the network, which will be generated once the contract is deployed to the network.
- State Variables: these represent the storage of the program, and they can only be changed through the implemented functions depends on its logic.
- Functions: these are the only way to interact with the contract; functions are triggered by users and act in the same way as transactions.

A smart contract usually contains events, which are used to track the execution of transactions. Events could also be used as cheap storage because they are stored in the log bloom; however, they are not accessible by a running contract. There is also the Abi-definition: a Json based representation that is used by web3 to implement the interface and interact with the smart contract.

The usual interaction with a smart contract is illustrated in Figure 2.6, and because of the immutability of the contracts after deployment, they usually have only three stages in their life cycle: Init (deploy) when the smart contract is deployed by a transaction, live (read, change state) when the smart contract is running on the network and all nodes are able to interact with it, and the last optional stage, destruct (kill), which could sometimes be used as a back door when a contract has a large unsolved bug. Meanwhile, there are contracts that could have the ability to be in a stopped or modify stage; however, to achieve that, they require a complex design pattern and a reliable voting system to ensure the governance between the participants.

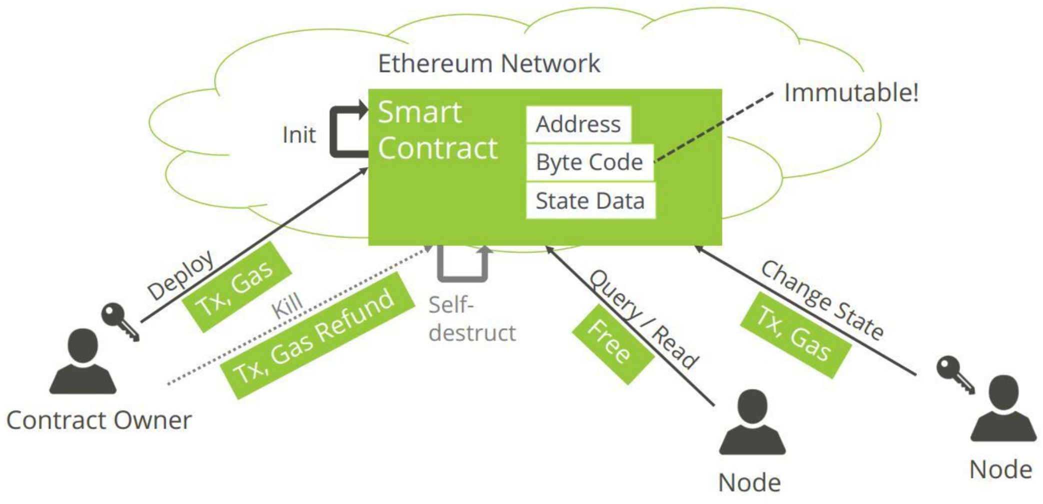


Figure 2.6: The possible interaction with a smart contract and its life cycle [BOOESC18]

Vulnerabilities

Smart contracts on the Ethereum platform could interact and send value between each other; but there is no direct interaction with the outside world, and a function cannot invoke itself without an outside human trigger. This problem adds additional limitation to the possible use cases. As mentioned above, a smart contract also inherits the characteristic of the blockchain, such as immutability, and there is thus no way of changing or modifying a contract after deployment, even if it has a large bug that could be exploited by hackers. To avoid such a bug, it is always preferable to use the principle of condition-effects-interaction for writing a smart contract.

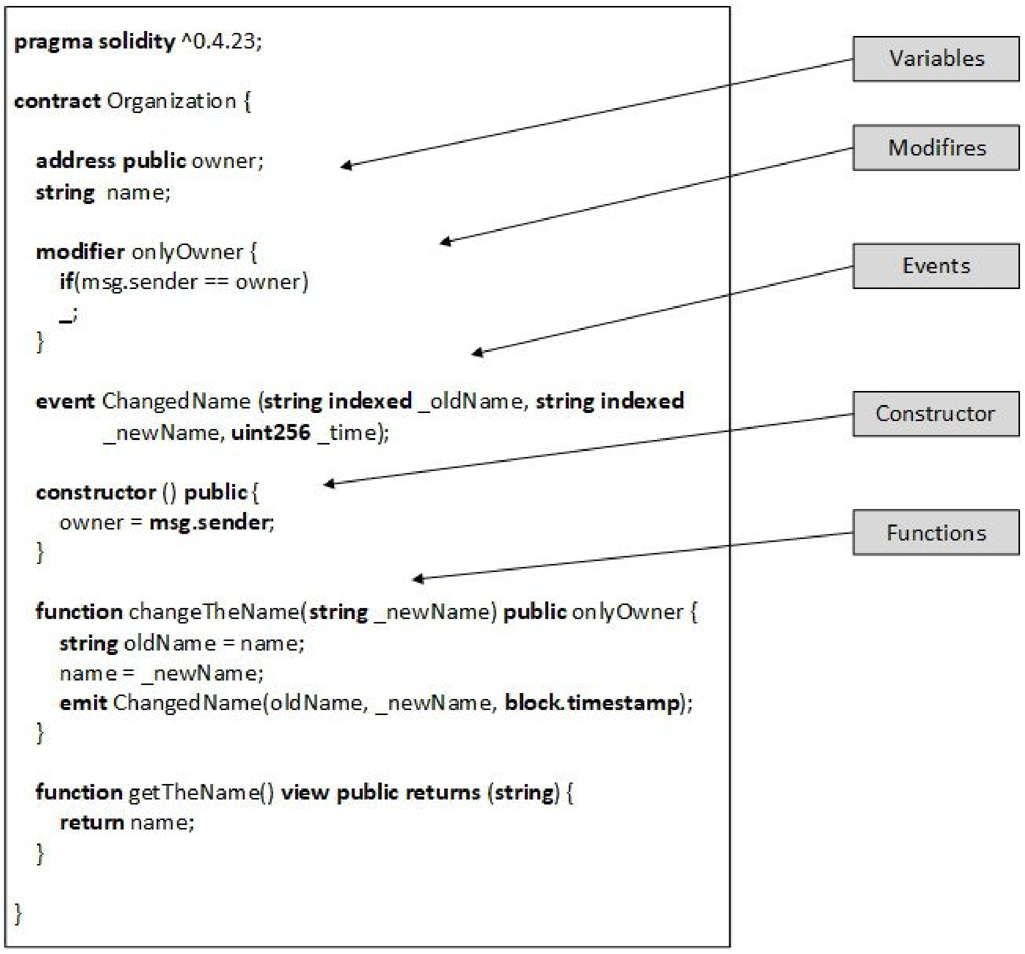
The deterministic characteristics of the network do not allow for the generation of a random number, which is useful in many use cases and games. There is also the readability problem that points to the unreadability of smart contracts by non-technical people, and it is one of the challenges that impedes smart contracts from expanding in daily life use cases.

Solidity

Solidity is one of the high-level programming languages for implementing smart contracts on the Ethereum network. Its JavaScript syntax with C-style data types makes it the most preferred and supported language on the platform; however, there are other languages, such as Serpent (a Python-like language) and LLL.

The Solidity `solc` compiler is used to compile the contract code into byte code, which can be executed with the EVM on every node in the network. `solc` has different versions — the latest version at the time of writing this thesis is 0.4.23.

The structure of a smart contract programmed in Solidity is shown in Figure 2.7. First, we declare the compiler version that we want to use, followed by the name of the contract, and in that code block, we declare the variables and then the modifiers, which represent a condition such as an If statement. Thereafter, we declare the events, then the constructor, which is the first function that runs when we deploy the contract, and at last, we declare and implement the methods. The Abi of this contract is also indicated in A.5



```
pragma solidity ^0.4.23;

contract Organization {

  address public owner;
  string  name;

  modifier onlyOwner {
    if(msg.sender == owner)
    _;
  }

  event ChangedName (string indexed _oldName, string indexed
        _newName, uint256 _time);

  constructor () public{
    owner = msg.sender;
  }

  function changeTheName(string _newName) public onlyOwner {
    string oldName = name;
    name = _newName;
    emit ChangedName(oldName, _newName, block.timestamp);
  }

  function getTheName() view public returns (string) {
    return name;
  }

}
```

Figure 2.7: A simple smart contract that show the structure of a program in Solidity [BOOESC18]

2.4.2 Decentralized Application

A DApp is an open-source application whose backend is run on a decentralized platform (in the case of Ethereum, it runs on the Ethereum P2P network).

We can classify DApps into two main categories [**GToDApps**]:

- type I: DApps that have their own blockchain and coin, such as Litecoin ,and Ethereum which has been introduced to resolve the need for type I DApps.
- type II: DApps that use the blockchain of a type I DApp.

In this thesis, we focus on type II DApps, such as all the programs[7] that are currently running on the Ethereum platform.

A DApp requires the comparison of one or more smart contracts and a user-friendly interface to build it. Furthermore, it usually performs the following steps:

1. Implementing the smart contracts, which will represent the logical and computational part of the application, using solidity or other languages supported by the EVM.
2. Compiling the contracts to generate the byte code and the Abi definition.
3. Deploying the contracts on the Ethereum blockchain.
4. Using the Abi definition, combined with web3 and a programming language, to build a user-friendly interface that could interact with the contracts. Web3 supports multiple programming languages, such as web.js for JavaScript, web3.j for Java and web3.py for python.

Smart contracts are the main but not the only part of DApps. They represent the computing functionality of a DApp in the logical component; however, there are also two other components, namely, Whisper for the communication layer and Swarm for static data storage.

- Whisper is a communication protocol between DApps when the consensus on the blockchain is not required because there is no change of state. Whisper provides a one-to-one, one-to-N and one-to-all channel.
- Swarm is a decentralized P2P storage platform, which is not controlled by any central authority. However, it is still under development, and most developers are using IPFS, which has a similar functionality.

Differences

Web applications and DApps share the same presentation layer that represents the front end; however, the main differences are in the backend on the other layers. Figure 2.8 illustrates the different between web applications and DApps.

Similarly to other application types, DApps also have some advantages and disadvantages [**BBp**].

- Advantages of DApps:
  - Decentralized applications are fault-tolerant because there is no single point of failure, and they are running on the blockchain.
  - There is no central authority that could possibly cheat the users and manipulate the data; therefore, it is easy for users to trust the application.
  - Data and records are permanently stored, and they are reachable by any user.
- Disadvantages of DApps:
  - Since smart contracts are immutable, updating and modifying DApps are difficult.

[7]https://www.stateofthedapps.com/

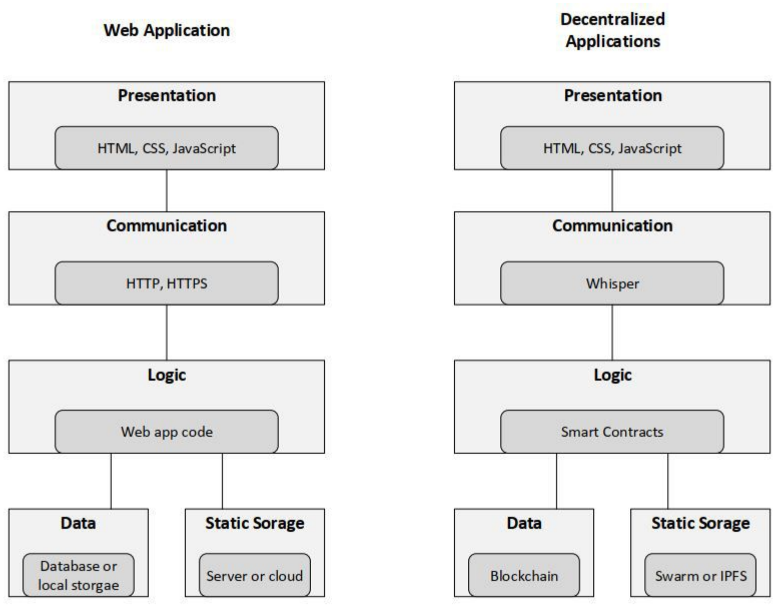


Figure 2.8: The difference between a web app and a DApp [BM17]

– Some applications require the verification of a user's identity; however, there is no central authority to verify this identity.

– Decentralized applications inherit the same problems as the blockchain, such as scalability and privacy.

– Regulations of some applications.

2.5 Donation Process

Donations involve a more complicated process than a simple sequence that flows from a donor to a charity to people in need. Furthermore, there are several schemas of the donation process, and each charity has its own structure, model and work flow, based on the types of problems they want to solve, the bank rules and government regulation. Nevertheless, crowdfunding is the most used and known schema.

Crowdfunding is simply raising small amounts of money from a large number of people to accomplish a significant project that has specific goals, an already defined structure and a time line, and when it is done, it will often affect the people who invested in it and those for whom the funds were raised. Crowdfunding projects do not necessarily have a charitable nature. For example, one of the largest crowdfunding projects in recent history is the EOS[8] platform; through its initial coin offering (ICO), which has similar characteristics to crowdfunding, it has raised about 4 billion U.S. dollars to develop a platform for DApps [EOSico18]. Crowdfunding schemas unfortunately cannot be customize to suit a wide variety of projects such as those that do not have a specific time

[8]https://eos.io/

line or that have a variety of goals and require irregular amounts of money at irregular times, or even projects that need money on a daily base to survive and achieve their own new challenges and long-term goals.

In this thesis, we focus on another model of the donation process illustrated in Figure 2.9. In this model, a charity has several active projects running in different countries around the world, and as is typical in first- and second-world countries, donors give their donations to a charity by sending a specific amount of money to the charity's bank account as well as providing their information, which will be stored in the charity's database. After receiving the money, the charity will assess its projects to determine which of them need the money the most and to verify their progress. Only then will it transfer or split the provided money between these projects; in this step, there may be a need for the traditional banking system, and the money will be transferred as transactions between two banks in two different countries. When a project receives its required amount of money, it will either send it to a provider company, which provides food and tools, or it will withdraw it and give it directly to the people or stores, since a banking system is lacking in most developing countries, where neither stores nor small companies have bank accounts. Furthermore, the project should manage the information between stores, companies and needy people, and they should also provide all that information to the main charity, which in turn needs to verify and store it and decide whether it will continue supporting that particular project. However, this schema has several issues, which we will discuss in the fifth chapter.

