Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Enabling the Digital Economy with Decentralised Autonomous Organisations on the Blockchain.

Author:
Pierre, Eugene Valassakis

Supervisor:
Prof. William Knottenbelt

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 2016

Abstract

The formidable potential of Distributed Ledger Technology (DLT) and the Blockchain has gained wider recognition in recent years, even attracting the attention of Governments and major Banks. Parallel to this backdrop is the ever increasing focus on SMEs and freelancers as a key driver of growth in the United Kingdom. Both categories are nonetheless facing a similar set of challenges in the current economic landscape. Identified issues include cash access difficulties, struggle in opportunity discovery and hard access to adequate talent. Part of the solution to these issues lies in new forms of economic interaction and a shift of paradigm towards the Digital Economy. Finding innovative ways to empower these professionals in joining the digital movement is therefore paramount, and has the potential for major social and economic impact.

One of the more recent innovations in the DLT space is Ethereum. This technology has further enlarged the scope of possibilities offered by DLT, by allowing the ledgers to keep track of Turing complete programs (called smart contracts) as opposed to simple data packages. Decentralised Autonomous Organisations (DAOs) are a novel concept emerging from this context. These entities build upon the idea that smart contracts can be used to govern and automate organisational operations, thereby increasing security, transparency and efficiency in the associated processes.

In our project we investigate the role that DAOs can play in the creation and promotion of a fully digitalised economy. Particularly, we examine and develop a digital ecosystem coordinated by DAO entities, which organises an efficient economic interaction between its participants. In this regard, our project demonstrates how DLT can be used for enabling the Digital Economy, and showcases the way such an ecosystem creates a beneficial environment for all users. Our model, named "Decentralised Startups", specifically considers all parts of the corporate life-cycle, namely (1) creation and funding, (2) employment, (3) content creation and (4) content distribution and revenue management. For each category we provide schemes and workflows that allow harnessing the potential of the underlying Blockchain technology and that cooperate together for delivering a full and powerful solution.

Acknowledgments

- To my supervisor, Prof. William Knottenbelt, for his help, guidance and enthusiastic support throughout this project and during the entire year.
- To my friends, for their support.
- To my family, for their love and support.

Contents

Chapter 1

Introduction

1.1 Motivation

In the United Kingdom (UK), Small and Medium Enterprises (SMEs) make up more than 99% of all private sector businesses and are responsible for about 47% of the revenue in the sector [1]. Additionally, freelancers become an ever increasing part of the UK economy, amounting to more than 1.5 million freelancers in 2015, a growth of 36% since 2008 [2]. Both SMEs and freelancers report a similar set of challenges faced in today's economic landscape. These include money matters such as cash access, prompt cash collection or funding issues, and difficulties in opportunity discovery such as the access to talent or self-promotion and marketing [3, 4, 5, 6]. Parallel to this economic landscape is the rise in attention from the Government towards the Digital Economy [7] and the novel concept of Distributed Ledger Technology (DLT) [8]. The Digital Economy is seen as an important driver of growth [7, 9, 10, 11], while DLT, first introduced in 2008 through the Bitcoin digital cryptocurrency, has gained significant momentum in the later years, recently even getting the attention of major international Banks [12, 13].

Of the many applications spinning off the principles of Bitcoin [14], and its main innovation, the Blockchain [15], few carry the disruptive potential of Ethereum [16]. Ethereum, originated in 2014 by Vitalik Buterin [17], introduces a mechanism for Blockchain technology to move away from a simple transaction and data storing mechanism. As a matter of fact, Ethereum introduces a fully fledged platform for decentralised applications, allowing Turing complete programs to live and execute on the Blockchain [16, 17]. This ability gives rise to numerous possibilities, the most famous being smart contracts. Smart contracts are executable pieces of code that are stored and executed on the Blockchain, and that can be made to reflect terms and clauses of real paper based contracts [18, 19, 20].

These smart contract structures can then be used to spawn a vast amount of possible decentralised applications, a truly remarkable one being Decentralised Autonomous Organisations (DAOs) [21, 22]. DAOs are a novel concept at a very young, experimental stage. They bring forth the idea that corporations, NGOs or any other

human organisations can be run autonomously and securely on the Blockchain, with automated code determining their governing and operating procedures [22, 23]. DAOs and other decentralised applications have gathered strong media attention, with recent highlights even incorporating concepts such as Arcade City, the so called "Decentralised Uber" [24, 22, 25, 26].

From these new capabilities offered by Ethereum, we foresee an opportunity for the establishment of a complete ecosystem promoting a new form of economic interaction through cryptocurrencies, DAOs and smart contracts. We also anticipate that such an ecosystem will be opening new doors and opportunities for freelancers and SMEs, by enabling them to be an integral part of the Digital Economy [9, 10, 11].

1.2 Objectives

This project aims to capitalise on the presented opportunity by defining, exploring and demonstrating such a digital economic environment. Specifically, we aspire to use smart contracts and the Ethereum Blockchain in order to address the following:

- The definition and development of a model for enabling the Digital Economy using DLT.
- The identification of the corporate processes that need to be modelled, and the development of the means by which they can be represented through smart contracts in our system.
- The development and research into DAOs and their role in this digital commercial ecosystem.
- The investigation of the technical systems that need to be put in place in order to empower a digital platform that enables such an environment.

1.3 Contributions

In our research we have examined the possibilities offered by the Ethereum technology and developed a model for economic interaction through Decentralised Autonomous Organisations. Our model, named "Decentralised Startups", focuses on creating value for all of its participants by enabling a direct, transparent and secure commercial interaction throughout the identified chain of the corporate life-cycle: funding, employment, content creation, content distribution and revenue management.

More specifically, the Decentralised Startups model creates the infrastructure that promotes and supports the following :

- Three governance schemes and organisation types (private, corporate and public), allowing for enterprises to have a tailored participation in the digital ecosystem depending on their needs.
- Easy access to funding via processes tied to the organisational types. Enhanced by opportunity discovery channels of the platform, these create value for both investors looking for promising organisations to support and for entrepreneurs or SMEs that have access to alternative, simple and secure channels of financing.
- Four contracting and employment models through smart contracts and DAOs, allowing for businesses to seemingly discover and employ talent tailored to their needs and for freelancers and other contractors to promote themselves and compete for the contracting opportunities.
- Facilitation of the content-creation process via easy access to talent and fair competition throughout the system.
- Revenue management and redistribution channels for Shareholders to see the benefits of their investments.
- Blockchain-secure processes in every commercial and governance operation, virtually (1) eliminating any malicious and hidden behaviours and (2) guaranteeing payments on terms and in time to all parties involved .

This project also supplies a platform prototype which demonstrates the core principles of the envisioned ecosystem. This implementation consists of a proof-of-concept focusing on the particular use case of content creation and distribution via individual freelancing contracts. Main achievements of our prototype are:

- A successful demonstration of the key principles of the Decentralised Startups environment.
- A practical and visual representation of the concepts developed as part of the ecosystem.

1.4 Outline

This report is structured in four main thematic chapters: The background Theory of distributed ledgers and the Blockchain, the specific topic of Decentralised Autonomous Organisations, a depiction of the Decentralised Startups ecosystem and the description of the proof-of-concept implementation.

In Chapter 2 we initially present an overview of the working principles of DLT. We then discuss in more detail the Bitcoin protocol, gaining insight into the mechanisms involved with Blockchain technology. We finally present the Ethereum project, looking at its main innovations and characteristics and discussing the inner workings of its Blockchain and the differences it introduces with the classic Bitcoin paradigm.

In Chapter 3 we examine the meaning of the DAO term and take a deeper look into instances of DAOs and the different groups involved in developing and promoting this concept. In particular, we closely review "The DAO" [27], the most notorious example of a DAO and biggest crowd-funding operation so far [28], which has shaken the Ethereum community via a hack resulting in millions of Ether being compromised.

In Chapter 4 we move on to present the Decentralised Startups ecosystem. In this chapter we describe and justify our design principles and we look into the different workflows and platform components necessary towards enabling the digital economy. We then describe the appropriate contract structures and platform architectures that would enable reaching those goals.

In Chapter 5 we introduce our proof-of-concept implementation: We closely examine the specifications it follows and workflows it represents, as well as the different technologies and implementation details it involved.

Finally, at the end of the report we outline an evaluation of our project and results, as well as possible future lines of work that can build upon our work.

Chapter 2

Background Theory

2.1 Distributed Ledger Technology

Distributed Ledger Technology is a fundamental conceptual cornerstone this project builds upon. It is therefore important to start by disambiguating this term while building a conceptual understanding of what it represents. Fig. 2.1 showcases some of the DLT terminology while Fig. 2.2 shows the DLT industry landscape of 2016.