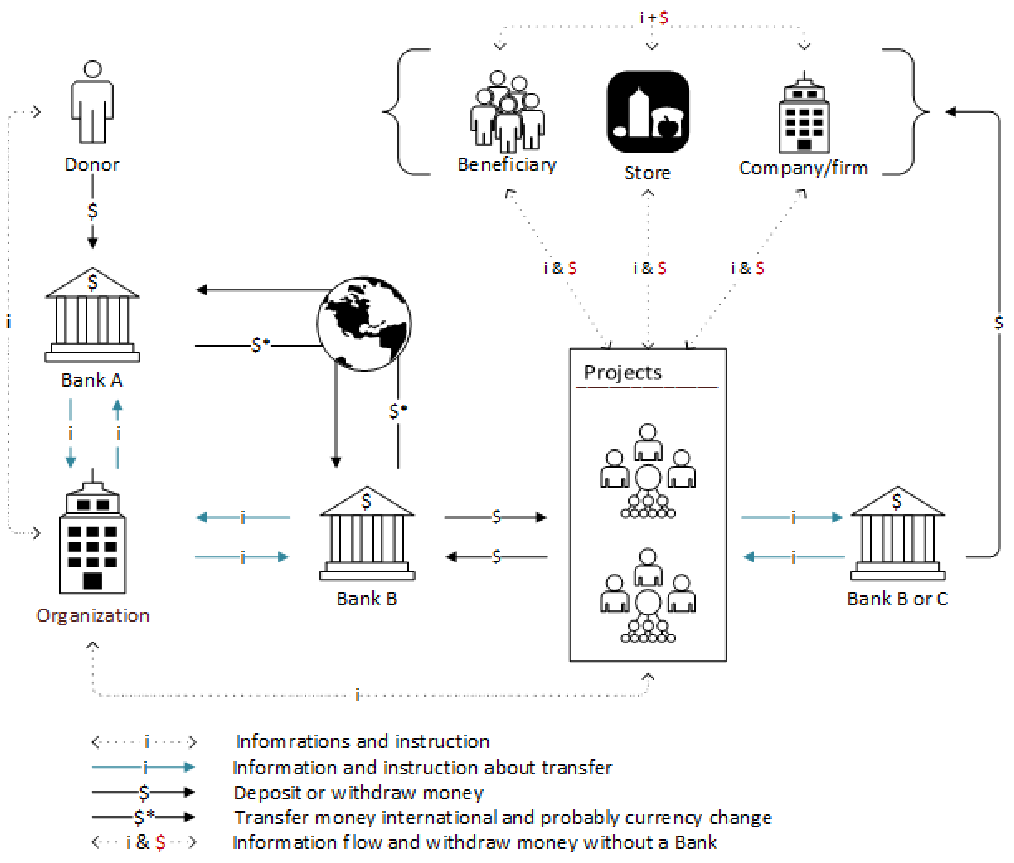

i + $
Beneficiary
Store
Company/firm
Donor
$
Bank A
i
i
i
$
$*
$*
i & $
i & $
i & $
Projects
$
Organization
i
i
Bank B
$
$
i
i
Bank B or C
i
Infomrations and instruction
Information and instruction about transfer
Deposit or withdraw money
Transfer money international and probably currency change
Information flow and withdraw money without a Bank

Figure 2.9: A model of a donation process shows the flow of donation from donors to people

3 Motivation and Related Work

3.1 Motivation

Throughout history, money as well as food, clothing and tools have been donated to disadvantaged groups, and donations comprise one of the daily and important activities that improve society and our world. Most people did not know exactly how to help, and donations were thus often organized through the Church; however, nowadays, there are many nonprofit, charitable organizations that collect all kinds of donations daily to support a wide variety of needy people such as children, sufferers of various diseases, homeless people and disadvantaged people around the world. Their goals are to be intermediaries between donors and the underprivileged to make the donation easier, superior and managed better.

The following are some points that may explain why people volunteer at or donate to charities:

1. To fulfill their desire to help others.
2. To experience more pleasure[1]
3. To support their beliefs and faith, and to bring more meaning to their lives.
4. To protect the local community.
5. To reduce their taxes.

Despite all of the advantages of donations and the substantial need for them, particularly nowadays, when the planet is suffering from warming, desertification, and more diseases and wars, the amount of giving is continuously decreasing. In Canada, for example, the amount of charitable donations fell to its lowest level in 10 years in 2016 [Dha16]; however, not just occurring in Canada but around the world, trust in charities and the amount of money raised from people for those charities is in gradual decline. Furthermore, most people do not know which charities they should trust or how to ensure that their donations go directly to addressing the cause related to their donations rather than to charities' operating expenses. This loss of confidence is one of the reasons that many people avoid charities and try to reach the needy directly, without an intermediary; however, this is not always easy to accomplish. On the other side, some people do not prefer to donate at all, even though they would like to help others.

For example, people in the UK would give an extra £665 million a year to charity if organizations provided more information about the issues they care about, such as how and where their donations are spent and evidence of their impact [UKcharity13]. Furthermore, in the past, studies have shown that the main driver of trust and confidence in nonprofits is their ability to satisfy public demands for transparency about their activities [RFP12] [TCC16]. Therefore, a fully transparent organization

[1] https://goo.gl/umfzdt

with a decentralized governance in its infrastructure could increase trust and efficiency in the way in which donations and charities work. This transparency and decentralized governance could only be applied by building applications or systems with no central authority that could cheat or manipulate the information, as well as applications that could enforce the rules on all participants, and that is exactly the environment that blockchain technology and smart contracts can provide.

In the next chapters, we will discuss and try to solve some of the issues that charities are facing in their work, especially the transparency issue, along with some other issues in the second schema 2.9. We will also implement a prototype that demonstrates how the blockchain and smart contracts could help to make the donation process more transparent and efficient for both donors and charities.

3.2 Related Works

The idea of using the Ethereum platform for donation is not new. Meanwhile, some charities and developers have noticed the benefits of using the blockchain and smart contracts, such as transparency, low fees and rule enforcing. However, most of the projects and DApps that are attempting to improve the donation process, depending on transparency rules and the efficiency of nonprofit organizations, are based on the crowdfunding model.

Alice[2] is a platform that brings transparency to social funding through blockchain technology [MW17], and it is one of a few platforms that are already using smart contracts and Ethereum to raise money for charity-related projects. The idea of Alice is to allow organizations, small charities and even groups of people to create and run crowdfunding projects on the platform, thereby increasing the transparency of the projects because all transactions are visible, verified and available to donors at any time. Furthermore, the running projects should follow the defined rules, which are enforced by the smart contracts.

The main rule that is enforced on all projects is that charities on Alice only receive the donations if they achieve their goals, which should be listed before the project is launched. The three main parties in all projects, as illustrated in Figure 3.1 are the charity, donors and validators. Donors donate to a project of their choice using the Alice token, which is a bank-based stable token, meaning that for each Alice token in the platform, there is a fixed amount of money in the account of the Alice foundation. The charity that owns a specific project can withdraw Alice tokens from a smart contract and then exchange them for fiat money only when the goals of this project are complete and verified by the validators; however, the problems that the Alice platform faces are how to choose the validators and why people should trust them and accept their decisions. Alice is still working on this problem to find a mechanism that could help to choose validators; however, it is already running some successful projects, such as London Street Impact, with the goals of helping homeless people in London by providing personal support to each person. For that particular project, the validator is the mayor of London; this could be ideal for small, internal projects, but it is still not optimal.

[2]https://alice.si/

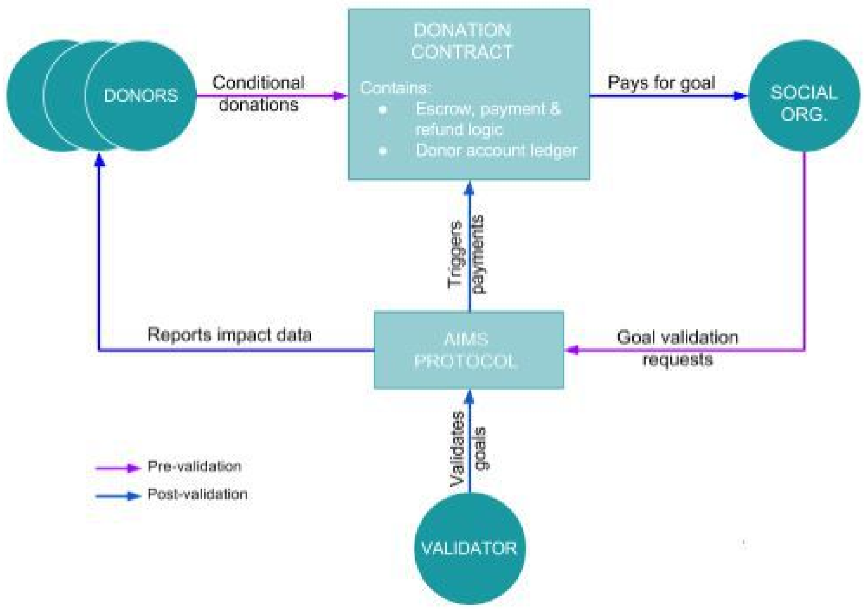


Figure 3.1: A figure describe the idea and different parties on the Alice platform [MW17]

The idea of a decentralized autonomous charity (DAC)[3] is another interesting subject that was proposed later. The idea was to run a charity as a decentralized autonomous organization, which is a new decentralized business model to organize and manage enterprises through smart contracts and some governance protocols. In a DAC, people can suggest and raise money for their projects from the community, and the following steps explain a simple process:

1. The charity receives a donation (in DAI, which is a stable token), and the donation will be locked until a fundraiser's goal is met.
2. Each donation triggers the creation of two tokens. The first token is the title token, and it represents the value of the donation given. The second token is a voting token, which grants the donor a vote in the governance.
3. Once the amount of money is reached, the title tokens begin to be distributed to projects according to the voting that has taken place by the governance model.
4. A project receives title tokens and then decides how to allocate them.
5. The last owners of the tokens trade them for DAI from the charity's DAI reserves.

The problem that such a system should solve is the governance and the weighted voting model. This kind of voting could lead to the elimination of many important projects that may be not interesting. However, the progress is stopped with the implementation of a proof of concept for a part of the suggested features without any governance model, and the developers are now trying to use their experience and that schema in the supply chain management.

[3]https://devpost.com/software/the-dac

Finally, many important nonprofit organizations find in the blockchain a good solution for some of their problems. For example, UNICEF, which started accepting Ether as a donation type on its multi-signature public wallet, initiated a project called Game Chaingers, whose goal is to raise money by mining Ethereum. The charity wanted gamers to lend their graphics cards and the processing power of their computers to mine Ether for Syrian children[4]. Furthermore, World Food Program (WFP)[5] — the world's largest humanitarian agency fighting hunger worldwide — is already successfully running a project for 10,000 refugees in Jordan. Those refugees are now able to pay for their food in a process that was recorded on the Ethereum blockchain. Working with the WFP, another similar blockchain-based tracking system is going to run soon in Tunisia for the school meals program.

[4]https://www.theguardian.com/global-development/2018/feb/06/unicef-recruits-gamers-mine-ethereum-aid-syrian-children

[5]http://innovation.wfp.org/project/building-blocks

4 Improvement the Work of Transparency International Deutschland

4.1 Transparency

Before we delve into the problems and the presented solutions, we need to understand the meaning and role of transparency and why it is an important requirement in every organization.

"Transparency is the perceived quality of intentionally shared information from a sender." [ST14]

As the definition explains, transparency is about openly sharing all relevant information, including how and when the actions are performed, and explaining decisions.
However, transparency is still a complicated term; therefore, for better understanding, we will separate it into three primary dimensions: disclosure, clarity and accuracy [ST14]. Disclosure refers to the accessibility and availability of the needed information; it implies that the information should be openly shared and accessible to all different parties. Clarity refers to the level of liquidity and comprehensibility of information; it focuses more on transferring information to the receiver in a simple and understandable way. Finally, accuracy, which is the most significant dimension to achieve, refers to the reliability of the shared information. Even if disclosure and clarity are achieved, without a certain level of accuracy, they are meaningless. The level of accuracy can also increase through more interactions with stakeholders and through the use of original data, papers and documents instead of just copies.

Transparency is important because it increases the trust in organizations, and it is always a first step to establish and repair trust in relationships with the stakeholders. In several studies, transparency has been considered to be an important dimension of the trustworthiness perception, which leads to trust; however, recent research indicates that transparency is associated with trustworthy perceptions, as illustrated in Figure 4.1. In both cases, transparency is the first stage of a trustworthy organizational structure.

The lack of transparency in some charities is one of the most important factors that explain why people do not donate, and it is one of the main reasons that public trust and confidence in charities has fallen to a low level. Therefore, every charity increase the transparency of its work nowadays by publishing information and involving stakeholders in the process in hopes that this will help to overcome a lack of confidence and will reduce the number of scams, thereby preventing the wasting of money on non-useful projects or even donations to the wrong people, which is a challenge that many people are currently facing.

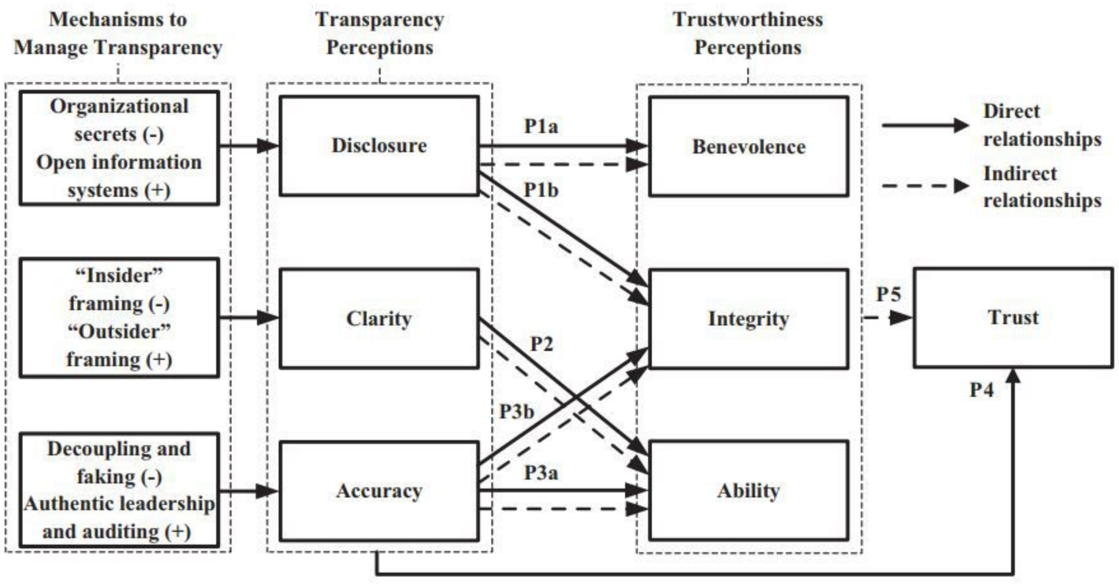

Mechanisms to Manage Transparency
Transparency Perceptions
Trustworthiness Perceptions
Organizational secrets (-) Open information systems (+)
"Insider" framing (-) "Outsider" framing (+)
Decoupling and faking (-) Authentic leadership and auditing (+)
Disclosure
Clarity
Accuracy
Benevolence
Integrity
Ability
Trust
P1a
P1b
P2
P3a
P3b
P4
P5
Direct relationships
Indirect relationships

Figure 4.1: Conceptual model of mechanisms to manage transparency and the association between transparency, trustworthiness, and trust [ST14]

4.2 Transparency International

Due to the importance of transparency in society and its main role in the fight against corruption, and since social organizations are not transparent enough about their work and impact, Transparency International was founded in 1993. Its nonprofit purpose is to take action to combat global corruption and prevent criminal activities arising from corruption.

Transparency International Deutschland (TID) is the national chapter of Transparency International, and it was started in 2010 with a focus on the basic principles, which are integrity, accountability, transparency and participation of civil society. To accomplish its goals, TID set 10 information points that any civil organization should publish to be a part of the fight against corruption and to increase its stakeholders' trust. The 10 points are as follows [Ziv16]:

1. Name, address and year of establishment.
2. Complete statute and statements about the organization's aims.
3. Information about tax concession.
4. Name and function of the main decision makers.
5. Annual report.
6. Structure of authority.
7. Information about the source of funding.
8. Information about the application for funding.
9. Affiliated third parties.

10. Names of persons whose annual paying account for more than 10% of the annual budget.

After checking some registered organizations, it was clear that some of these points could overlap, especially the seventh and eighth points.

By using those 10 points, TID hopes to ensure disclosure and the clarity of information, and it hopes to make the work of civil organizations transparent for people in a structured way. Since checking the accuracy of information is not an easy mission for several reasons, TID does not guarantee the correctness of information; however, they work hard to check the accuracy of the information, and they support any external report that can prove any fraud, which could count as a criminal activity.

4.3 Problem Statement

The work of TID is riddled with several problems. The first problem is the process of conforming and managing the registration of any organization that wants to commit to the agreements, this process has a low level of transparency for an external donor. Second, the roles of external interaction with the public are low and not clearly defined. Third, the information of the registered organizations could be manipulated or changed at any time after a successful registration. Transparency International Deutschland tries to prevent this third problem with a random test at a random point in time; however, the solution is not optimal and requires much effort to compare the new information with the old, existing data, which could also be manipulated if they are stored in an unsafe environment; moreover, this comparison is impossible for people who do not have the previous version of the published information or do not have the necessary knowledge or software for that.

To solve the mentioned problems, the Ethereum blockchain and smart contracts, with their significant benefits of transparency and immutability, seem to be the right place to improve the work of TID and increase its level of transparency. This will occur by moving all information and decisions to the blockchain and by defining the roles of all participants and their tasks in a clear way using smart contracts, which are transparent and cannot be manipulated or changed.

4.4 Details and Implementation

For a better understanding of the implemented smart contract, and before we present our idea and the implementation of the prototype, we will use the following modelling schema to represent our contracts in an understandable way.

Annotation

The following Figure 4.2 is a graphic representation for two smart contracts, which can be found in A.4 .

- State variables represent the storage of the smart contract where all data are stored.

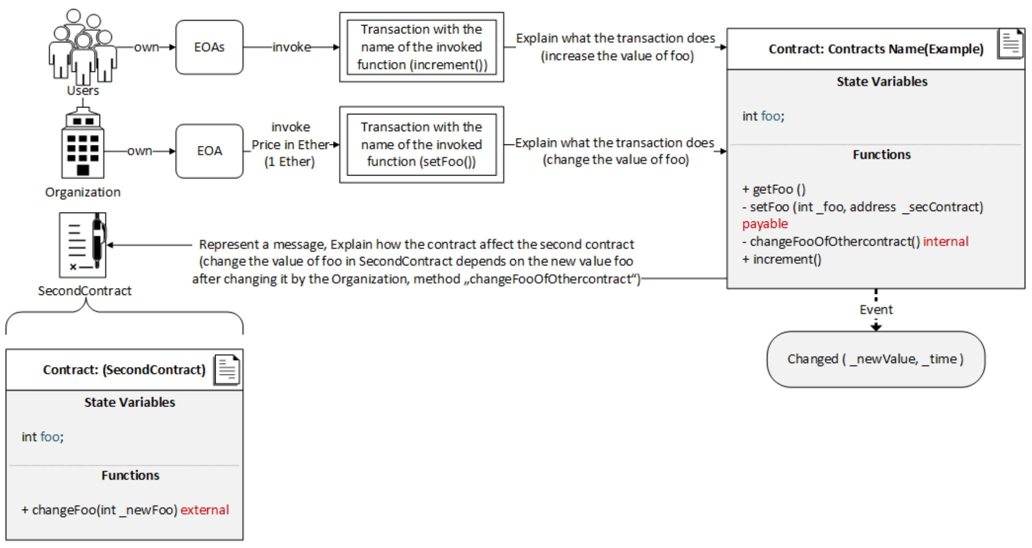

Users
own
EOAs
invoke
Transaction with the name of the invoked function (increment())
Explain what the transaction does (increase the value of foo)
Contract: Contracts Name(Example)
State Variables
int foo;
Functions
+ getFoo ()
- setFoo (int _foo, address _secContract) payable
- changeFooOfOthercontract() internal
+ increment()
Organization
own
EOA
invoke Price in Ether (1 Ether)
Transaction with the name of the invoked function (setFoo())
Explain what the transaction does (change the value of foo)
Represent a message, Explain how the contract affect the second contract (change the value of foo in SecondContract depends on the new value foo after changing it by the Organization, method „changeFooOfOthercontract")
SecondContract
Event
Changed (_newValue, _time)
Contract: (SecondContract)
State Variables
int foo;
Functions
+ changeFoo(int _newFoo) external

Figure 4.2: A graphic representation of two simple smart contracts A.4

- Functions are the methods that could be invoked by a transaction or messages. All the data on the blockchain are public and accessible to everyone; therefore, we will ignore the Get-functions in our future figures that describe smart contracts.
- The "-" behind the method's name means that the method can only be invoked by a specific actor — in our example, just by the organization, when the conditions are valid.
- The "+" means that the method can be invoked by any actor, users and organization, when the conditions are valid.
- Payable accepts the Ether that the function requires to be invoked.
- Internal methods are the methods that could only be invoked through another method in the same contract or its sub contract. Private methods are similar; however, they deny sub contracts the use of the method.
- External methods could be invoked from inside and outside the contract.

4.4.1 The General Idea

First, we should identify the main participants in the system. Those participants are the TID organization, represented by TID-manager-Account, which could be a multiple-signature wallet controlled by the decision makers in the TID-organization; the charities or civil organizations; and normal users, who could be any external people. Second, to prevent misleading or misunderstanding, it is important that any organization has a clear status that defines its situation in the system. Therefore, depending on several conditions, a civil organization could have only one of the following statuses shown in Figure 4.3.

To ensure transparency in the work of the TID organization, to prevent spam on the smart contract and to compel organizations to update their information, the registration process is completed in two steps. The first step involves invoking a registration request by sending a transaction with the

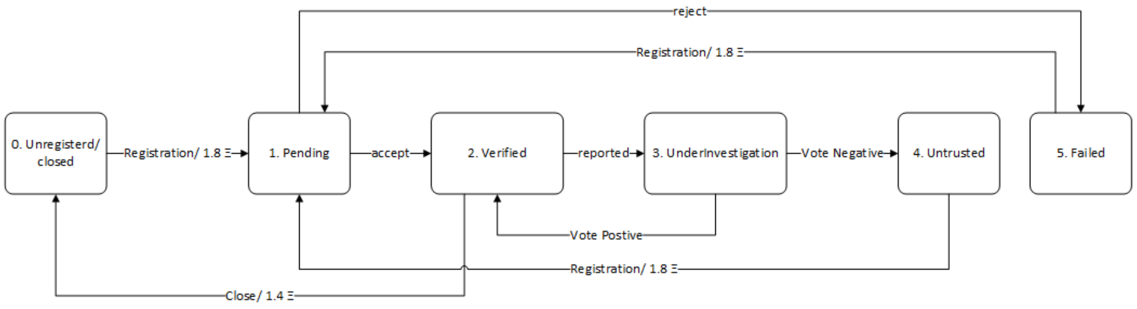


Figure 4.3: The possible status of an registered organization

necessary data, like name, office address and the web page where the 10 informations are stored, plus a certain amount of Ether to the contract. This request will be visible to all users, and the requester organization will be moved to the pending status. In the second step, the TID manager will check the provided information and consequently accept or reject the registration. The TID manager will also receive a certain amount of Ether as a reward for its work on processing the request. On the one hand, if the request is accepted, then the organization's status will be set to status 2, namely, verified, and the rest of the Ether will be locked in the contract as a fund to ensure that the organization will continue updating its information and maintain the correctness thereof. If, on the other hand, the registration is rejected, then the remaining amount of Ether will be sent back to the organization, and the status of the rejected organization in the contract will be changed to failed. At this point, feedback to the organization is necessary, but that could take place outside the smart contract to avoid additional costs. Since the work of TID is now transparent and each decision — accepting as well as rejecting — is stored publicly on the blockchain, and because the organizations set a fund, the first problem, along with a part of the third problem, is solved.

Storing a large amount of data on the blockchain is difficult, inefficient and even costly; therefore, an organization should use cryptographic hashing methods to obtain the hash value of the documents they want to publish, and only those hash values will be added to the smart contract and stored on the blockchain. That unique value will prove that the document existed at that time and prevent it from being manipulated or changed by the organization or any third party. Furthermore, each time a hash value is added or changed, it will be visible to all users through a new event that will be released to register the change. In this way, a part of the third problem is also solved, and the user can easily compare just the hash value of the published document from the organization's web page with the hash value that is saved in the smart contract to check whether the document is present or has been changed, modified or manipulated without an announcement. For example, if an organization wants to add its annual report of 2018, it should send a transaction to invoke a method that takes two parameters, namely, the year and the hash value of the annual report; if the transaction has been successfully mined, then the year and the hash value will be saved on the smart contract in a specific mapping for that organization, and the event modified-data will be released to register this.

To solve the second problem and enhance the solution of the third problem, we provide the user with an important and clearly defined role in the process of determining whether the organization is following the rules, publishing its information and updating it correctly. A user can report any registered organization by sending a transaction with the address of the reported organization and

Table 4.1: Voting power measured by two algorithm

Participant	**Weight**	**Absolute Banzhaf Index**	**Normalized Banzhaf Index**
TID manager	5	0.50000	0.33333
Organizations	5	0.50000	0.33333
Users	4	0.50000	0.33333

a certain amount of Ether; this Ether will be used as a fund to prevent users from spamming the contract and report organizations without a significant reason. However, the user will also be rewarded if his report is correct.

When an organization is reported, a new smart contract will be deployed, wherein the voting, rewards and calculations will be processed and controlled. The three actors in the new contract are the TID manager, organizations with a verified status and any users.

To ensure the fairness of the voting the points are shared between the actors like following:

- A TCS manager gives 5 points or 0 points.
- Organizations with status 2, grouped together, give between 0 to 5 points depending on the voting percentage.
- Users, grouped together, can give 0 to 4 points depending on the voting percentage.

The reported organization needs 8 points or more to pass the voting successfully and regain its verified status. The weighted vote is not optimal and could be improved; however, counting users and organizations as groups will guarantee that all three players have the same voting power and that there are no dictators or vetoes in the voting. The weight-vote of each group combined with its power is measured by two algorithms, as indicated in 4.1.

The reward and punching that will occur after the voting is determinate, as illustrated in Figure 4.4
.

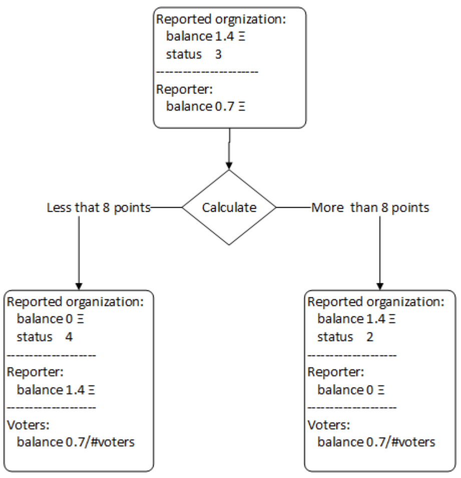
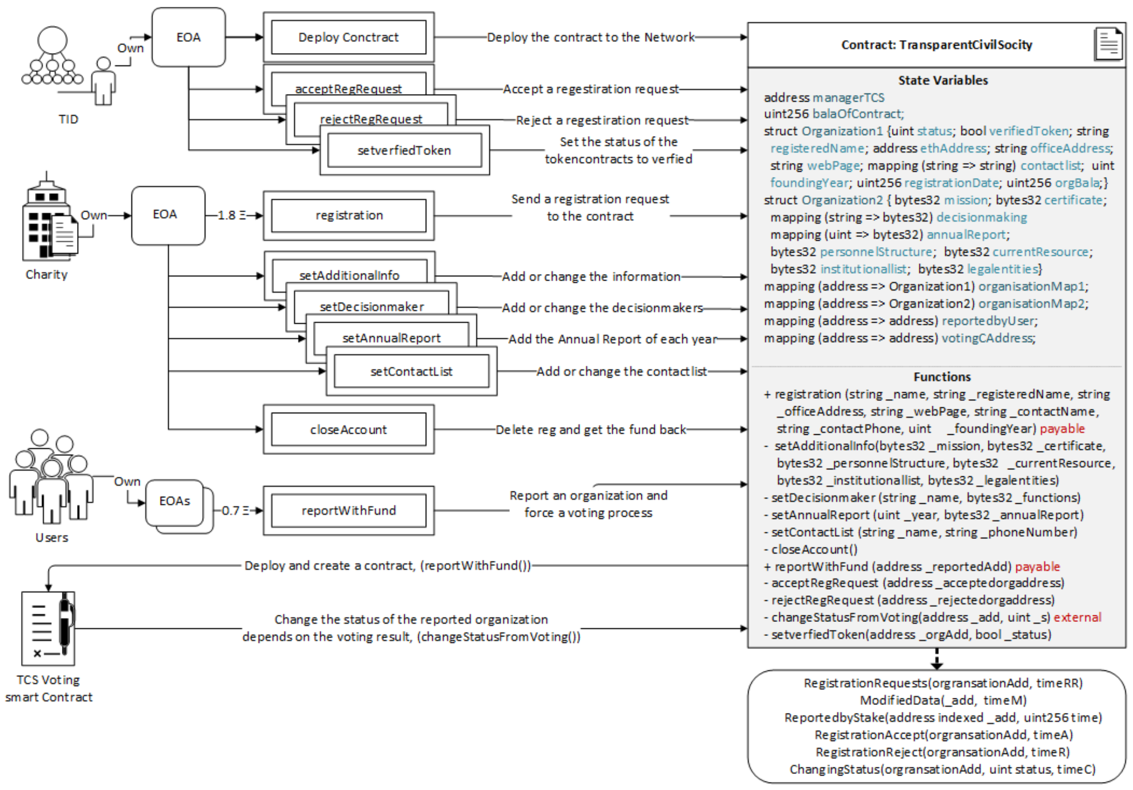


Figure 4.4: The rewards after the voting is complete

If the reported organization has failed in the voting, then it will lose its Ether as punishment for incorrectly published or not updated information. To prevent the burning of Ether, to motive any user to be a part of the voting process and to cover the voting fees, the voter will earn a small reward depending on how many voters voted, and the reporter will also receive his funds back plus half of the organization's balance.