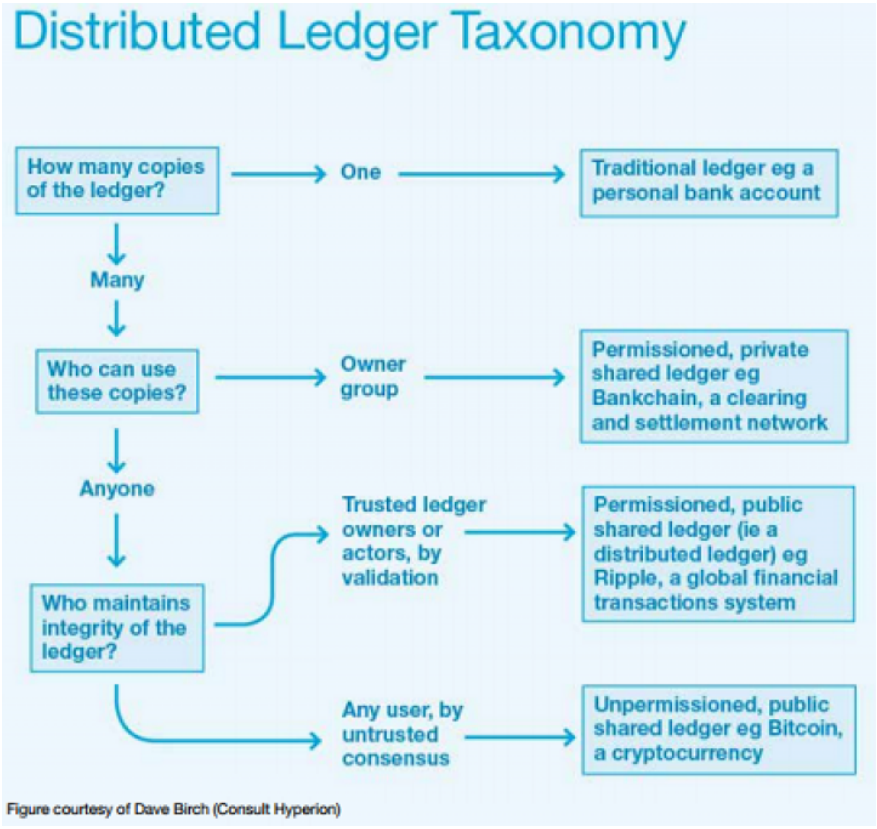


Figure 2.1: DLT Taxonomy (taken from *Distributed Ledger Technology: beyond block chain*, a UK Government Office for Science report [8]).

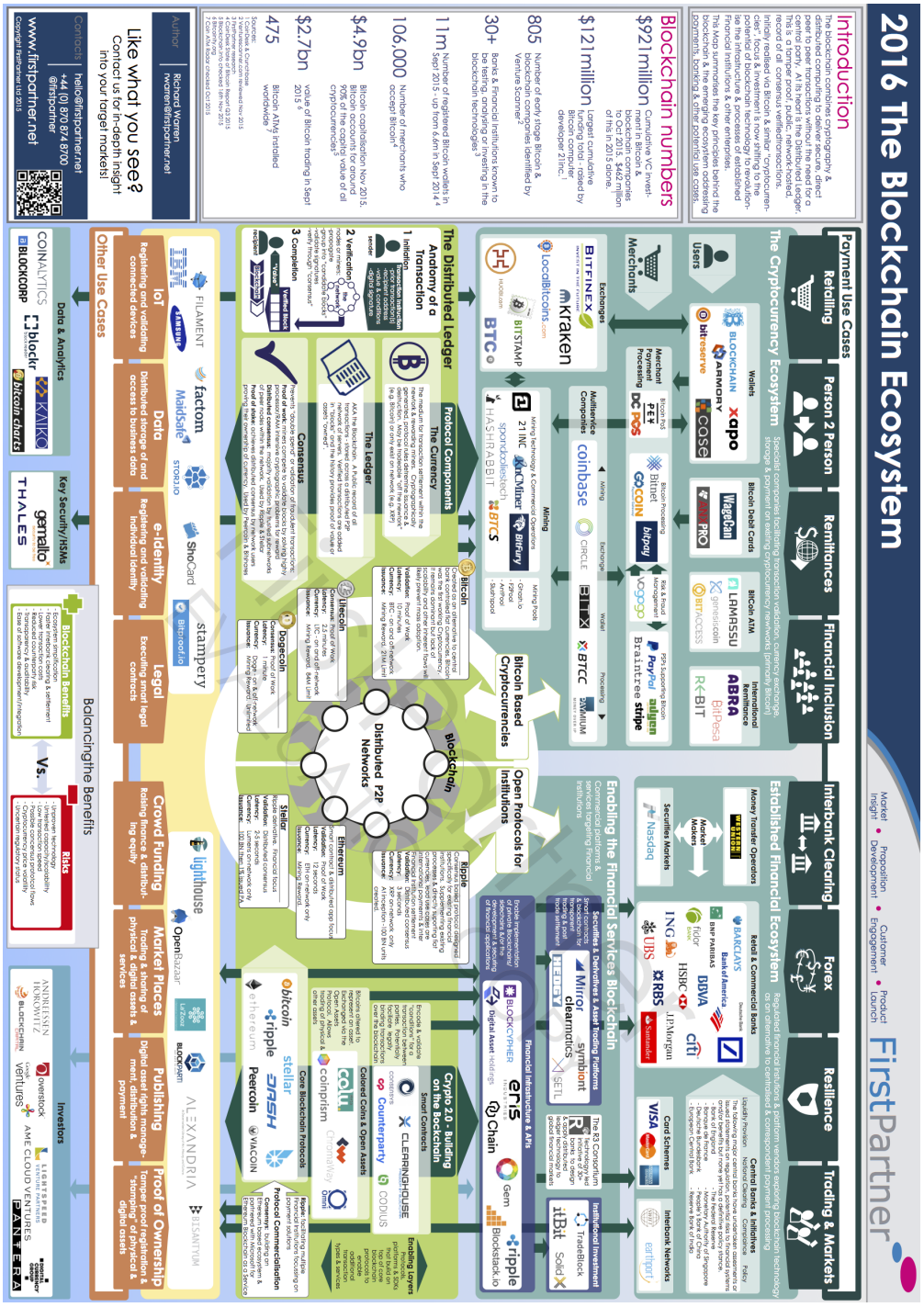


Figure 2.2: Blockchain 2016 Ecosystem (taken from FirtsPartner [29]).

DLT can be viewed as a decentralised database system formed by a distributed ledger and its attached protocol [8]. It is run on an underlying network where each computing unit makes up a node [8, 30]. As illustrated in Fig. 2.1 and Fig. 2.2, there exists many variations of DLT, with arguably the most important distinction being between permissioned and permissionless systems [30, 31]. As the names suggest, the main difference is that the former (e.g. Ripple [32], Hyperledger [33]) only allow a specific set of nodes to modify their associated ledger while any willing user can contribute to the latter (e.g. Bitcoin [14],Ethereum [16]).Subsequent mentions of DLT throughout this report will be referring to permissionless systems, unless explicitly stated otherwise.

A Distributed Ledger is a shared public ledger that is owned by all the nodes participating in the network. It essentially serves the same purposes as a classic ledger: keeping track of transactions, ownership and value transfer while preventing fraud and inconsistencies in the underlying asset distribution [8]. As opposed to any traditional system however, each node has a copy of the ledger (and hence has auditing power) and the whole network works together to maintain its integrity [34, 35, 8].

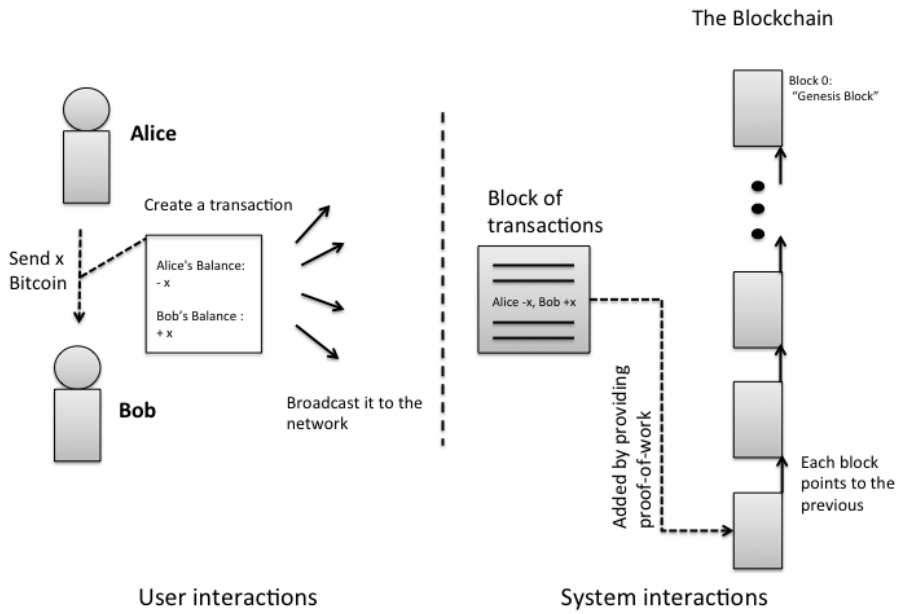


Figure 2.3: Conceptual representation of the Blockchain formation (inspired from [34, 35]).

The most common system used by DLT is known as the Blockchain, and was first introduced in Satoshi Nakamoto's Bitcoin original paper [36, 15]. In the following sections we will go into much technical detail on the inner workings of the Blockchain technologies of Bitcoin and Ethereum. Before doing so however we believe it is important to give a simple conceptual overview of the process associated

with DLT, which can also help as a reference point in the subsequent technical discussions. Perhaps unsurprisingly, this is most easily achieved through the example of a Bitcoin transfer.

Fig. 2.3 illustrates two parties, Alice and Bob, where Alice is sending x Bitcoin to Bob. The process starts with the creation of a transaction that is broadcast to the network.The network then needs to reach consensus on which transactions are valid. In order to achieve this, special nodes called miners have the task of validating all the broadcast transactions that have not yet been validated. They collect them in groups called "blocks", and by providing cryptographic evidence (called proof-of-work), they add these transaction blocks to the set of confirmed transactions accepted by the network. In fact, each validated block points to the one that was validated before it. This makes up a "chain" of blocks that goes all the way down to the very first block, and forms the data structure known as the Blockchain [37, 35, 34].

2.2 The Bitcoin Prototcol

As briefly mentioned previously Bitcoin is the first implementation of Blockchain technology. Although, as seen in Fig. 2.2, a multitude of other cryprocurrency systems have followed, the fundamental principles of distributed ledgers are still best represented by Bitcoin. Most importantly, Ethereum -the main technology used in our project- also follows many of these principles, with the main differences illustrated in Section 2.3. As such, it is important to start our discussion with the essential parts of the Bitcoin protocol. Throughout this section, we will be referring to generic Bitcoin holders by Alice, Bob and Eve and consider examples of different interactions between these parties.