Figure 4.5 depicts the smart contract and how the actors could interact with it; since all data are public and visible by any user, the get methods are hidden.


TID
Own
EOA
Deploy Conctract
Deploy the contract to the Network
acceptRegRequest
Accept a regestiration request
rejectRegRequest
Reject a regestiration request
setverfiedToken
Set the status of the tokencontracts to verfied
Charity
Own
EOA
1.8 Ξ
registration
Send a registration request to the contract
setAdditionalInfo
Add or change the information
setDecisionmaker
Add or change the decisionmakers
setAnnualReport
Add the Annual Report of each year
setContactList
Add or change the contactlist
closeAccount
Delete reg and get the fund back
Users
Own
EOAs
0.7 Ξ
reportWithFund
Report an organization and force a voting process
Deploy and create a contract, (reportWithFund())
Change the status of the reported organization depends on the voting result, (changeStatusFromVoting())
TCS Voting smart Contract
Contract: TransparentCivilSocity
State Variables
Functions

```
address managerTCS
uint256 balaOfContract;
struct Organization1 {uint status; bool verifiedToken; string
  registeredName; address ethAddress; string officeAddress;
  string webPage; mapping (string => string) contactlist;  uint
  foundingYear; uint256 registrationDate; uint256 orgBala;}
struct Organization2 { bytes32 mission; bytes32 certificate;
  mapping (string => bytes32) decisionmaking
  mapping (uint => bytes32) annualReport;
  bytes32 personnelStructure;  bytes32 currentResource;
  bytes32 institutionallist;  bytes32 legalentities}
mapping (address => Organization1) organisationMap1;
mapping (address => Organization2) organisationMap2;
mapping (address => address) reportedbyUser;
mapping (address => address) votingCAddress;
```

```
+ registration (string _name, string _registeredName, string
   _officeAddress, string _webPage, string _contactName,
   string _contactPhone, uint    _foundingYear) payable
-  setAdditionalInfo(bytes32 _mission, bytes32 _certificate,
   bytes32 _personnelStructure, bytes32   _currentResource,
   bytes32 _institutionallist, bytes32 _legalentities)
- setDecisionmaker (string _name, bytes32 _functions)
- setAnnualReport (uint _year, bytes32 _annualReport)
- setContactList (string _name, string _phoneNumber)
- closeAccount()
+ reportWithFund (address _reportedAdd) payable
- acceptRegRequest (address _acceptedorgaddress)
- rejectRegRequest (address _rejectedorgaddress)
- changeStatusFromVoting(address _add, uint _s) external
- setverfiedToken(address _orgAdd, bool _status)
```

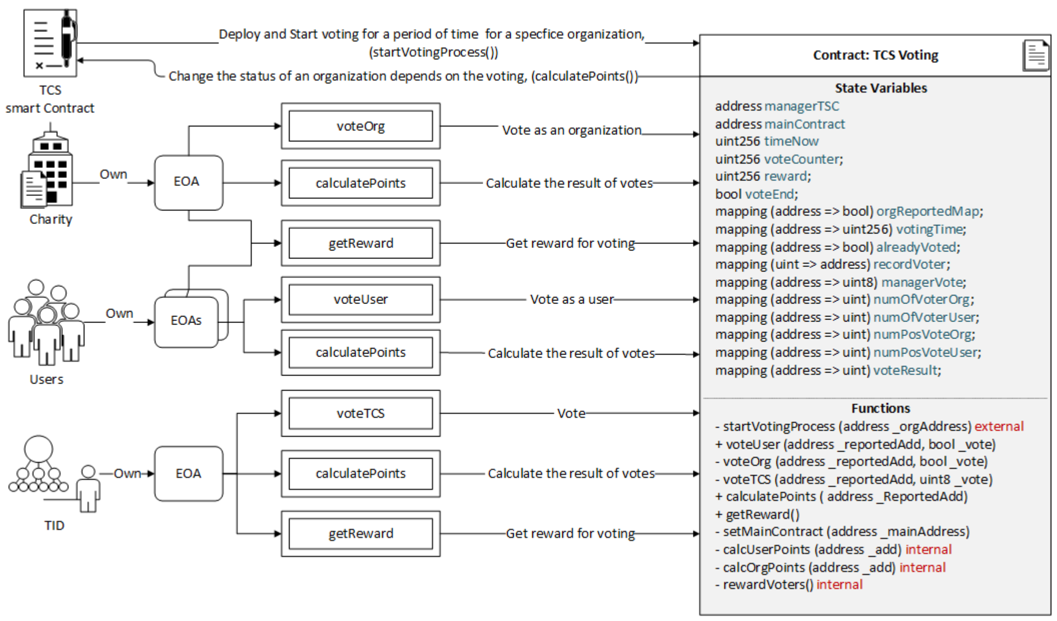
```
RegistrationRequests(orgransationAdd, timeRR)
ModifiedData(_add,  timeM)
ReportedbyStake(address indexed _add, uint256 time)
RegistrationAccept(orgransationAdd, timeA)
RegistrationReject(orgransationAdd, timeR)
ChangingStatus(orgransationAdd, uint status, timeC)
```

Figure 4.5: The TransparentCivilSociety contract that represent the main contract which is deployed by TID

Figure 4.6 displays the voting contract, which will be deployed by the main contract when a user reports an organization.

4.4.2 Address Book

When an organization starts using the Ethereum network to accept and process donations, it will have several Ethereum addresses for each project, and storing this information on a normal server is not favorable because of the risk of hacks, manipulation and unavailability even for a small period of time; therefore, there is the need to store those critical addresses on the blockchain. Storing those addresses on the blockchain in a smart contract will ensure that the information is saved safely and will always be available for any donor to be verified and used. Furthermore, an organization should store its addresses in specific categories, and it would thus not be able to deny that a particular address is related to it, nor would it be able to deny its role and task in its infrastructure.


TCS smart Contract
Deploy and Start voting for a period of time for a specfice organization, (startVotingProcess())
Change the status of an organization depends on the voting, (calculatePoints())
Charity
Own
EOA
voteOrg
calculatePoints
getReward
Vote as an organization
Calculate the result of votes
Get reward for voting
Users
Own
EOAs
voteUser
calculatePoints
Vote as a user
Calculate the result of votes
TID
Own
EOA
voteTCS
calculatePoints
getReward
Vote
Calculate the result of votes
Get reward for voting
Contract: TCS Voting

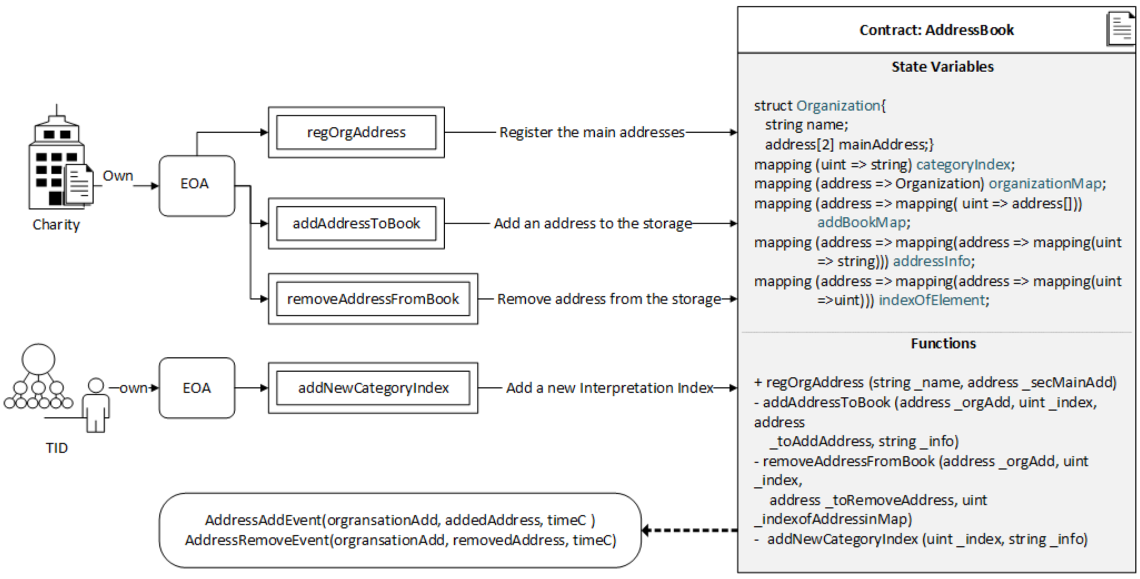
```
State Variables
address managerTSC
address mainContract
uint256 timeNow
uint256 voteCounter;
uint256 reward;
bool voteEnd;
mapping (address => bool) orgReportedMap;
mapping (address => uint256) votingTime;
mapping (address => bool) alreadyVoted;
mapping (uint => address) recordVoter;
mapping (address => uint8) managerVote;
mapping (address => uint) numOfVoterOrg;
mapping (address => uint) numOfVoterUser;
mapping (address => uint) numPosVoteOrg;
mapping (address => uint) numPosVoteUser;
mapping (address => uint) voteResult;

Functions
- startVotingProcess (address _orgAddress) external
+ voteUser (address _reportedAdd, bool _vote)
- voteOrg (address _reportedAdd, bool _vote)
- voteTCS (address _reportedAdd, uint8 _vote)
+ calculatePoints ( address _ReportedAdd)
+ getReward()
- setMainContract (address _mainAddress)
- calcUserPoints (address _add) internal
- calcOrgPoints (address _add) internal
- rewardVoters() internal
```

Figure 4.6: The voting contract that will be deployed for each reported Organization

The smart contract AddressBook shown in Figure 4.7 enables organizations to store their Ethereum addresses on the blockchain and to sort them into different categories depending on their functionality. The already initialized categories are 1. token contracts, 2. projects, 3. money-administration, 4. employees and 5. investors, and the TID organization could add a new category if needed. Participant organizations could add new addresses to any category with information about the addresses such as a name, a web page, a role or a hash value of a description. While they cannot change the information about the address, they could delete the address if there is no longer a need for it; however, any deleted address and its information will remain on the blockchain in the log bloom as proof of their existence, and the organization cannot make them disappear completely.

The AddressBook contract does not validate or check all the addresses of the registered organization. It only provides a simple way in which to store and categorize the addresses and ensure that they can only be managed by the organization that owns them. The donors will be able to check the addresses that are added by a specific organization, along with their respective categories, to be sure that their donations are not going to the wrong address. Furthermore, because the addresses are categorized, the donor will now have a better understanding of where his donation is going and the exact part of the project in which it had an impact.
The first category (token contracts) is related to the approach that we discuss in the next chapter.



```
struct Organization{
   string name;
   address[2] mainAddress;}
mapping (uint => string) categoryIndex;
mapping (address => Organization) organizationMap;
mapping (address => mapping( uint => address[]))
            addBookMap;
mapping (address => mapping(address => mapping(uint
            => string))) addressInfo;
mapping (address => mapping(address => mapping(uint
            =>uint))) indexOfElement;
```

```
+ regOrgAddress (string _name, address _secMainAdd)
- addAddressToBook (address _orgAdd, uint _index,
address
   _toAddAddress, string _info)
- removeAddressFromBook (address _orgAdd, uint
_index,
   address _toRemoveAddress, uint
_indexofAddressinMap)
-  addNewCategoryIndex (uint _index, string _info)
```

```
AddressAddEvent(orgransationAdd, addedAddress, timeC )
AddressRemoveEvent(orgransationAdd, removedAddress, timeC)
```

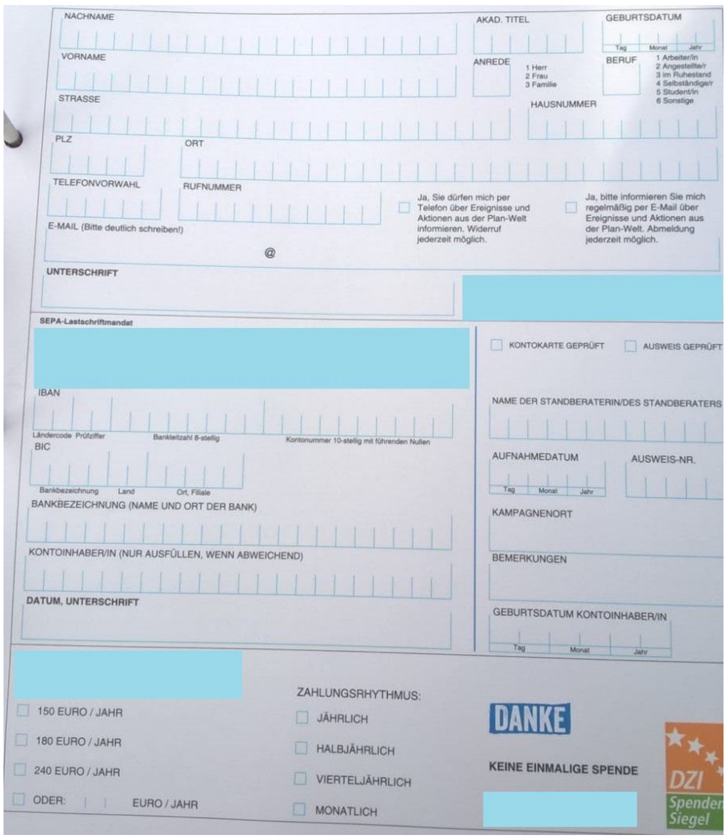
Figure 4.7: The AddressBook contract to store and categorize the addresses

5 Approach of a Donation Token

In this section, we will take a deeper look at the second schema (Figure 2.9), which we described in a previous chapter without highlighting its problems.

5.1 Problem Statement

The first and second problems that we can address are related to donors and the method of collecting donations. First, it is true that most donors want to use the advantage of their donations to reduce paid taxes, and while they do not want to be totally anonymous, they also do not necessarily want to share their information, specifically their banks accounts, credit cards and addresses, with charities, especially small and new ones. Figure 5.1 is a picture of a form that shows the needed information the donor should fill and sign.



NACHNAME

AKAD. TITEL

GEBURTSDATUM
Tag Monat Jahr

VORNAME

ANREDE
1 Herr
2 Frau
3 Familie

BERUF
1 Arbeiter/in
2 Angestellte/r
3 im Ruhestand
4 Selbständige/r
5 Student/in
6 Sonstige

STRASSE

HAUSNUMMER

PLZ

ORT

TELEFONVORWAHL

RUFNUMMER

☐ Ja, Sie dürfen mich per Telefon über Ereignisse und Aktionen aus der Plan-Welt informieren. Widerruf jederzeit möglich.

☐ Ja, bitte informieren Sie mich regelmäßig per E-Mail über Ereignisse und Aktionen aus der Plan-Welt. Abmeldung jederzeit möglich.

E-MAIL (Bitte deutlich schreiben!)
@

UNTERSCHRIFT

[redacted]

SEPA-Lastschriftmandat

[redacted]

IBAN
Ländercode Prüfziffer Bankleitzahl 8-stellig Kontonummer 10-stellig mit führenden Nullen

BIC
Bankbezeichnung Land Ort, Filiale

BANKBEZEICHNUNG (NAME UND ORT DER BANK)

KONTOINHABER/IN (NUR AUSFÜLLEN, WENN ABWEICHEND)

DATUM, UNTERSCHRIFT

☐ KONTOKARTE GEPRÜFT ☐ AUSWEIS GEPRÜFT

NAME DER STANDBERATERIN/DES STANDBERATERS

AUFNAHMEDATUM
Tag Monat Jahr

AUSWEIS-NR.

KAMPAGNENORT

BEMERKUNGEN

GEBURTSDATUM KONTOINHABER/IN
Tag Monat Jahr

[redacted]

☐ 150 EURO / JAHR
☐ 180 EURO / JAHR
☐ 240 EURO / JAHR
☐ ODER: EURO / JAHR

ZAHLUNGSRHYTHMUS:
☐ JÄHRLICH
☐ HALBJÄHRLICH
☐ VIERTELJÄHRLICH
☐ MONATLICH

DANKE

KEINE EINMALIGE SPENDE

[redacted]

DZI Spenden Siegel

Figure 5.1: A camera shot of a donation form that should be filled by donors and verified by workers of the organization.

Second, donors are donating directly to a charity and not to a specific project. Although some charities enable donors to donate directly to a particular project, this is not a preferred way for an important reason, which we will discuss later in this chapter.

Another significant problem that we clearly notice is that this model depends on third parties such as banks to deposit, withdraw and transfer money internationally between countries. However, using banks for this purpose is not an optimal solution for various reasons. First, there are high accumulated transaction fees and long periods of time needed to confirm transactions, especially when transferring money internationally. Second, there is a lack of transparency because data are often locked, and banks have strict rules regarding how to and who can access data and trace the transactions and the account's history. Third, when money is transferred from the main charity to one of its project in another country, the charity loses both control over and a view of the donated money. The fourth reason is the lack of banks and identity systems in developing countries and crisis areas, where not all people have the ability and possibility to identify themselves by a legal document, nor do they have some type of bank account; furthermore, the banks of most countries in crisis areas are completely under the control of unstable governments, and they could close or break at any moment. In many cases, the third and fourth problems combined could lead to corruption inside the charity itself, and this requires much of management's effort, such as verification and increased communications, to achieve an acceptable trust level between projects and the main charity.

In summary, the problems that we are trying to solve are as follows:

1. The given information by the donors.
2. Donations made directly to projects.
3. The dependency on banks
4. Transaction fees and long period of time to transfer.
5. Transparency and untraceable donation.
6. The significant effort to manage projects and information.
7. The leak of banks and governments.

5.2 Solutions and Discussion

Using the Ethereum blockchain to donate money has many benefits and could solve most of the above-mentioned issues.

By using Ether or tokens, donors are anonymous from the viewpoint of the charities because donors are using EOAs to donate; a charity knows only the value of the donation and the address of donors, but none of their private information, and that may solve the problems of privacy and the sharing of non-relevant information such as addresses and bank accounts with the charity. The donor does not lose the ability to reduce his taxes because he could easily prove his donation to the tax office by presenting the transaction hash and using the method of a digital signature. In this case, the information is naturally shared between the tax office and the donors, but no one else can access it. Furthermore, charities no longer have to care about that information, which comprises a large amount of data and needs much effort from charities to maintain the accuracy and security thereof.

Sending Ether or tokens as a donation not only solves the problem through anonymity, but it also greatly reduces the transactions fees and the needed period of time to complete all phases of the transactions. Figure 5.2 illustrates that even a simple a transaction between two entities using Visa is a complex, expensive process. In contrast, when it comes to Ethereum, the transfer of Ether, as depicted in Figure 5.2, is P2P between the two entities and does not go through banks; therefore, a currency exchange is not necessary, and this also reduces the high accumulated fees. In Ethereum, a normal transaction to send Ether requires approximately 21,000 gas, which was only about 0.05 USD on July 20, 2018, and for tokens, the cost is slightly higher because some relevant data should be added or changed. Furthermore, unlike other methods, the costs are independent from the transferred amount of Ether or tokens, and those transactions are also finalized in roughly 45 seconds after three mined blocks. Also, unlike usual bank transfers, processing those transactions does not require many days.

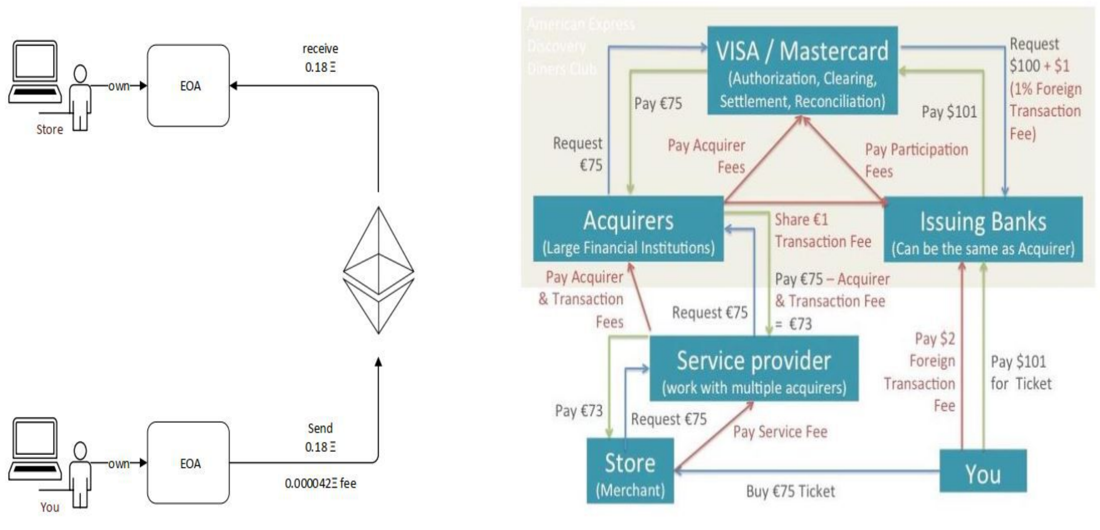


Figure 5.2: A transaction between a store and an individual using the Ethereum blockchain (left), and using a credit card (right) [Che14].

Since all transactions on the Ethereum blockchain are publicly available to all users, the transparency problem is almost solved; any user or organization can trace the account history of any specific project and check every transaction separately. Although transactions are public, there are still two slight problems: the anonymity of the addresses and the semi-transparent nature of the Ether and ERC20 tokens. Since they are not accompanied by a unique identifier, if Bob sends 10 Ether to account A and Alice also sends 10 Ether to the same account, then when account A sends 10 Ether to another account, neither Alice nor Bob will know whether her or his Ether or that of the other person has been transferred to that account. To solve the first problem, all organizations should publish each used Ethereum account and prove the ownership thereof. They should also describe its role in their infrastructure. Furthermore, to solve the second problem, there is the need for a more advanced token that is “non-fungible,” which means each coin is different from the others and has its own ID, and in our case, its own value, because there is no fixed amount to donate, while generating a certain number of tokens with a value of 1$ is much costlier and more difficult to manage than generating one token with a certain value. In that way, Bob and Alice can trace their tokens using the IDs and see the new owner whenever the token is transferred. This will make the transfer completely transparent, and since the accounts are categorized, the donor will know where his impact is in the project.

There are several problems with direct donations to projects, as described next. Charities do not prefer donations to go directly to a specific project. In addition to bureaucracy issues, there is another reason, namely, the unconfirmed need of each project at the time of each donation. However, the main reason is people’s thought process when they choose charities and projects. Donations could be described as “taste-based” rather than “needs-based,” and a study has found that people do not give to projects with the most urgent requirements; instead, they support and donate due to factors that mean something to them depending on their experience, passions or even self-interest [Bre10]. This problem could also be solved using smart contracts with a simple pattern. In this pattern, a donor could donate tokens to an organization. Those tokens will be locked in a smart contract—usually called a vault—until it becomes clear which projects are in need and ready to accept donations. At that time, the donor can unlock and transfer his tokens to the available projects; however, if donors for some reason did not transfer their tokens, then the organization will have access to those tokens after a defined period of time, at which point it could transfer them to one of the projects in a transparent way. In the suggested solution, tokens are locked in a vault contract for a period of time; however, this will lead to another issue, namely, volatility, which points to the instability price of Ether and other tokens. For example, the price of Ether on January 2018 was almost 1,310 USD, whereas one month later, it was worth 610 USD, which is less than half the former amount[1], and this instability leads to a dilemma in managing and planning projects. This problem is not exclusive to cryptocurrencies; it applies to varying foreign currency markets and donations in the form of stocks. However, the MakerDAO proposed a solution to this problem by introducing the DAI on December, 2017. In its Whitepaper, DAI claimed to be a collateral-backed cryptocurrency stablecoin whose value is stable relative to the U.S. Dollar [Mak17]; DAI is also an ERC20 token that runs on the Ethereum blockchain, which makes it a flexible token and compatible with all Ethereum wallets and exchanges that support the transfer and management of tokens.

The Ethereum network and most tokens running on it are not dependent on governments, banks or other financial institutions, and with smart contracts, organizations are able to control and trace their money even after transferring it to their projects. Furthermore, all that is required to use the Ethereum network is Internet access and an Ether account, which is easy to generate in a secure way. Also, despite the large size of the Ethereum blockchain, principles of light nodes and MetaMask currently allow users to interact with the network without the need to download and verify the whole blockchain. This important characteristic of Ethereum allows it to play a significant role in the problem of the lack of banks and government infrastructure in poor countries because any person is able to identify himself with an Ether address and receive or send money to any other person in any country around the world exactly as a bank account but cheaper, simpler and quicker.

[1]https://coinmarketcap.com/

There is also the ability to create multiple signature wallets, which can control parts of or all tokens and this is useful for grouping people into one unit as a family, and it simultaneously permits each family member a vote about how and when the tokens could be spent.

5.3 Approach

In this section, using the same notation that was used in the last chapter, we will describe the approach, which is constructed based on the solutions and discussion from the previous section.

The approach is implemented as a DApp that includes a simple interface and several connected smart contracts, which could be deployed and run on the Ethereum blockchain, each charity could deploy and manage those contracts individually by customizing its main addresses in the source code.

The General Idea

The main participants in this approach are the donors who donate Ether, the charity or organization, the charity's supported projects (each of these projects has its own smart contract) and the TID organization that has an optional functionality, which we will discuss later in the last section. Figure 5.3 provides a simple view of the participants in the approach and the relationships between the contracts.

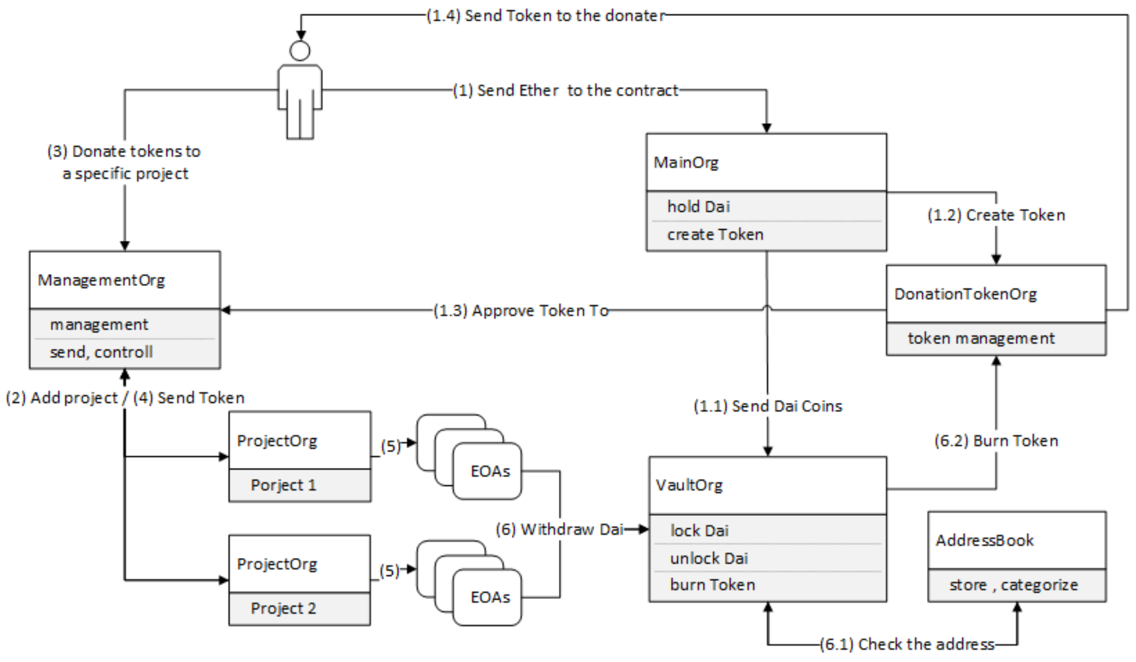


Figure 5.3: A description of the approach by showing the used smart contracts and the work flow between them and the donor

Here, we will describe the tasks of each smart contract and its role separately. However, first, a simple donation flow is explained below:

1. A donor donates a certain amount of Ether to the charity, and in exchange, he will receive a new generated token with a special ID and value. The Ether he donated will be converted to Dai and locked in another smart contract.
2. The main organization has several running projects, and it will register those that currently require donation in the management contract. These projects will remain registered until they receive the amount of money they require.
3. A donor can later choose one of the registered projects in the management contract and send his token to that project.
4. (Possible) if a donor missed the time given to him or forgot his private key, then the charity could transfer his token to one of the projects.
5. The project owner will split the tokens and transfer them to several addresses, which are already registered and categorized in the AddressBook contract. Addresses could be the addresses of employees, companies or even the people who will receive the donation, even though they might not have a bank account. However, the assignment of giving an identity to the people is left to the charities and not discussed in the thesis.
6. The new owner of tokens can unlock the Dai, withdraw it, and then exchange it for Ether or fiat money.