2.2.1 Nodes, Addresses and Cryptography

Anyone who wants to participate in the Bitcoin project can do so by downloading a client (user-side software) to become a node in the Bitcoin network. It is important to note that there are different types of clients (and by extension of nodes) that exist and that cater to different needs. Nevertheless, all these are typically subsets of the full Bitcoin node adding no further insight to our considerations, and hence we will only consider the complete client in our discussions [38].

When transferring Bitcoin, the value transfer takes place between two Bitcoin Addresses. Anyone who can prove they are behind the receiving address can then claim the Bitcoin transferred for themselves [38, 35, 34].

In order to allow such an interaction, the Bitcoin network uses digital signatures and cryptographic keys established through Public Key Cryptography. This technique is used to create pairs made of a private key and a public key. As the name suggests,

it is cryptographically secure to broadcast the public key over any conventional public channel, while the private key is used to prove ownership of the public key. This then is conceptually the fundamental mechanism through which owners of receiving addresses in Bitcoin transfers can prove their identity [38, 34, 39].

For this mechanism to hold, the mathematical functions that are used to create public keys from private keys need to have the crucial property of being one way functions. This property means that it is computationally easy to get the public key from the private key but it is computationally infeasible to get the private key from the public key (cf. Fig. 2.4a). Bitcoin uses an Elliptic Curve Digital Signature Algorythim to ensure this, and specifically the secp256k1 standard, which specifies a set of parameters to be used in the calculations [38, 39, 40].

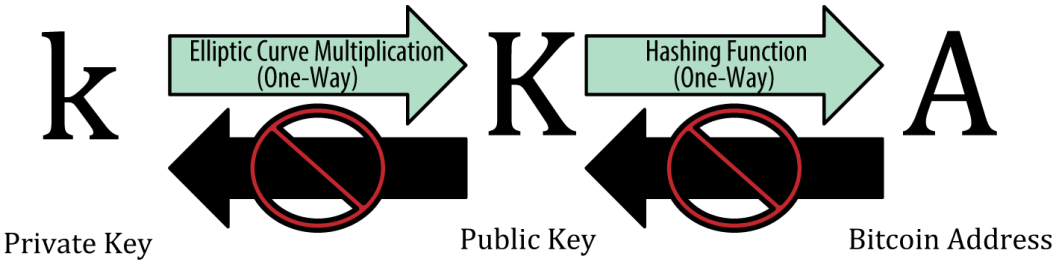
Another important point is that users do not interact directly with public keys. Instead they deal with Bitcoin Adresses, which are a cryptographically hashed and encoded version of the public key. This is illustrated in Fig. 2.4b. The reason for this process is (1) to add an extra layer of security, (2) increase convenience through a simplified representation of the public key and (3) to add a layer of protection against errors such as typos [38].

Arguably the most important function in the process is the Secure Hash Algorythm (specifically SHA256) which is a cryptographic hash algorithm used throughout the Bitcoin and Ethereum protocols [38, 24, 39]. Main properties of this hash function are that (1) it produces a 256 bit output from an arbitrary length input and (2) it is only one way. As will be apparent in the following sections, a useful implication of this is that the only practical way to find the input to SHA256 that produces a given output is through brute force, i.e trying random inputs until one gives the desired result [38].

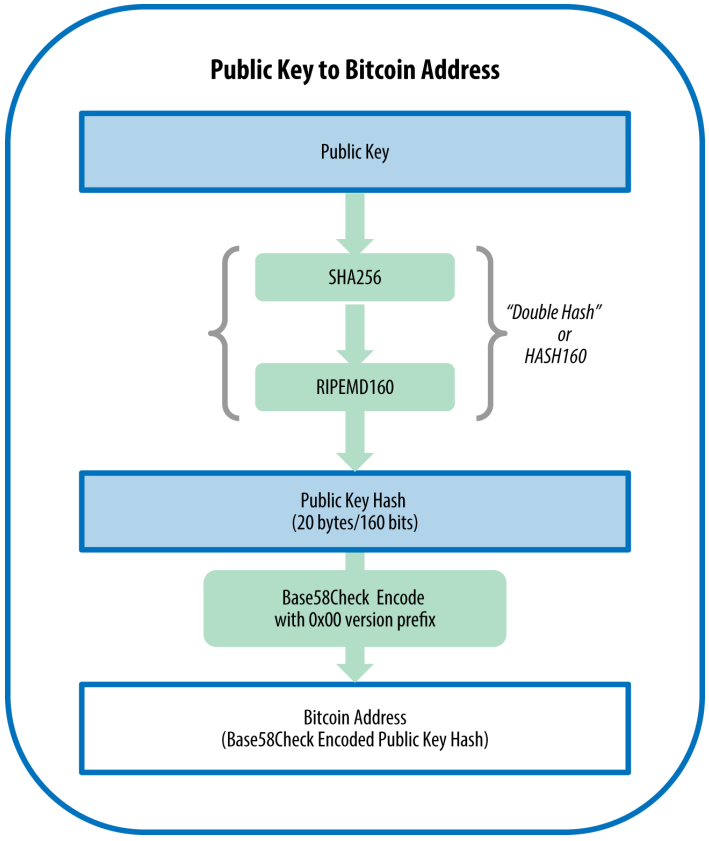
2.2.2 Transactions

Bitcoin transactions are paramount to the Bitcoin architecture since they are the only value capturing information stored in the Blockchain database. As such, unlike any traditional currency system, there are neither accounts nor balances, and transferring Bitcoin does not correspond to depleting one party's, say Alice's, balance while filling the counterpart's,say Bob's, balance. Instead, value is transferred from transaction to transaction, with the inputs of one transaction being the outputs of another transaction. Consequently, in order to determine how much Bitcoin corresponds to say Alice, Alice's node needs to find all transactions that transfer value to Alice, and sum up all of their outputs [34, 38, 24, 35].

The details of this mechanism become much clearer by understanding the structure of a Bitcoin transaction. Fig. 2.5 summarises this structure: (1) A Version field with the validation rules to use for the transaction, (2) An Inputs field which specifies the



(a) Private Key to Bitcoin Address overview.



(b) Algorithms used to obtain a Bitcoin Address from a public key.

Figure 2.4: Diagrams illustrating the process of obtaining a Bitcoin Addresses (taken from Antonopoulos' *Mastering Bitcoin, Figures 4-1 and 4-5* [38], as found at [41]).

inputs to a transaction, (3) An Outputs field that specifies the outputs of the transaction and (4) a Locktime field that indicates the minimum time in the future for that transaction to be processed (generally set to 0) [38, 39].

Furthermore, transaction inputs and outputs are essentially indivisible Bitcoin amounts along with cryptographic scripts that permit the establishment of ownership. More precisely, transaction inputs reference outputs of other transactions (cf. Fig. 2.6). In order to do so, a transaction input specifies a transaction hash that points to the transaction having the desired output and an identifier to locate this output within that transaction (cf. Fig. 2.5) [38, 39].

An important distinction is between Unspent Transaction Outputs (UTXOs) and Spent Transaction Outputs, with only UTXOs being valid candidates for transaction inputs. UTXOs also have an attached locking script, or Pubkey Script, which essentially ties the UTXO to a specific address [39]. In fact, only the holder of the private key corresponding to that address can generate a signature (or unlocking) script that will give him the right to use that UTXO in a transaction input [39, 38].

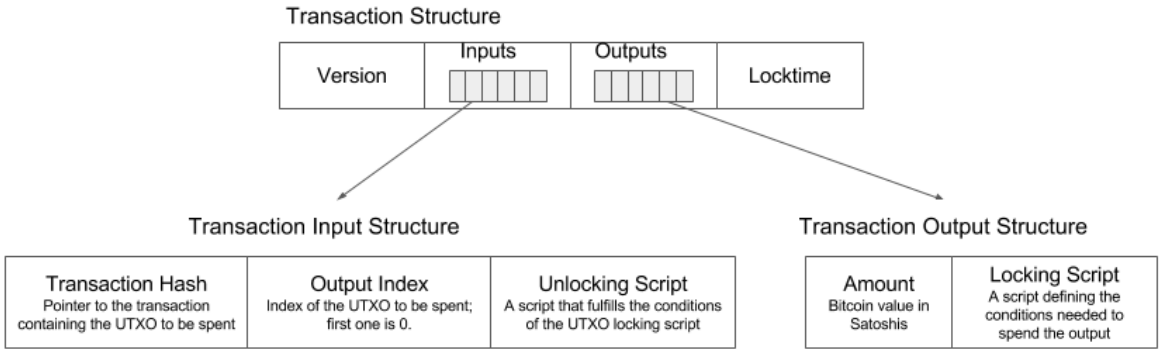


Figure 2.5: Simplified Representation of the Structure of a Transaction (adapted from the Bitcoin Developer Guide [39] and tables 5-1 , 5-2 and 5-3 of Antonopoulos' *Mastering Bitcoin* [38]).

To summarise, when Alice wants to send a certain amount A of Bitcoin to Bob, she creates a transaction with inputs that reference UTXOs whose sum equals or exceeds the amount that is being sent. She also provides scripts proving that she is the holder of the private key corresponding to the address bound to the UTXOs that are referenced. If the transaction validates, these UTXO inputs are marked as spent, and new UTXOs, now tied to Bob's address, are formed. If the input amount exceeds the amount sent, the transaction also creates UTXOs belonging to Alice as "change" [35, 38, 39, 34].