The approach involves six fixed smart contracts plus one contract ProjectOrg for each running project:

- DonationToken
- MainOrg
- ManagementOrg
- Vault
- AddressBook
- FaiToken (it simulate the work of an ERC20 Dai token)
- ProjectOrg

Donation Token Contract

The DonationToken smart contract depicted in Figure 5.4 describes the work of the used token that each donor receives after his donation, and it relates to three other contracts: MainOrg, ManagementOrg and VaultOrg.

Each DonationToken has its own ID and value based on the amount of the donation in DAI, and the token can just be minted by the MainOrg contract when someone donates a certain amount of Ether to the contract; this will ensure the value of the token. When a DonationToken is minted, the total supply will increase, and an event called Transfer will be triggered. This event shows that a

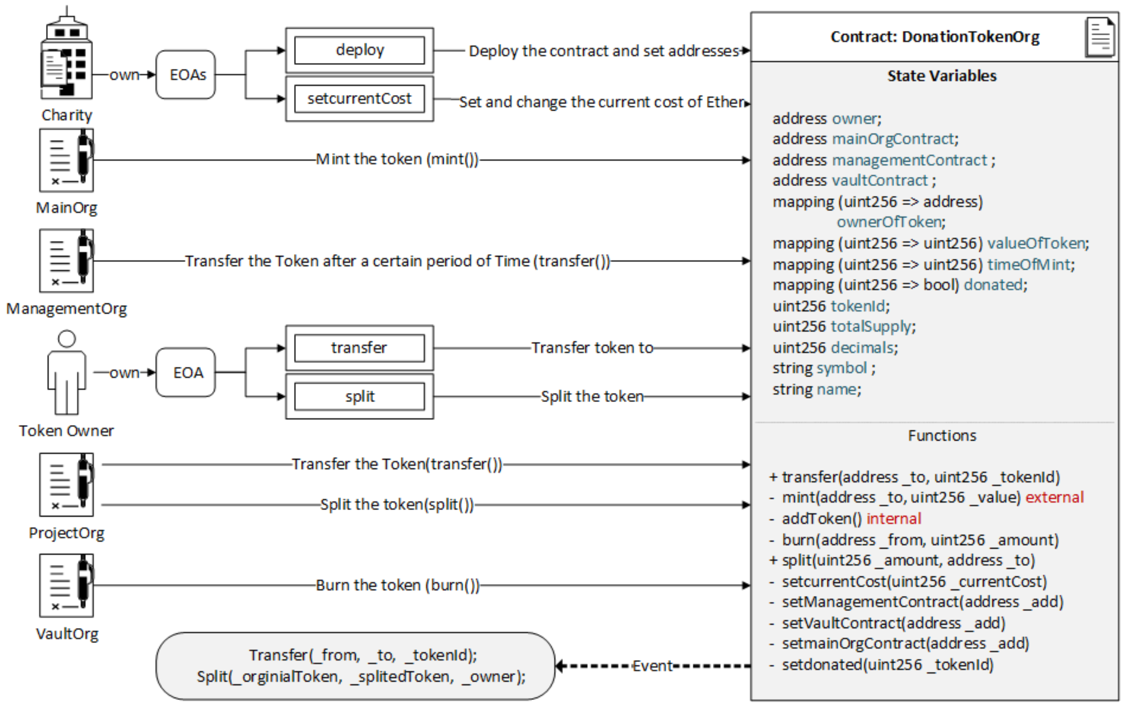

Charity
own
EOAs
deploy
setcurrentCost
Deploy the contract and set addresses
Set and change the current cost of Ether
MainOrg
Mint the token (mint())
ManagementOrg
Transfer the Token after a certain period of Time (transfer())
Token Owner
own
EOA
transfer
split
Transfer token to
Split the token
ProjectOrg
Transfer the Token(transfer())
Split the token(split())
VaultOrg
Burn the token (burn())
Transfer(_from, _to, _tokenId);
Split(_orginialToken, _splitedToken, _owner);
Event
Contract: DonationTokenOrg
State Variables
address owner;
address mainOrgContract;
address managementContract ;
address vaultContract ;
mapping (uint256 => address) ownerOfToken;
mapping (uint256 => uint256) valueOfToken;
mapping (uint256 => uint256) timeOfMint;
mapping (uint256 => bool) donated;
uint256 tokenId;
uint256 totalSupply;
uint256 decimals;
string symbol ;
string name;
Functions
+ transfer(address _to, uint256 _tokenId)
- mint(address _to, uint256 _value) external
- addToken() internal
- burn(address _from, uint256 _amount)
+ split(uint256 _amount, address _to)
- setcurrentCost(uint256 _currentCost)
- setManagementContract(address _add)
- setVaultContract(address _add)
- setmainOrgContract(address _add)
- setdonated(uint256 _tokenId)

Figure 5.4: The Donation-Token contract that generate and control the Donation-Token

token has been sent from the address 0x0000000000000000000000000000000000000000 to the donor's address, and it also displays the ID of that token. That information will be saved in the cheap memory log bloom, whereas the main data of the token, such as owner's address, the time and the value, will be saved in the storage. The token can be transferred between several accounts; however, the first transfer is combined with a condition to ensure that the receiver's address is a registered project in ManagementOrg and that the project needs donation at that time. Each transfer will trigger the event Transfer, which shows the sender, the receiver and the ID of the token.
As mentioned before, the donor has a period of time to choose one of the projects. If he misses the time line or forgot his private key, then the organization will be able to transfer his token to one of the projects. However, he can trace his token and see the project to which it has been donated.

Due to its fixed value, and to add flexibility to the sacrificed token, the owner could split the token into two tokens. By splitting it, a new token will be minted with a new ID and value, and only the value of the old token will be changed. Furthermore, an event split we be triggered to register the information about the split in the log bloom, so users can trace the history of the token.

By implementing token contract, such as in ERC20, there is usually a function called balanceOf(address owner), which returns the amount of tokens owned by the user. However, since our tokens are non-fungible and have different values, and to save the amount of additional necessarily gas required to save and store the ID of a user's tokens and its index in arrays, we decided not to implement such a function. Nevertheless, because all transfer and split events are saved in the log bloom, there is a possibility to see the tokens that are owned by any address through the following simple algorithm 5.1 that could be implemented in JavaScript or any other programming language that is supported by web3.

Listing 5.1 1 Pseudo-code could be implemented outside the blockchain to obtain a user's balance

```
//event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
//event Split(uint256 indexed _orginalToken, uint256 indexed _newSecToken, address indexed _owner);

function balanceOf(address _addr, uint fromBlock ,uint toBlock) {
        // a list that hold the Id of the tokens owned by the address
        list tokenOwnedBy ;
        for each Block in the range watch the events Transfer & Split
                if(_to == _addr){
                        add _tokenId to the list tokenOwnedBy
                }
                if(_owner == _add){
                        add _newSecToken to the list tokenOwnedBy
                }
                if(_from == _addr){
                        remove _tokenId from the list tokenOwnedBy
                }
        print tokenOwnedBy
}
```

Finally, when the token reaches its destination and has been converted back to DAI, it will be burned by setting its value to 0 and deleting the owner.

Vault Contract

The vault smart contract shown in Figure 5.5 is deployed by the organization, and its task is to ensure that the DAI, which is sent after a donation, is locked and that no one has access to it until the conditions are met.

Not any owner of the DonationToken can unlock and withdraw the DAI because there are conditions requiring that the owner is registered in the AddressBook contract by the organization itself. The reason for that is to be sure that organizations register and categorize their accounts as well as to ensure that organizations are informed about all accounts using the system. If the condition is met, then the owner should send his token to the vault where it will be burned. Therefore, the vault contract is connected to three contracts, which are the AddressBook, DonationtokenOrg and MainOrg.

MainOrg Contract

The MainOrg contract shown in Figure 5.6 represents a part of an organization's tasks, and it is owned by the organization, represented by its owner. Before the MainOrg can receive donations, the organization should fill the address with DAI tokens and set the current price of Ether. Thereafter, when a donor sends Ether to the contract, the MainOrg will calculate the value of the donation in Dai; then, it will connect to the DonationTokenOrg contract to mint a DonationToken with a new ID and the corresponding value, and it will also send the DAI tokens to the VaultOrg contract to

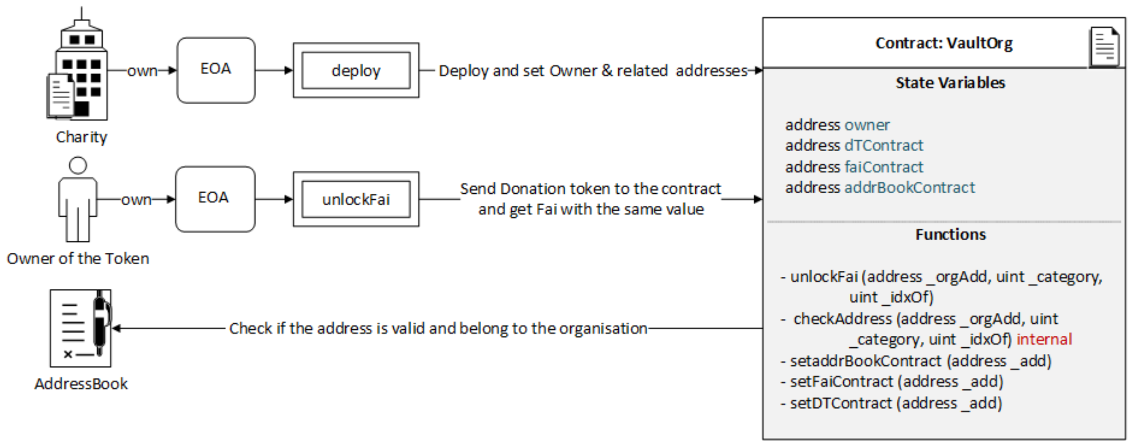

Charity
own
EOA
deploy
Deploy and set Owner & related addresses
Owner of the Token
own
EOA
unlockFai
Send Donation token to the contract and get Fai with the same value
AddressBook
Check if the address is valid and belong to the organisation
Contract: VaultOrg
State Variables
address owner
address dTContract
address faiContract
address addrBookContract
Functions
- unlockFai (address _orgAdd, uint _category, uint _idxOf)
- checkAddress (address _orgAdd, uint _category, uint _idxOf) internal
- setaddrBookContract (address _add)
- setFaiContract (address _add)
- setDTContract (address _add)

Figure 5.5: The vault contract where the Dai will be locked

lock them there. The other task of the MainOrg contract is adding projects, after confirming their needs, to the specific array in the ManagementOrg contract in order to allow donors to send their DonationToken to those projects.

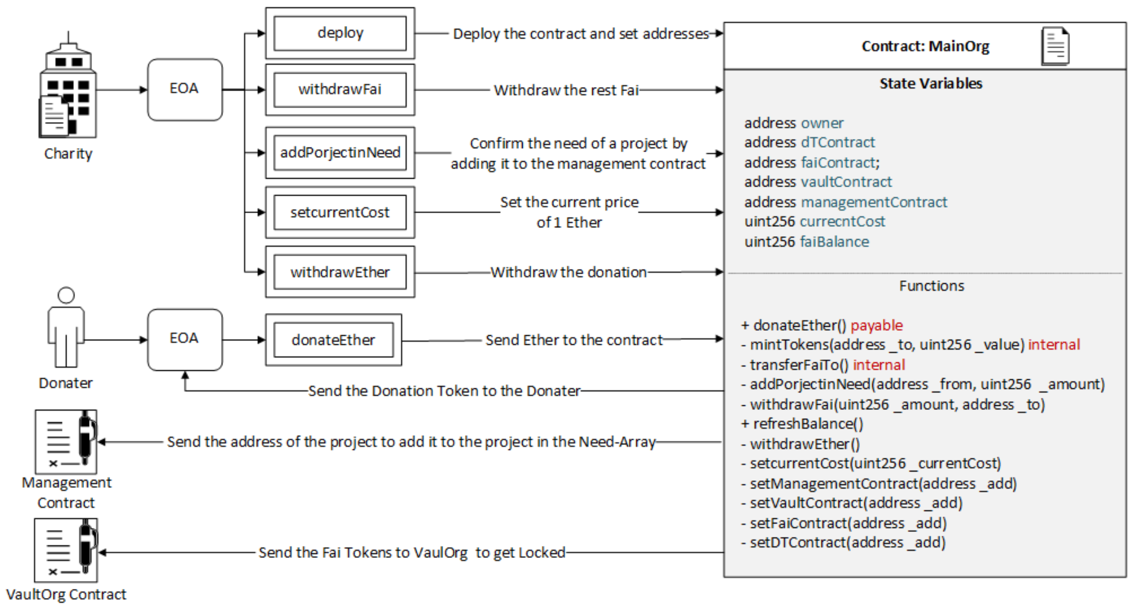

Charity
EOA
deploy
Deploy the contract and set addresses
withdrawFai
Withdraw the rest Fai
addPorjectinNeed
Confirm the need of a project by adding it to the management contract
setcurrentCost
Set the current price of 1 Ether
withdrawEther
Withdraw the donation
Donater
EOA
donateEther
Send Ether to the contract
Send the Donation Token to the Donater
Management Contract
Send the address of the project to add it to the project in the Need-Array
VaultOrg Contract
Send the Fai Tokens to VaulOrg to get Locked
Contract: MainOrg
State Variables
address owner
address dTContract
address faiContract;
address vaultContract
address managementContract
uint256 currecntCost
uint256 faiBalance
Functions
+ donateEther() payable
- mintTokens(address _to, uint256 _value) internal
- transferFaiTo() internal
- addPorjectinNeed(address _from, uint256 _amount)
- withdrawFai(uint256 _amount, address _to)
+ refreshBalance()
- withdrawEther()
- setcurrentCost(uint256 _currentCost)
- setManagementContract(address _add)
- setVaultContract(address _add)
- setFaiContract(address _add)
- setDTContract(address _add)

Figure 5.6: The MainOrg contract which accept the Ether donation and act like a start point

Management Contract

The ManagementOrg contract shown in Figure 5.7 completes the other part of an organization's tasks. It is also owned by the organization, and its first task is to store information about the projects in need, which are added to the array by the MainOrg contract or the owner. Through this contract, the projects could also be removed when they reach their goals. The ManagementOrg contract also has a function that allows donors to send their DonationToken to a particular project, and that function will also change the condition that the DonationtokenOrg contract uses to enable the token to be transferred directly between EOAs, Furthermore, only this contract has the ability to either transfer the token if the donor has not used it after a certain period of time or take the ownership of it if the project loses the trust and support of the organization.

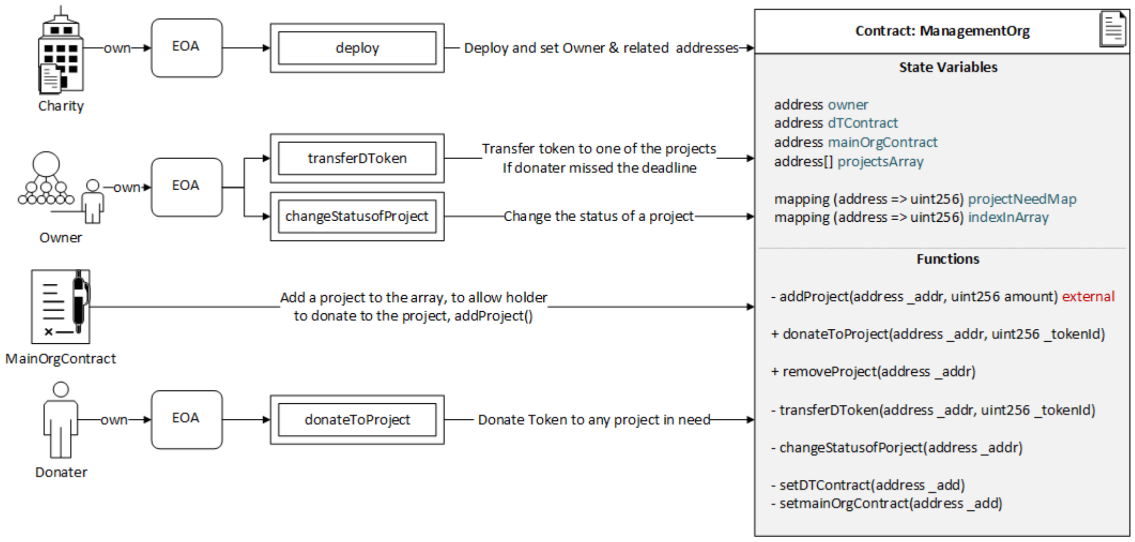

Charity
own
EOA
deploy
Deploy and set Owner & related addresses
Contract: ManagementOrg
State Variables
address owner
address dTContract
address mainOrgContract
address[] projectsArray
mapping (address => uint256) projectNeedMap
mapping (address => uint256) indexInArray
Functions
- addProject(address _addr, uint256 amount) external
+ donateToProject(address _addr, uint256 _tokenId)
+ removeProject(address _addr)
- transferDToken(address _addr, uint256 _tokenId)
- changeStatusofPorject(address _addr)
- setDTContract(address _add)
- setmainOrgContract(address _add)
Owner
own
EOA
transferDToken
Transfer token to one of the projects If donater missed the deadline
changeStatusofProject
Change the status of a project
MainOrgContract
Add a project to the array, to allow holder to donate to the project, addProject()
Donater
own
EOA
donateToProject
Donate Token to any project in need

Figure 5.7: The ManagementOrg contract where the projects in need are stored

Project Contract

The ProjectOrg contract shown in Figure 5.8 represents a simple project that is running and supported by the main organization. Each project has an owner and its own status, which is controlled and could be changed by the organization through the ManagementOrg contract. The project also has some variables, such as name, location and start date. The ProjectOrg contract is able to receive and split the Donation-Token and then transfer it to the EOA of the people who are a part of the project.

5.4 Connect the Approach with the First Use Case

As previously mentioned, one of the issues that stops the adoption of smart contracts in daily life use cases is the readability problem. Since smart contracts are new terms and written with a programming language, they are difficult for non-technical people to read and verify; for this reason, people need someone who can translate or guarantee the work of those smart contracts. Therefore, we could add an additional task to the work of TID to cover that issue, as indicated in Figure 5.9.

When an organization wants to use the suggested approach, it should first be registered and verified with status 2 in the TCS smart contract and the AddressBook contract. Thereafter, the organization should register all the deployed smart contracts of our approach under the category Token-Contracts in the AddressBook contract. Furthermore, the organization should provide a description as a document on the same web page where the 10 published points are located, and that document

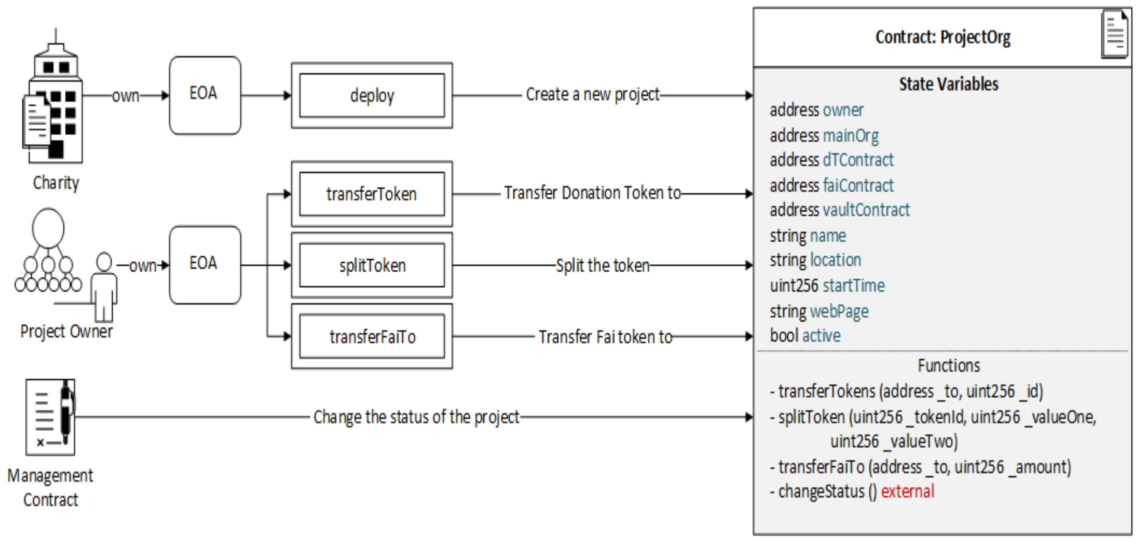


Figure 5.8: The Project contract will be deployed for each project

should explain, in simple language or diagrams, the working of the contracts and the relations between them and the organization. The hash of that document will be stored in the information field of each address in the first category. When those two steps are completed, TID could check those addresses, the smart contracts and the documents, and depending on the correctness of the descriptions, TID will change the variable verifiedToken in the charity's TID contract to true or false.

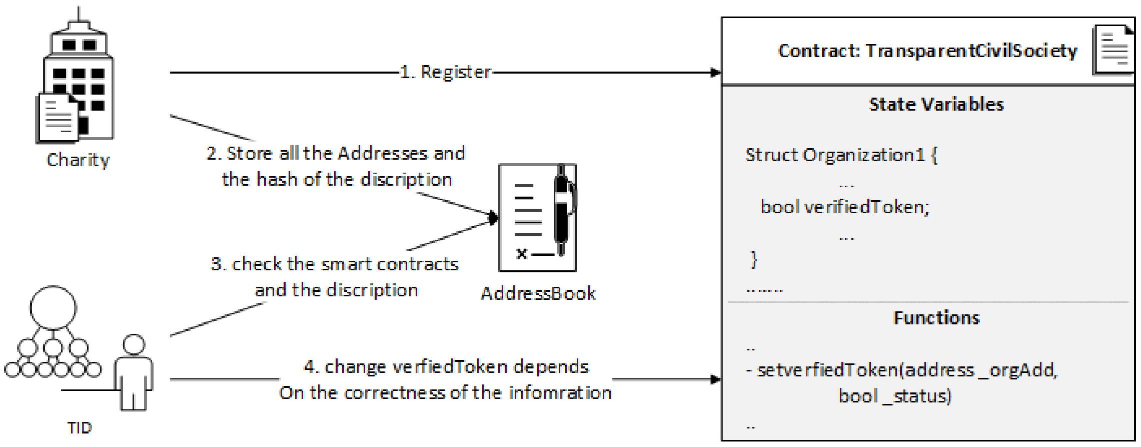

Charity
1. Register
2. Store all the Addresses and the hash of the discription
3. check the smart contracts and the discription
AddressBook
TID
4. change verfiedToken depends On the correctness of the infomration
Contract: TransparentCivilSociety
State Variables
Struct Organization1 {
...
bool verifiedToken;
...
}
......
Functions
..
- setverfiedToken(address _orgAdd, bool _status)
..

Figure 5.9: The relationship between the approach and the TID

5.5 Limitations

In this section, we will discuss some of the limitations of the approach.

Table 5.1: Transaction fees of each operation with gas cost of 2 Wei on July 20, 2018

Function	Amount of used Gas	Transaction fee in Ether	Transaction fee in $
mint	173690	0.0003474	0.17162
transfer to project	65799	0.0001316	0.06501
transfer from project	35086	0.0000702	0.03468
transfer	32309	0.0000646	0.03191
split	102130	0.0002043	0.10092
unlockDai	57559	0.0001151	0.05686

First, the donation token does not follow the standards of ERC20 tokens; therefore, it will not be able to be managed by a normal Ethereum wallet. This could affect the availability of the approach in a case when the server that runs the front end is down. However, any user can obtain the Abi-definition and the Address of the smart contracts, and with a simple import in the Mist wallet, he will be able to access all functionality at any time.

The second limitation is the function balanceOf(address user), which is not implemented directly in the smart contract. In this case, if the front end is not available, then the mist wallet will not be helpful to check the balance of an account. We chose to remove this function because managing the balance of each donor inside the smart contract will cost a high amount of gas. The reason is the need for an array for each donor, and with every transfer between owners or contracts, the array of the new and old owners should be modified and updated. Furthermore, we believe that a user does not necessarily need that function because he will likely only focus on his tokens and trace them to their destination.

The last limitation comprises the transactions fees, which are listed in 5.1. The fees of each operation separately are inexpensive; however, they could accumulate to a relatively larger amount than expected. For example, for a simple donation, the mint operation will first be invoked, which costs 173,690 gas; then, the donor will donate it to a project, and this will cost an additional 65,799. Thereafter, with a high probability, the project will split the received token into several tokens — calling a two - split function will cost (102,130 * 2) gas — now, there are three tokens in the system, and the estimated number of transfers is three. Each transfer from a project to an EOA will cost (35,086 * 3), while a normal transfer will cost (32,309 * 3), and last, unlocking the DAIs costs (57,559 * 3), which brings the total estimated donation of one token to 818,611 gas, and the sum of all steps will cost approximately 0.0016372 Ether, which is equal to 0.80714$.
That means our approach is not optimal for small donations, and it is limited to larger ones. Furthermore, we cannot choose a low limit value to donate because the Gas price is not stable, and the number of transfer calls can just be estimated.

6 Conclusion

6.1 Summery

In summary, the thesis discusses and demonstrates how the public blockchain and smart contract address can solve issues related to donations and the work of civil organizations, such as a lack of transparency, corruption and decreased trust levels.

The main purpose of this thesis was to use the benefits of DApp to solve the mentioned issues. To reach this goal, we presented two possible use cases. The first introduced use case was oriented towards the transparency of the work of civil organizations, and it is based on the principles of TID and its 10-point system. In this use case, we declare the exact status of the organizations depending on several conditions enforced by a smart contract. Furthermore, we prevent the information from being manipulated by uploading the hash values into the blockchain, and finally, the smart contracts define the exact role of external users and enforce their actions on the system. In contrast, the second use case was built as an approach, and it focused more on the way in which to be used for the best controlled donations, the role of donors in the suggested way and the best method for the management of projects. we achieved that by presenting a schema for a donation model with a DonationToken that has a special ID and features. The token could transfer the value of a donation in an efficient, transparent and traceable manner, which offers advantages not only for organizations but also for donors, who would know exactly where their impact is. We also combined the work of the two use cases by assigning the TID organization from the first use case the task of confirming the correctness of the work of the second use case.

We developed the related prototypes as DApps on the Ethereum platform. The smart contracts were implemented with Solidity and the front-end with HTML, CSS and basic JavaScript.

6.2 Future Work

The developed use cases serve as the basis for our idea in a good way. However, before running studies to measure the effect, adoption and acceptance of those use cases, it is possible to make further improvements and add more functionality to the prototype, such as the following:

- Increase the interaction with users by expanding their role in the first use case — it is possible to add a new method that any user can call to give a negative or positive point to a specific organization, and when an organization receives a certain number of negative points, a voting process will be necessary to determine whether the organization is publishing the correct information. However, in this case, a new reward system is necessary to cover the remaining amount of Ether. Furthermore, determining the number of negative points that would lead to a report is critical, and it requires further investigation and time to observe the adoption and interaction quote with the contract.