As an interesting note, in Ethereum's white paper [24], Vitalik Buterin equates the Bitcoin system to a state transition system. In his view, the Bitcoin state can be seen as set of all UTXOs, i.e. spendable and indivisible Bitcoin amounts cryptographically tied to a specific address. A transaction will then consume a certain amount of

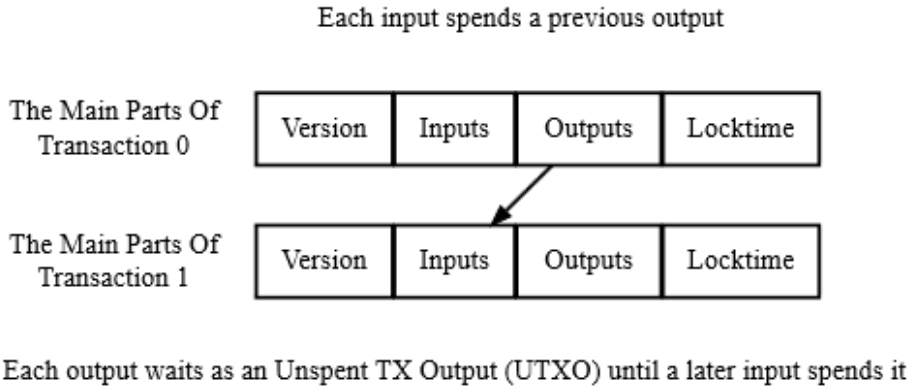


Figure 2.6: Referencing Transaction outputs (taken from the Bitcoin Developer Guide [39]).

UTXOs (which will then become spent outputs) and will return new UTXOs that can be subsequently spent, thereby changing the state of the system [24].

The final piece of the puzzle are coinbase transactions,that are the original source of UTXOs. Coinbase transactions are special types of transaction that do not require any input, and that create new Bitcoins as compensation for the miner nodes (cf.Section 2.2.3). These special transactions then steadily increase the total amount of Bitcoin in existence. Nevertheless, the rate at which these new Bitcoins are created decreases over time, and by the year 2140 no new Bitcoins will be created, at which stage miners will solely be compensated through transaction fees [38].

2.2.3 Blockchain and Consensus

Once transactions are created and sent to the network, they are not considered confirmed until validated by a mining node and added to the Blockchain [38]. In this section we will look at this process in more detail, through analysing the structure of a block and the mining process.

The structure of a Block

Transactions received by miners are bundled together in blocks. A simplified structure of a block is illustrated in Fig. 2.7a. As we can see, a block is composed of the transactions themselves, a block header and some metadata. It is within the block headers that the mission critical data are placed, with two particularly interesting fields being the *Previous Block Hash* and the *Merkle Root* [38]:

The Previous Block Hash field is a hash of the header of the previous block, obtained through the application of the SHA256 function twice (double-SHA256). This way,

the block is effectively linked to the previous block, thereby forming the Blockchain structure (cf. Section 2.1 and Fig. 2.7b) [38, 39]. An important implication of this is that if a block header is modified in any way its block hash will change, and then the next block in the chain will no longer point to it, making it infeasible to modify a block without also changing all the blocks that followed (cf. Section 2.2.4) [34, 38, 24].

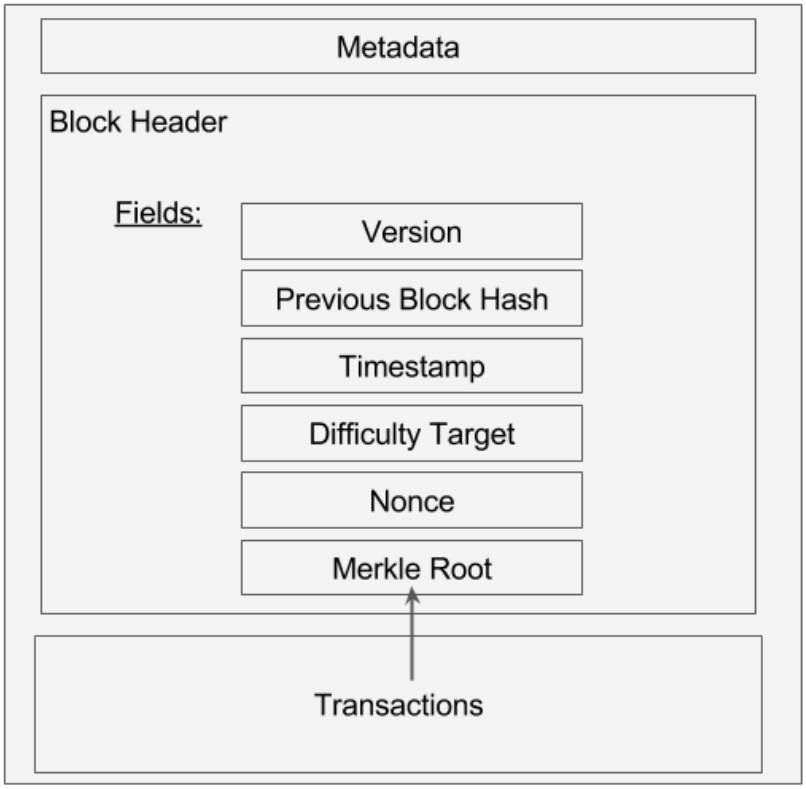
The Merkle Root field is the root of a merkle tree obtained from hashing the transactions included in the block. Specifically, pairs of transactions included in the block are hashed using double-SHA256 . The obtained hashes are paired and hashed again, and so on until a single hash remains : the merkle root. This system has a double benefit. On one hand, it allows for no transaction to be modified without modifying the block header, hence invalidating the block as described above. On the other hand, it allows for a much more efficient data storage and search that contributes to the scalability of Bitcoin [39, 38, 24].

Mining and Validation

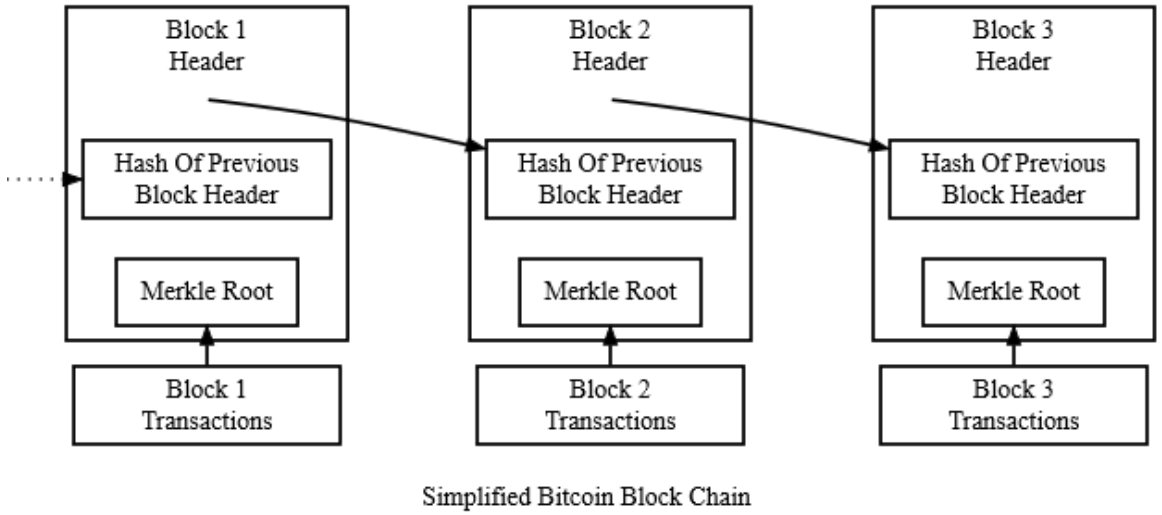
In order to place a new block to the Blockchain a miner needs to provide provide proof-of-work (PoW), which ensures it is very computationally expensive to do so [34, 35].

Technically, the condition for a new block to be added to the Blockchain is that its block hash needs to be smaller than a certain value. In order to achieve this, the miner can modify at will a single parameter in the header: the *Nonce* (cf. Fig. 2.7a). Because of the properties of the SHA256 algorithm discussed in Section 2.2.1, the only way to achieve this is by repeatedly trying different nonce values, taking the hash of the header and comparing it to the target value. An interesting note is that the the value of the target hash to be produced is dynamically adjusted by the network (through the *Difficulty Target* field on the block headers) so that, on average, a block is mined by the network every 10 minutes. Once a valid nonce (now PoW) has been found by the miner and the block is also confirmed as valid, it can be submitted to the network [39, 38, 24].

The validation mechanism of the block is very important and consists of a long checklist of rules that need to be confirmed before the block is deemed valid. Some major validation steps consist in (1) making sure the block is correctly constructed, (2) making sure that PoW is valid, (3) making sure that the block was made after the last block and that it points to a valid parent block, (4) making sure that all the transactions in the block are valid. Transaction validation itself consists of a checklist process that essentially verifies that the transaction is well formed and is not fraudulent (it would be verified for instance that the sending party has the necessary funds) [38, 24].



(a) Representaion of the Structure of a Bitcoin Block.(adapted from the Bitcoin Developer Guide [39] and tables 7-1 and 7-2 of Antonopoulos' *Mastering Bitcoin* [38]).



(b) Simplified Blockchain representation (taken from the Bitcoin Developer Guide [39]).

Figure 2.7: Diagrams illustrating the structure of the Bitcoin Blockchain.

Once the block has been broadcast to the network, each receiving node will independently validate this block by following the same process. Upon acceptance, the receiving node adds this new block to its Blockchain and propagates the information, allowing the network to eventually reach consensus. In the case that roughly more than one miner finds proof-of-work at the same time, the Blockchain forks, meaning that different nodes will be seeing different versions of the Blockchain. The situation however corrects itself as information propagates through the network, given that nodes will automatically choose the longest version of the Blockchain [34, 38].