- Implement the DonationToken as ERC721 and support micropayments - making the DonationToken follow the Ethereum standards ERC721 could afford the token more flexibility because it will be able to be held and transferred through any Ethereum wallet that support ERC721, and it will also be usable in micropayment channels in the future. Using those micropayments could solve the scalability issues of the blockchain as well as the problem of the accumulated transactions fees in our approach. Charities could establish micropayment channels between themselves and donors on the Ethereum network using Raiden[1], which is a protocol that implements and supports these types of channels. However, with the already introduced approaches such as sharding, plasma and proof of stake, the need for micropayment channels is decreasing.
- Allow for monthly donations - many projects and charities prefer to receive their donations at a monthly rate, which will grant them the ability to make long-term plans because they know the minimum amount of donations that they will be receiving in the future. With regard to our approach, it is possible to implement additional smart contracts that would acquire the DonationToken, instead of the project acquiring it, and then split it into 12 parts; each month, the smart contract would release only a part of it. However, for small donations, this would be totally inefficient because of the additional transaction fees
- Allow for community governance in our approach, the organization is the entity that confirms the needs of its projects by adding them to the management contract, thereby enabling them to receive donations. However, this step requires further improvement. It is possible to increase the role of donors in the approach and grant them the power to confirm the needs of projects, combined with the organization, but as previously mentioned, donors make ‘taste-based’ rather than ‘needs-based’ choices. Therefore, constructing such a governance protocol requires further research and adoption tests.

Disclaimer:

The implemented prototype and all used smart contracts in this thesis could be enhanced to save more gas; however, they could also contain several bugs that lead to one losing money. For those reasons, they are not ready to be deployed on the Ethereum live net. Additionally the amount of funds and rewards in the first use case are just a suggestion.

[1]https://raiden.network/

Bibliography

[Bas17] I. Bashir. *Mastering Blockchain*. Packt Publishing, 2017. ISBN: 1787125440 (cit. on pp. 21, 24, 25).

[BM17] A. Bahga, V. Madisetti. *Blockchain Applications: A Hands-On Approach*. VPT, 2017. ISBN: 0996025561 (cit. on pp. 22, 32).

[BOOESC18] Jan Paul Buchwald. *51Nodes – Overview of Ethereum Smart Contracts and DApps*. 2018. URL: https://www.dropbox.com/sh/fm715xioygxnr10/AAD_Lzi6e9z6qJl9AmnA-wGPa?dl=0 (cit. on pp. 29, 30).

[Bre10] B. Breeze. "Centre for Charitable Giving and Philanthropy –How donors choose charities". In: (2010). URL: https://www.kent.ac.uk/sspssr/philanthropy/documents/How%20Donors%20Choose%20Charities%2018%20June%202010.pdf (cit. on p. 52).

[But13] V. Buterin. "A Next Generation Smart Contract & Decentralized Application Platfrom". In: (2013). URL: http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (cit. on p. 23).

[CAPB00] E. A. Brewer. *Towards robust distributed systems. (Invited Talk)*. 2000. URL: http://pld.cs.luc.edu/courses/353/spr11/notes/brewer_keynote.pdf (cit. on p. 22).

[Che14] H.-C. Chen. "International Transactions 101 – Credit Card". In: (Apr. 2014). URL: https://moneymattersforglobetrotters.com/international-transactions-101-credit-card/ (cit. on p. 51).

[Dha16] A. Dharssi. "Charitable giving in Canada drops to 10-year low, according to tax data". In: (Dec. 2016). URL: https://globalnews.ca/news/3130108/charitable-giving-in-canada-drops-to-10-year-low-according-to-tax-data/ (cit. on p. 35).

[Dre17] D. Drescher. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. mitp, 2017. ISBN: 1484226038 (cit. on p. 17).

[DSS94] *Digital Signature Standard (DSS)*. 1994. URL: https://web.archive.org/web/20131213131144/http://www.itl.nist.gov/fipspubs/fip186.htm (cit. on p. 18).

[EOSico18] John Grinder. *Block.one has raised a record $ 4 billion after one year long ICO*. 2018. URL: https://block-chain.com/news/blockone-has-raised-a-record-4-billion-after-one-year--long-ico- (cit. on p. 32).

[Lam82] L. Lamport. “The Byzantine Generals Problem”. In: *ACM Transactions on Programming Languages and Systems* (1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf (cit. on p. 20).

[Mak17] the Maker Team. “The Dai Stablecoin System”. In: (Dec. 2017). URL: https://makerdao.com/whitepaper/DaiDec17WP.pdf (cit. on p. 52).

[Mer79] R. Merkle. *Secrecy, Authentication and Public Key Systems*. Stanford Electronics Laboratories, 1979. URL: http://www.merkle.com/papers/Thesis1979.pdf (cit. on p. 19).

[MW17] R. Mazet, J. Wojciechowski. “Alice white paper”. In: (Nov. 2017). URL: https://github.com/alice-si/whitepaper/blob/master/Alice%20white%20paper%20-%20FV%200.9.pdf (cit. on pp. 36, 37).

[Nak08] S. Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: (2008). URL: https://bitcoin.org/bitcoin.pdf (cit. on pp. 19, 20).

[RD10] R. Rodrigues, P. Druschel. “Peer-to-Peer Systems”. In: *The Journal of Neuroscience* 53.10 (2010), pp. 72–82. URL: http://zoo.cs.yale.edu/classes/cs426/2012/bib/rodrigues10peer-to-peer.pdf (cit. on p. 20).

[RFP12] Raymund Flandez. *Philanthropy – Donors Say They Would Give More If They Saw More Results*. 2012. URL: https://www.philanthropy.com/article/Many-Donors-Would-Give-More-if/156463 (cit. on pp. 15, 35).

[Sch16] K. Schmeh. *Kryptografie: Verfahren, Protokolle, Infrastrukturen*. dpunkt.verlag GmbH, 2016. ISBN: 3864903564 (cit. on p. 17).

[ST14] A. K. Schnackenberg, E. C. Tomlinson. “Organizational Transparency: A New Perspective on Managing Trust in Organization-Stakeholder Relationships”. In: *Journal of Management* (Mar. 2014), pp. 1–27. URL: http://journals.sagepub.com/doi/abs/10.1177/0149206314525202?journalCode=joma (cit. on pp. 39, 40).

[Sza97] N. Szabo. “Smart Contracts: Formalizing and Securing Relationships on Public Networks”. In: *First Monday* 2 (1997), pp. 1–9. URL: http://firstmonday.org/ojs/index.php/fm/article/view/548/469-publisher=First (cit. on p. 28).

[TCC16] The Charity Commission. *Gov.UK – Public trust in charities has fallen, reports Charity Commission*. 2016. URL: https://www.gov.uk/government/news/public-trust-in-charities-has-fallen-reports-charity-commission (cit. on pp. 15, 35).

[TLCM15] Tami Luhby. *CNN Money – 71% of the world’s population lives on less than $10 a day*. 2015. URL: http://money.cnn.com/2015/07/08/news/economy/global-low-income/ (cit. on p. 15).

[UKcharity13] Abby Young-Powell. *The GuardianUK –charities are missing out on £665m in donations every year*. 2013. URL: https://www.theguardian.com/voluntary-sector-network/2013/mar/14/charities-missing-donations-survey (cit. on p. 35).

[VBERC2016] V. B. Fabian Vogelsteller. *ERC20 Token Standard*. 2016. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md (cit. on p. 26).

[Woo14] D. G. Wood. “Ethereum: A Secure Decentralized Generalized Transaction Ledger”. In: (2014). URL: https://ethereum.github.io/yellowpaper/paper.pdf (cit. on p. 23).

[Ziv16] I. T. Zivilgesellschaft. “Leitfaden für die Umsetzung von Selbstverpflichtungserklärungen”. In: (2016). URL: https://www.transparency.de/fileadmin/Redaktion/Mitmachen/ITZ/Initiative-Transparente-Zivilgesellschaft-Leitfaden.pdf (cit. on p. 40).

All links were last followed on August 26, 2018.

A Appendix

Tools

Some of the tools and interfaces that are usually required for developing and using decentralized applications:

- **Truffle**
  Truffle[1] is one of the various development environments that are available for Ethereum. It makes it easier and simpler to compile, link and deploy smart contracts on the blockchain, and it provides a test framework using *Mocha* for automated testing and *Chai* for assertions. Truffle can interact with all types of Ethereum Nets and even with TestRPC; furthermore, it provides a command-line interface to run web3 commands and to interact with the blockchain and the deployed contracts.
- **TestRPC**
  Ethereum has various Test Nets; furthermore, users can easily construct their own private Ethereum blockchain using geth. Those Nets are an ideal place in which to develop and test smart contracts before deploying them on the main Ethereum Net; However, there is an easier and faster environment to test and run smart contracts; that environment is known as Testrpc (the currently provided Testrpc from Truffle is called *Ganache*[2]).
  Ganache is a Node.js-based Ethereum client, and it simulates full-node client behavior and creates an in-memory blockchain. It also includes all RPC functions and additional features, such as time manipulation and managing accounts; however, it had an issue that the developer should care about: holding and simulating everything in memory, which means it loses the state when the node is restarted.
- **Metamask**
  MetaMask[3] is a browser extension, and it provides a friendly user interface that allows users to visit and interact with DApps using the normal Internet browser without running a full-node. *MetaMask* includes a secure identity vault, providing an interface to manage EOAs and ERC20 tokens and to sign transactions. However, it does not support mining or deploying contracts such as Mist, the original wallet, and a browser that is developed and supported by the Ethereum Foundation..

[1] https://github.com/trufflesuite/truffle
[2] http://truffleframework.com/ganache/
[3] https://metamask.io/

- **Versions:**
  - Truffle: v4.1.8 (core: 4.1.8)
  - Solidity: v0.4.23 (solc-js)
  - MetaMask: V4.7.4
- **Ganache configuration:**
  - Hostname: 127.0.0.1
  - Port Number: 8545
  - Network ID: 5777
  - Mnemonic: virus mimic manage dress jazz gauge teach destroy upon pool seat tip
  - Gas limit: 8000000
- **bower components:**
  - bignumber.js
  - chai
  - crypto-js
  - jquery
  - mocha
  - sprintf
  - web3: v0.20.6

Files

To initialize the project and then deploy the smart contracts using truffle the following commands are needed:

- Run the command `truffle init`
- Then modify the file truffle.js as shown in A.1.
- In the migrations file add 2 JavaScript files. The first one is called 2-deploy-contracts.js and the second one is 3-deploy-contracts.js. The code inside those files is shown in A.2 and A.3.
- After coping the smart contracts in the contracts file run the command: `truffle compile`
- And then the command: `truffle migrate --reset`

Listing A.1 Truffle Configuration

```
module.exports = {
  networks: {
    development: {
      host: "localhost",
      port: 8545,
      network_id: "*", // * to Match any network id
      from: "0xf62fef864D88e3d70F22468EEe3B412b67114946"
    }
  }
};
```

Listing A.2 deploy Contracts Use Case

```
var TransparentCivilSociety = artifacts.require("./contracts/UseCase/TransparentCivilSociety.sol");
var AddressBook = artifacts.require("./contracts/UseCase/AddressBook.sol");

module.exports = function(deployer) {
  deployer.deploy(AddressBook);
  deployer.deploy(TransparentCivilSociety);
};
```

Listing A.3 deploy Contracts Approach

```
var DonationTokenOrg = artifacts.require("./Approach/DonationTokenOrg.sol");
var FaiToken = artifacts.require("./Approach/FaiToken.sol");
var ProjectOrg = artifacts.require("./Approach/ProjectOrg.sol");
var MainOrg = artifacts.require("./Apporach/MainOrg.sol");
var VaultOrg = artifacts.require("./Approach/VaultOrg.sol");
var ManagementOrg = artifacts.require("./Approach/ManagementOrg.sol");

module.exports = (deployer) => {
  deployer.deploy(DonationTokenOrg);
  deployer.deploy(FaiToken);
  deployer.deploy(MainOrg);
  deployer.deploy(ManagementOrg);
  deployer.deploy(VaultOrg);
  deployer.deploy(ProjectOrg, 'FirstProject', 'first.com', 'Africa');
  //deployer.deploy(ProjectOrg, 'SecPorject', 'second.com', 'Asia');
  //deployer.deploy(ProjectOrg, 'ThirdProject', 'third.com', 'space');
};
```

Listing A.4 The code of two simple Smart contracts for figure 4.2

```
pragma solidity ^0.4.23;

contract Example {
    // the address of the Organization.
    address organization =  0xca35b7d915458ef540ade6068dfe2f44e8fa733c ;

    // an integer variable.
    int public foo;

    // an event to register the change of foo in the logs.
    event Changed(int256 indexed _newValue, uint256 indexed _time);

    //constructor
    constructor() public {
        foo = 1;   //initialize foo to 1
    }

    function getFoo() public constant returns (int) {
        return foo;
    }

    function setFoo(int _foo, address _SecContract) payable {
        require(msg.value == 1 ether);   //check the required amount ether to continue.
        require(msg.sender == organization);   //check if the owner invoked the function.
        foo = _foo;   //change the value of foo
        changeFooOfOthercontract(_SecContract, _foo);   //call an internal method.
        emit Changed(_foo, block.timestamp);   //emit an event.
    }

    function changeFooOfOthercontract(address _SecContract, int _newFoo) internal {
        SecondContract secContract = SecondContract(_SecContract);   //create an instance.
        secContract.changeFoo(_newFoo);   //call the public method changeFoo from the other contract.
    }

    function increment() public {
        foo = foo + 1;   //increment the value of foo.
    }

}

contract SecondContract{

        // an integer variable.
    int public foo = 0;

    function changeFoo(int _newFoo) external {
        foo = _newFoo;   //change the value of foo in the contract.
    }
}
```

Listing A.5 The Abi-definition of a Smart contract 2.7.

```
[
        {       "constant": true,
                "inputs": [],
                "name": "getTheName",
                "outputs": [
                        {       "name": "",
                                "type": "string"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {       "constant": true,
                "inputs": [],
                "name": "owner",
                "outputs": [
                        {       "name": "",
                                "type": "address"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {       "constant": false,
                "inputs": [
                        {       "name": "_newName",
                                "type": "string"
                        }
                ],
                "name": "changeTheName",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {       "inputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "constructor"
        },
        {       "anonymous": false,
                "inputs": [
                        {       "indexed": true,
                                "name": "_oldName",
                                "type": "string"
                        },
                        {       "indexed": true,
                                "name": "_newName",
                                "type": "string"
                        },
                        {       "indexed": false,
                                "name": "_time",
                                "type": "uint256"
                        }
                ],
                "name": "ChangedName",
                "type": "event"
        }
]
```

Declaration