2.2.4 A Secure, Transparent, Immutable environment

All the elements seen in the previous sections contribute to create a secure, transparent and virtually immutable environment. Because of the proof of work algorithm, it is computationally expensive to add a new block to the Blockchain. Because of the Merkle Root and the Previous Block Hash fields, any modification in a block or in a block's transactions will require to find a new proof-of-work for the block and any blocks that have followed [38, 34]. These circumstances make it practically infeasible to modify the history recorded on the Blockchain. In order to illustrate this, it is easiest to use the famous example of the double spend attack [35, 34]:

Consider that Alice possesses X amount of Bitcoin which she sends to Bob in exchange for a service. Bob then sees the transaction on his Blockchain, and waits for a few blocks to be mined before delivering the service. At that point, Alice tries to send the same X Bitcoin to Eve, in exchange for another service. If Alice submits this new transaction on the current version of the Blockchain, it will be rejected, as Alice's Bitcoin are already spent. So Alice has no choice but to try to rewrite history by changing the block where she sends the Bitcoin to Bob. Because her intervention changes the block hash, this creates a fork of the Blockchain, which Alice needs to make the network accept. In order to do so she needs to produce enough blocks to create a Blockchain that is longer than the one legitimate miners are working on. In order for Alice's attack to be successful, she would therefore theoretically need 51% of the combined computing power of the whole network, which is virtually impossible [35, 38, 24, 34].

An interesting remark from Vitalik Buterin in the Ethereum white paper is that, with the way the Bitcoin community has evolved with huge mining pools [42] gathering most of the mining power, it is not entirely inconceivable that such a 51% attack occurs. Nevertheless, if such an attack attempts to take place, honest miners that are part of the pool have the possibility of leaving it, thereby negating the attack [24].

2.3 The Ethereum Project

2.3.1 Conceptual considerations

One of the most recent implementations of DLT is the Ethereum project. Ethereum's main innovation is that it moves beyond a simple record keeping of transactions: It provides a platform and a Turing complete programming language that enables users to write, store and execute computer programs on the Blockchain [17, 43]. Interestingly, Bitcoin also provides a scripting language allowing for simple computation to take place.This language is however quite limited and is Turing incomplete, with the inability for example to create loops [24, 38]. Through its complex high level languages, what Ethereum aims to provide is the infrastructure for general Decentralised Application (or Dapp) development [17, 16].

At its core, Ethereum Blockchain provides the ability to write smart contracts. The term is attributed to Nick Szabo, which he originally defined as follows:

"A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries." (Nick Szabo, 1994 [19]).

As Josh Stark points out in his CoinDesk article however [18], the meaning of the term "smart contract" has taken an ambiguous turn. The article distinguishes between different ways the concept of smart contracts has been interpreted, and attempts to clarify the meaning of the term using three main categories: "smart contract code", "smart legal contracts" and "smart alternative contracts".

According to Stark, "Smart contract code" would then refer to any program that lives and is secured by Blockchain technology. Due to the security and audibility features inherent to the Blockchain, this code can then be used "in governing something important or valuable"(quote from Stark [18]), hence the origin of the term "contract". "Smart legal contracts" on the other hand would refer to certain instances of smart contract code that is made to reflect terms and clauses of traditional paper based contracts. Such smart contracts could be used in reinforcement of legal contracts by enforcing execution of a certain amount of the contractual terms. Finally, "smart alternative contracts" would refer to systems using smart contract code to implement novel and creative ways of doing commerce [18].

It becomes apparent that in the Ethereum sense a smart contract is closer to the "smart contract code" definition above [18]. In fact, Solidity, the main high level language provided by Ethereum, defines in its documentation a contract as follows: "A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum Blockchain." [43]. We take a closer and more precise look at Solidity and its features in Section 2.3.5 below.

2.3.2 The Ethereum State

As mentioned in Section 2.2.2, the Bitcoin state is made of the current set of all UTXOs which are the only value capturing entities in the ecosystem [24]. Ethereum has taken the stance to drift from this model. Instead the Ethereum state, as envisioned by Vitalik Buterin [24] and formalised by Dr. Gavin Wood [44], is made of accounts [24, 17]. As illustrated in Fig. 2.8, accounts themselves have an associated state which consists of the following: (1) a balance, (2) contract code, (3) some storage and (4) a nonce value (an implementation constant with no significant conceptual impact) [24].

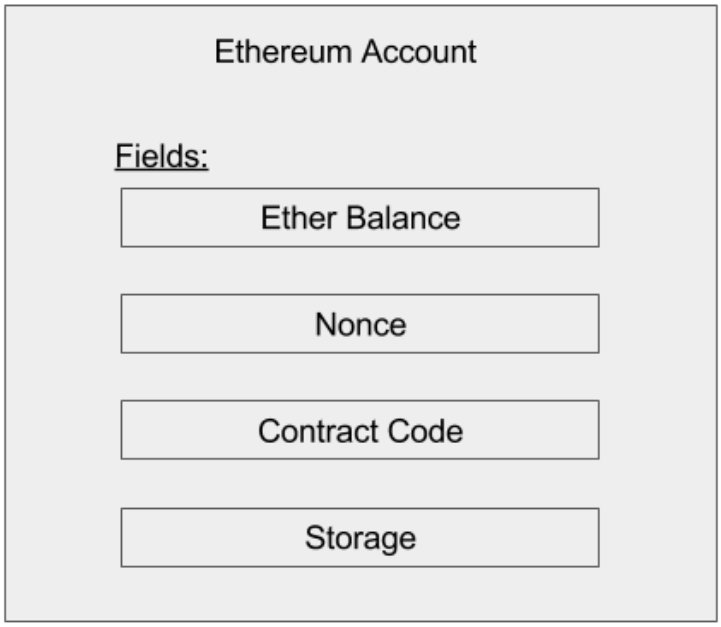


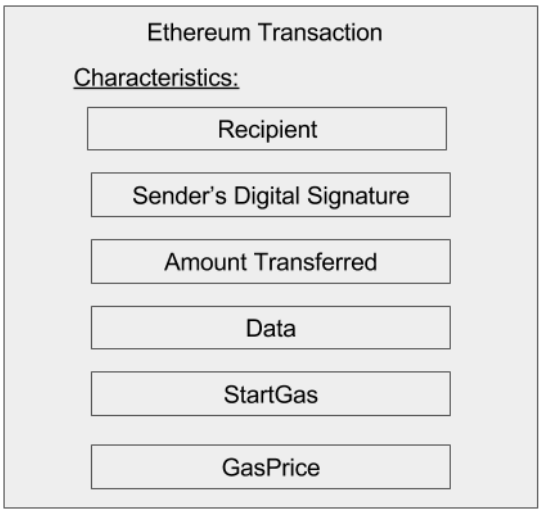
Figure 2.8: Representation of the State of an Ethereum Account [24].

It is also essential to differentiate between the two different types of accounts that exist in the Ethereum ecosystem, namely contract accounts and externally owned accounts. Externally owned accounts are accounts that are controlled by human entities. Analogously to Bitcoin, any person holding the private key corresponding to a certain account has control over it. Contract accounts on the other hand are controlled by their *Contract Code* field (cf. Fig. 2.8), which is empty for externally owned accounts [24, 17].

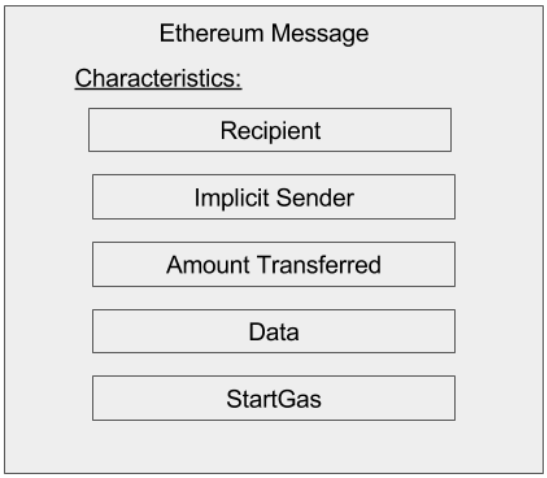
Accounts communicate between each other through messages and transactions, which are also the means through which the Ethereum state changes. For instance, in a simple value transfer transaction where some ether is sent between two external accounts, the transaction will change the Ethereum state by lowering the balance of the sending account while increasing the balance of the receiving account. A

more interesting situation arises when the receiver of a message or transaction is a contract account. This case can trigger the contract's code to execute, which can in turn modify its associated *Storage* field or send further messages to other accounts [45, 24, 17].

The main difference between transactions and messages is that the former consist of a cryptographically signed data package created and sent by externally owned accounts, while the latter are internal objects sent from contract accounts. Both messages and transactions mostly have the same structure however, illustrated in Fig. 2.9 [24, 17]. A noticeable difference is that the "GasPrice" field is missing in the message, which is discussed in Section 2.3.4 [24]. Another difference, according to Ethereum's yellow paper [44], is that the term "transaction" also encompasses the special case of creating new accounts, which is slightly different from message calls.



(a) Representation of the Structure of an Ethereum Message.



(b) Representation of the Structure of an Ethereum Transaction.

Figure 2.9: Ethereum Messages and Transactions [24].

It is interesting to note that in a way this is a more traditional approach to currency and value exchange than Bitcoin: primary entities are account holders possessing balances and other data, and those accounts can communicate with each other through transactions, thereby modifying those balances and data. Where the analogy falls short however, and where the real power of Ethereum arises, is that the account entities do not need to be under human control. Specifically, contract accounts behave as dictated by their predetermined code, which is securely stored on the Blockchain [17, 24].

2.3.3 The Ethereum Architecture

Data Structures

The most important structures in the Ethereum architecture are arguably Merkle Patricia trees and tries. Merkle Patricia trees are a modification of a Merkle tree that allow for cryptographically secure and efficient storage of key value pairs. In fact, the Ethereum state is stored using a Merkle Patricia trie, where the keys are the unique addresses of all existing Ethereum accounts and the values represent the state of those accounts [46]. Other important data represented using this structure are the transactions included in each block and the receipts of those transactions which encode metadata regarding those transactions [46, 47, 48, 49, 24].

The Ethereum Virtual Machine

Once a transaction is received by a contract account, it can trigger the execution of part of its contract code. This code is executed on the Ethereum Virtual Machine (EVM), and is run independently from each node of the system as part of the transaction validation process (similarly to Bitcoin where the validity of each transaction is verified independently by each node) [24]. The EVM is a relatively standard stack based environment, with an associated low level language referred to as EVM Bytecode. Nevertheless, some particular attention is brought in the way it handles memory. Code executing in this environment will have three means of accessing memory, namely (1) the execution stack where execution takes place, (2) a infinite memory store that is local to each EVM execution and (3) permanent storage on the Blockchain as part of the account's storage or balance [24, 48].

Ether: Ethereum's cryptocurrency

Ethreum's Blockchain comes with its own Native Cryptocurreny called Ether. Ether is used both for rewarding miners similarly to Bitcoin, and to prevent system abuses enabled by Turing completeness as described in Section 2.3.4 below. Ether's smallest denomination is called Wei, and 1 Ether $= 10^{18}$ Wei [17, 24].

2.3.4 Blockchain Mechanisms

The fundamental mechanism under which the Ethereum Blockchain operates is quite similar to the Bitcoin Blockchain discussed in Section 2.2.3 [24]. In this section we examine some of the noticeable differences.

Execution Fees

The Turing completeness of Ethereum naturally implies the risk for infinite loops to occur in the EVM. In order to safeguard against such a scenario, the Ethereum protocol implements a fee system tied in with transaction execution. This system works by assigning each computational step of a transaction execution to a certain

value of an entity called Gas. Each gas unit gets associated with a certain amount of ether through the *GasPrice* field in the transaction (cf. Fig. 2.9b). Given that the total amount of gas each transaction can consume is limited by the *StartGas* (cf. Fig. 2.9a) value it provides, the risk of bringing the system to a halt due to an infinite loop is nullified [24, 48].

More precisely, the gas system works by subtracting the Ether amount equal to $GasPrice * StartGas$ from the sender account at the start of the transaction processing. The following execution consumes a certain amount of gas per computational step. At the end of the execution, the value corresponding to the gas consumed is utilised as a reward for the miners. If the execution runs out of gas before it terminates, all the changes from the transaction are reverted and the entirety of the gas is used as compensation to the miners [24, 48, 45, 17]. From this follows the reason message calls do not specify a StartGas value: Ethereum has taken the design approach that gas can only be paid by human held accounts, so any subsequent message calls that the initial transaction causes will continue consuming the initial transaction's gas [48, 24].

State Representation and Consensus

An interesting difference between Ethereum and Bitcoin is in their state representation.While Bitcoin stores all transactions (cf. Section 2.2),Ethereum, (ignoring optimisations through pointer manipulations) stores the entirety of the tries/trees representing (1) the state of Ethereum, (2) the transactions in each block and (3)their receipts [48, 24, 49, 45]. It is interesting to note however that analogously with Bitcoin's Merkle root, the corresponding Ethereum block headers also store the roots of these three trees/tries [44].

Another important point is that Ethereum also uses to some extent blocks that have been mined but later disregarded due to the longest chain rule (or stale blocks). These blocks are added into the Blockchain as "uncles" and are counted in determining the longest chain, thereby both somewhat rewarding miners even on stale block production and increasing security [24, 48].

Finally, Ethash, the mining algorithm of Ethereum is also somewhat different that Bitcoin's approach [50]. Both protocols rely on the same principles, namely increasing a nonce to meet a certain target in the output of a cryptographic hash function. Ethash however is slighly more complex that Bitcoin's algorithm with the intent to make it resistant to specialised software and excessive pool centralisation [24, 51, 48]. In order to do so Ethash relies on a DAG (Dirrectly Asyclic Graph), a large data structure that is periodically changed [51, 50].

2.3.5 Solidity

Overview

Solidity is the main programming language provided by Ethereum. It is compiled into an Ethereum specific low level language understood by the EVM called EVM Bytecode [17].

At its core, Solidity is used by developers to create smart contracts, or more precisely Solidity contracts, which are compiled and stored in the Blockchain in the contract code section of the contract account to which they correspond [45, 43].

The Solidity syntax is based on JavaScript [43, 52]. It's code is organised in contracts, which are quite analogous to classes in Object Oriented Programming(OOP). Just as classes, contracts contain attributes (which will be stored in the contract account's storage field) and functions, which define the logic of the contract and its behaviour upon receiving messages. Moreover, like in OOP, contracts can inherit from other contracts -Solidity allows for multiple inheritance-, and contracts can be abstract [43]. Typical contract syntax is then illustrated in Listing 2.1.

At a higher level, sending a message to a contract account would correspond into calling one of the functions defined in its code [43, 53, 17]. In fact, one of the uses of the data field of the message (cf. Fig. 2.9) is to pass the identifier and parameters of the function called to the EVM [54]. The message will then cause this function to execute, which in turn may (1) modify the attributes of its contract, (2) call functions of other contracts or(3) create entirely new contracts [43, 24, 45].

```
1  contract myContract is myContractParent{
2
3    /*Contract attributes:
4      * typeDeclaration  optionalModifier identifier;
5      * E.g:
6      */
7      uint myAttribute1;
8      string myAttribute2;
9
10     /*Events: Defined in the contract and then called from within a
11     * function body like any other function would be
12     * event (parameterList);
13     * E.g:
14     */
15     event(uint anEventParameter, bool anotherEventParameter);
16
17     /*Function Modifiers : "_" placed where function body is needed
18      * modifier identifier{
19      *  Body
20      *  _
21      *  }
22      * E.g:
23      */
24
```

```
25      modifier myModifier{
26        /*some fuctionality*/
27        _
28      }
29
30      /*Constructor:
31      * Executed once during the deployment of the contract.
32      * Has the same name as the contract
33      * E.g:
34      */
35      function myContract(uint constructorParameter){
36        myAttribute1 = constructorParameter;
37      }
38
39      /*Funcitons:
40      * function identifier( parameterList ) optionalModifier
41      *                          returns (returnValuesList){
42      *       FunctionBody
43      *  }
44      * E.g:
45      */
46      function foo(address anAddress) returns (bool aflag){
47        /* function body*/
48      }
49
50      /* FallBack function:
51      * A contract can have a single fallback function with no
            identifier and
52      * no arguments
53      *E.g:
54      */
55      function(){
56        /*body*/
57      }
58
59  }
```

Listing 2.1: Solidity Syntax Illustration [43].

Features

Noteworthy syntactic features of Solidity are exceptions, function modifiers, events and fallback functions [43] (cf. Listing 2.1).

In solidity, exceptions can take place if certain issues occur with the EVM execution, such as running out of gas [43]. When an exception happens, all changes related to the execution are reverted, and the execution stops. It is also possible to trigger an exception manually in Solidity functions using the `throw` keyword [43].

Function modifiers can be defined once and used in multiple functions. When a function receives a modifier, it's body is inserted where the modifier's underscore "_" symbol is (cf. Listing 2.1) [43].

Events provide a way for Solidity contracts to log information. Event syntax is shown in Listing 2.1. When an event gets called as part of a function execution, its defined parameters are logged as part of the transaction information that is stored on the Blockchain. Through the Web3 Application Programming Interface (API) [53] these logs can be accessed (cf Section 2.3.6), and clients are also able to monitor these events and act upon receiving the corresponding information [43, 53].

The fallback function is a special function defined without an identifier. There can only be a single fallback function per contract and its syntax is shown in Listing 2.1. This function is called automatically whenever the contract account receives a message or transaction that does not specify a function to be called in the contract. For instance, if a transaction is sent to a contract with some ether but with its data field being empty, the fallback function will be executed [43].

It is important to note that at the time of writing Solidity is still an incomplete language under development, and its specifications are drastically changing in short time-scales. For instance, as of the time of writing, there is still no implementation for floating-point numbers. As such it is possible that information on this document regarding syntax specific information might become deprecated or suboptimal some time in the future [43].

2.3.6 Interracting with Ethereum

In order to become a node and participate in the network, one needs to download one of the many Ethereum clients, or the ethereum wallet which comes bundled with a user interface [17, 55, 16].

In order for Dapps to interact with the Ethereum Blockchain, Ethereum provides a JavaScript API called the Web3 API [53]. This API provides a JavaScript object, web3, which can be used from a JavaScript environment to interface with an Ethereum node [53]. Some of the most important tools provided by the web3 API are discussed below.

In order to connect to a contract, the web3 API syntax is:

```
var myContract = web3.eth.contract( abi ).at( address );
```

where abi represents the Application Binary Interface of the contract [56], generally obtainable by any Solidity compiler,and address is the address of the contract. The variable `myContract` will then contain the contract object, which can be used to interact with said contract [17, 53].

The `sendTransaction` function is used in order to send a transaction in Ethereum [53]. It can be applied to a contract object using :

```
myContract.myFunction.sendTransaction(params,txObject,callback);
```

Here, `myFunction` refers to the function in `myContract` to be executed with `params` as its parameters; `txObject` refers to a JavaScript object specifying parameters such as the sender or the receiver of the transaction; and finally `callback` is a JavaScript callback that will be executed when the transaction is sent. This then returns a hash of the transaction it creates, which is used to identify and find this transaction in the Ethereum Blockchain. Finally, it is interesting to note that for simple transactions such as ether transfer, this function can also be used directly [53, 17]:

```
web3.eth.sendTransaction(txObject,callback)
```

The `call` function is used to call a contract function locally, without sending a transaction to the network. This is very useful in scenarios such as verifying contract attributes [17, 53].

The `getTransactionReceipt` function allows to get metadata associated with mined Ethereum transactions. It receives a transaction hash as an argument and returns either `null` if the transaction has not yet been mined or the metadata object if it has. An interesting part of this transaction receipt are the logs for this transaction, which include any associated events as mentioned previously [53, 43].

Finally, the `watch` function can be used in order to monitor if particular events occur in contracts. Whenever an event is logged, it will trigger the watch function to run, executing its corresponding body [53].

2.3.7 Summary

Primary agents on the Ethereum ecosystem are Accounts. These are divided in externally owned accounts (owned by the users) and contract accounts (which essentially are Solidity contracts). Both types of accounts are similar in almost every aspect, except that an external account is controlled by a user while a contract account is controlled by its own code [24, 17].

Accounts can hold funds in the form of Ether (the native Ethereum cryptocurrency). They can also send transactions or messages to other accounts, which may or may not be accompanied with some ether value transfer. In the case of a contract account, the contract's code will determine its behaviour upon receiving a message or transaction [17, 24].

All transactions are permanently, transparently and securely recorded in the Ethereum Blockchain. This process is analogous to the Bitcoin process, with a noticeable difference being that it also stores its state as part of each block. Ethereum transaction execution also requires a fee in the form of Gas. The Gas (which is paid in Ether) used by each transaction depends on the amount of computational steps that the transaction involves, and serves the purpose of preventing abuses such as infinite loops from occurring [24].

Finally, common interactions in Ethereum can be loosely described as follows. User accounts can (1) send ether to other users, (2) send ether to contracts,(3) call some contract function that will get executed or (4) create contract accounts. Contracts will sit idle until a function of the contract is called for execution. While executing, the contract function might (1) call other functions of the same contract, (2) call functions of other contracts (3) send ether to contracts or users or (4) create other contract accounts [17, 43, 53].

Chapter 3

Decentralised Autonomous Organisations

3.1 The DAO Concept

3.1.1 Overview and Definition

Using smart contracts at their core, Ethereum applications are very diverse, ranging from personalised currency tokens to gambling systems [24]. In this project we particularly focus on the topic of Decentralised Autonomous Organisations [21].

As such it is quite important to clarify what the term designates. As a matter of fact, due to the nascent and rapidly changing nature of the space, the concept does not seem to be solidly grounded even within the DLT community. Other similar concepts that have often been used interchangeably are Decentralised Autonomous Companies (DACs), Decentralised Organisations (DOs) or Decentralised Corporations (DCs) [21]. For the purposes of our project we will use the generic acronym DAO to designate any of the above. Overall, based on our investigation [21, 27, 24], our definition of a DAO is as follows :

> A DAO is a software structure that uses Blockchain-safe code at its core in order to formalise, automate and enforce rules and regulations for the governance and/or operations of a human organisation.

Being one of the featured applications of Ethereum, DAOs have also attracted important media attention [24, 22]. As seen in Section 3.2 there are many groups getting involved with DAOs, offering their perspective on how a DAO system should get implemented. Nevertheless, most of these implementations seem to follow a general philosophy that appears to originate from the Ethereum foundation itself [57].

3.1.2 The Foundation's DAO model

Probably the most widely accepted boilerplate for building DAOs originates from the Ethereum Foundation (the organisation responsible for the creation and mainte-

nance of Ethereum [58]). Initially, this took the form of the "Ethereum in practice" blog series by Alex Van de Sanders [59, 60, 61]. Subsequently there was an updated version of DAO tutorials in the Ethereum webpage, nevertheless keeping the same working principles [57].

Essentially, foundation's DAO model can be summarised to the following [57]:

- DAOs are governed by its members, which may or may not include an owner with special privileges.
- DAOs take decisions based on a proposal submission system. Proposals are submitted to the DAO by authorised members and enter a voting period. During the voting period members can vote on whether they are in favour or against the proposal. At the end of this period, the proposal is either executed or dismissed, depending on the voting results.
- A DAO proposal is a data structure containing a recipient address that specifies (1) the beneficiary of the proposal, (2) an amount specifying the funds to be sent to the beneficiary, (3) a hash that encodes any specific actions that will be carried out by the proposal and (4) any other attributes necessary to the voting and verification mechanisms of the proposals.
- In organisations using a shareholder based membership, shares of the DAO are represented by an Ethereum token (also sometimes referred to as currency or coin). These tokens are created and exist through an Ethereum contract that is used to manage their holdings and can represent any value capturing entity the contract is written to reflect [62] .

In the tutorial implementation [57], the most intriguing part from a technical standpoint is arguably the mechanism for proposal execution, so we feel it is necessary to delve more on this point.

In order to understand how this functionality is constructed, it is first necessary to look at the `.call()` Solidity function (not to confuse with the web3 `call` function), which is used when executing the proposal. This is a low-level function that is not yet clearly documented. Though our search [63, 64, 65, 43, 57], its functionality has become apparent:

- The `.call()` function allows for a message (cf. Section 2.3.4) to be sent by the contract to an arbitrary account. In order to do so, `.call()` is applied to the *address* where the target account resides.
- If the `.call()` receives an argument, it will be passed in the constructed message's data field. For consistency with the tutorial's terminology, we will refer to such an argument as the `transactionBytecode` [57].
- In the case of a contract account, this argument can specify the function identifier that is to be run along with any parameters necessary, and this information needs to be encoded in EVM Bytecode.

- It is possible to modify the `.call()` syntax in the following way :

  ```
  someAddress.call.value(someAmount).gas(maxGas)(transactionBytecode);
  ```

  where either/both the `.value` and `.gas` modifications can be added or omitted. If a value is specified, it will specify the amount of Ether to be sent with the message and if the gas is specified, it will designate the maximum amount of gas this call can burn.

- At the time of writing, using Solidity and according to our search, the `.call()` function is the only way for calling an arbitrary function of an arbitrary contract from within another contract.

With this functionality in mind, it is possible to de-construct the mechanism for proposal execution. When the execution is called, an optional `transactionBytecode` parameter can be passed [57]. Two main cases are then identified: (1) If the proposal only involves sending funds, the beneficiary can be any Ethereum account and the `transactionBytecode` needs to be empty. The funds specified in the proposal are sent to the beneficiary. (2) If the proposal needs to encompass some more complex logic, the beneficiary needs to be a contract account and the `transactionBytecode` needs to encode the function identifier (and parameters) to execute in the beneficiary contract. A message is then constructed using the `.call()` function as specified above and the corresponding beneficiary function is run [57].

3.2 Related work

3.2.1 Colony

Colony [66] is a "community management and governance tool" (DEVCON1 conference talk [67]) that is being built on the Ethereum Blockchain. It also "is a platform for Decentralised Autonomous Organisations" as mentioned by one of its founders, Jack du Rose [67].

At its core, Colony aims to pool together a community of people that can aggregate in order to collaborate on projects. Colony aims to disrupt traditional organisational structures and project delivery models by proposing an alternative, peer-to-peer, and flat management system. As such, it moves away from conventional top-to-bottom project administration in favour of a bottom-up approach. Colony's users will have a profile and an overall score that depends on their involvement and achievements on the platform. They will be able to submit project proposals for acceptance by the community as well as aggregate to work on accepted projects [66, 67, 68].

At the time of writing, Colony is still under development, with only few visible signs of activity in the past few months. Colony's white paper (paper traditionally published along decentralised applications explaining the aims, principles and design

of the application [69]) is still to be published. As such, the information gathered in this section is the product of Colony's landing page information and conference talks, and might need revisiting as more information is released.

3.2.2 BoardRoom

Boardroom [70] aims to create a platform providing decentralised governance tools built using smart contracts on the Ethereum Blockchain. Similarly to Colony, Boardroom still seems under development. Although a white paper exists [71], describing the overall design of the Boardroom system, there still remain several unclear implementation design details .

Boardroom's design philosophy goal is to create a "decentralised governance apparatus"(Nick Dodson, [71]) that can be leveraged by DAOs and other structures to automate their governance through smart contracts. Nevertheless, it advertises use cases such as crowd-funding, asset management and equity allocation on top of capabilities such as a proposal submission and voting systems [70, 72]. As such,it remains unclear with the available documentation if creating a Board is the same as creating a DAO under our definition in Chapter 3 .

Nevertheless, two interesting ideas brought forth by Boardroom are:

- The Boards can take a hierarchical structure with sub-comities and executive Boards, therefore allowing for high flexibility in the governance model used.
- Proposal submissions to the Board are typed, meaning that there can only be pre-defined proposal templates.

3.2.3 Otonomos

Otonomos advertises a product made of a dashboard front-end that allows users to manage their company's governance (Shareholding, funding and corporate decision making), via the Ethereum Blockchain. In short, Otonomos offers to its clients the ability to create a DAO representing their corporation, and presens them with a front-end tool for managing it. No white paper or implementation design details are available from Otonomos, but the advertised capabilities of their software suggests that Otonomos focuses entirely on corporate governance, leaving out other potential aspects such as project and employment management [73, 74].

3.3 "The DAO" Incident

3.3.1 Timeline

"The DAO" is a DAO created by Slock.it [75], and has arguably caused the biggest commotion in the Ethereum community since the inception of Ethereum itself.

Launched on April 30th 2016, it benefited in its early days from considerable attention, not only by the Ethereum community but also by mainstream media. In fact, "The DAO" became the biggest ever crowd-funding operation, with more than 150M$ worth of ether raised [76, 28, 77]. Unfortunately, things did not go so well from there.

On June 12 2016, a "recursive call" bug was found and announced by Slock.it, with the founders nonetheless assuring that the DAO funds were safe [78].

On June 17 2016, "The DAO" was hacked, and around 3.6M Ether were stolen, which corresponded to approximately 60M$ based on the Ether price at the time [79, 80, 81].

The events that followed the attack where quite erratic. Furious discussions spawned amongst the community on what should be the response of Ethereum to the situation [82, 83]. Proposals for soft forks and hard forks were made and contested [84, 85]. Hard and soft forks happen in Blockchain technologies following changes in their consensus rules, usually in order to correct for identified bugs and weaknesses [86, 87, 88]. This created a central point of debate as the Ethereum protocol had actually worked as intended. It was a bug in the Solidity contract that permitted the exploit and controversy spawned over whether the foundation itself should intervene in an issue with a mere application of Ethereum [82, 89, 83].

Adding to the turmoil, the attacker also showed himself and addressed the community in a public letter (although it was never confirmed that the letter was genuine and not a fake). He claimed the funds he recovered were his legitimate property (claiming he merely used a "feature" [90] of "The DAO"). He even threatened with legal action if he was denied access to "his" Ether, and offered rewards to the miners that would oppose the foundation's initiatives to remedy the attack [90, 81].

A the same time, the community was attempting to fight back through a so-called "White Hat" [91] attack that was meant to recover some of the Ether by sending it to another child DAO (cf. Section 3.3.2) [91]. Unfortunately, this child DAO was also hacked, by exploiting the same vulnerability as the original [92].

On July 20 2016, a hard fork was launched by the Ethereum foundation in order to invalidate all transactions related to "The DAO" [84]. Even so, there is still part of the community against this change that refuses to switch to the new chain created by the hard fork, thereby creating what is referred to as "Ethereum Classic" [93]. At the time of writing, most of the community has settled to the chain backed by the foundation [94].

3.3.2 Design Principles

"The DAO" follows the foundation's model described in Section 3.1.2 [27]. Some interesting design points of "The DAO" are as follows [27]:

- At its creation it goes through a crowd-funding phase. During this phase, any Ethereum account holder can send ether to "The DAO"s address in exchange for DAO tokens, thereby effectively becoming a "shareholder" of "The DAO".
- It has no specific owner. The implemented decision making process is entirely stake driven, meaning that proposals are voted by all shareholders and their voting impact is proportional to their stake.
- "The DAO" implements a splitting mechanism for protecting its token holders against a "Majority Robs Minority" (Jentzsch, [27]) attack. It consists of splitting "The DAO" by moving one's stake onto another DAO contract, generally referred to as a"child DAO" [92]. Ironically, it is this splitting mechanism that contains the bug that resulted in the "Minority robs the Majority" hack traced in Section 3.3.1, and the technical details of which are discussed below.

3.3.3 The Hack and Lessons learnt

The bug exploited in "The DAO" attack is commonly known as a "recursive call" [78] bug. It is interesting to identify and describe it more in depth here in order to demonstrate a critical type of weakness that we should protect from when developing our contracts. As described by Peter Vessenes in his blog post [95], the bug originates from attempting to send funds subject to a certain condition using the `.call()` function, and updating the condition after sending the funds [95]. It is easier to illustrate this through the example below [95]:

```
1  contract RecursiveBugIllustration{
2    bool condition; //Assume the condition is initially true
3      ...
4      function onlyOnceFundsTransfer(address recipient){
5        if(condition){
6            recipient.call.value(20)();
7          }
8          condition = false;
9      }
10 }
```

Listing 3.1: Illustration of the recursive call bug (adapted from [95]).

This illustration contract intends for `onlyOnceFundsTransfer` to only enable sending funds once, but unfortunately suffers from the recursive call bug. In order to exploit the bug, it is important to notice that the `recipient` address does not necessarily correspond to an externally owned account but may well be a contract account. As such, it may also have a fallback function which will be executed before the `condition` changes to false. This fallback function may in its logic call the `onlyOnceFundsTransfer` function again. In this case, specifically because `condition` has not yet been changed to false, `recipient.call.value(20)` will be fired off

again, thereby making the `onlyOnceFundsTransfer` function to send funds more than once. Depending on the malicious fallback function's code, the attack can be made to be recursive, hence the name "recursive call bug" [95].

Possible protections against this bug include (1) using the `.send()` [43] function, which automatically limits the gas that can be used to 21000, (2) specify a maximum amount of gas in the `.call()` function (cf. Section 3.1.2), (3) update the conditions for sending funds before sending them or (4) implement mutex behaviour [95].

It is interesting finally to identify the points in "The DAO"s code that make up this bug. A simplified version of the attack can be explained using code snippet below, taken from "The DAO"s white paper [27], as described by Phil Daian in a CoinBase article [96]:

```
1  contract ManagedAccount is ManagedAccountInterface{
2
3      ...
4
5    function payOut(address _recipient, uint _amount) returns (bool) {
6      ...
7      if (_recipient.call.value(_amount)()) {
8          ...
9      }
10     ...
11   }
12
13   ...
14 }
15
16
17 contract DAO is DAOInterface, Token, TokenCreation {
18
19     ...
20
21     function splitDAO(
22     uint _proposalID,
23     address _newCurator
24     ) noEther onlyTokenholders returns (bool _success) {
25         ...
26         if (p.splitData[0].newDAO.createTokenProxy.value(
               fundsToBeMoved)(msg.sender) == false)
27             throw;
28         ...
29         withdrawRewardFor(msg.sender); // be nice, and get his
               rewards
30         totalSupply -= balances[msg.sender];
31         balances[msg.sender] = 0;
32         ...
33     }
34
35     ...
36
```

```
37      function withdrawRewardFor(address _account) noEther internal
            returns (bool _success) {
38          ...
39          if (!rewardAccount.payOut(_account, reward))
40              throw;
41          ...
42      }
43
44      ...
45
46  }
```

Listing 3.2: Indentification of "The DAO" vulnerablity (code taken from The DAO white paper, Jentzsch [27], and modified according to Phil Daian's analysis [96])

The parts of "The DAO" code shown above are the ones that could be exploited, and the attacker had to create the conditions and make sure they are accessed [96]. The attack is understood by noticing the following pattern: (1) line 21 is responsible for sending funds to a child DAO (cf. 3.3.2) and the amount of funds sent is determined by the `balances` array that appears in line 26, (2) `withdrawRewardFor(msg.sender);` in line 24 is called before the `balances` are updated and results in line 5 being run, (3) line 5 has the same format as the one seen in the code snippet 3.1 [96]. Essentially, because balances were updated after they are used to send the funds, and because `_recipient.call.value(_amount)()` is used in the meanwhile, the attacker was able to create an attack on "The DAO", recursively transferring funds to his own child DAO before the balances could be updated to prevent that [96, 95].

The events with "The DAO" have shaken the Ethereum community, and resulted in the realisation that writing contracts is not as much of a simple matter as originally thought. As it becomes clear from our previous analysis, contract bugs can be far from obvious, and can have dramatic consequences [96, 85]. This incident has also shown that extreme precautions need to be taken in smart contract development for real applications. Attempts are starting to be maid to identify possible bug patterns and propagate best practices in writing contracts [97], and in our project we will attempt to write our Ethereum contracts keeping these in mind.

Chapter 4

Enabling the Digital Economy

4.1 Design Philosophy

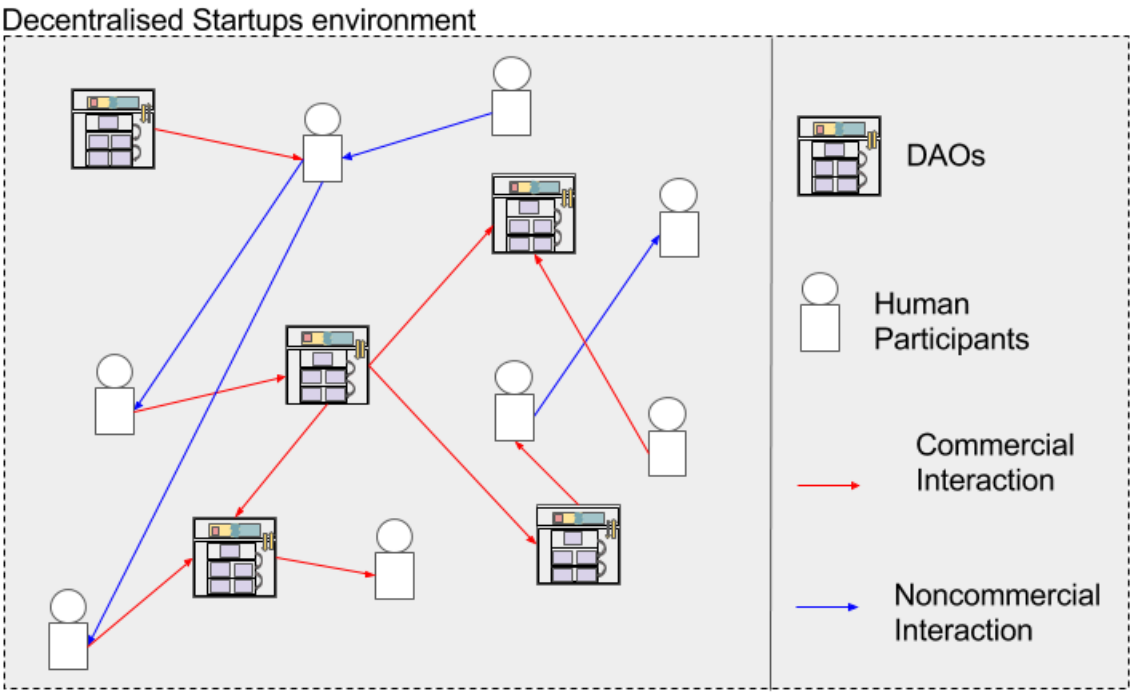
4.1.1 A Digital Economy Ecosystem

So far we have discussed existing and developing instances of DAOs, looking at the different approaches and designs that involved groups follow. Most of them, with the apparent exception of Colony, seem to have the shared approach of regarding DAOs as individual entities. As such, they consider the inception and deployment of DAOs as a goal in its own right. In this section we will discuss the driving conceptual design philosophy that backs the technical solutions that we propose.

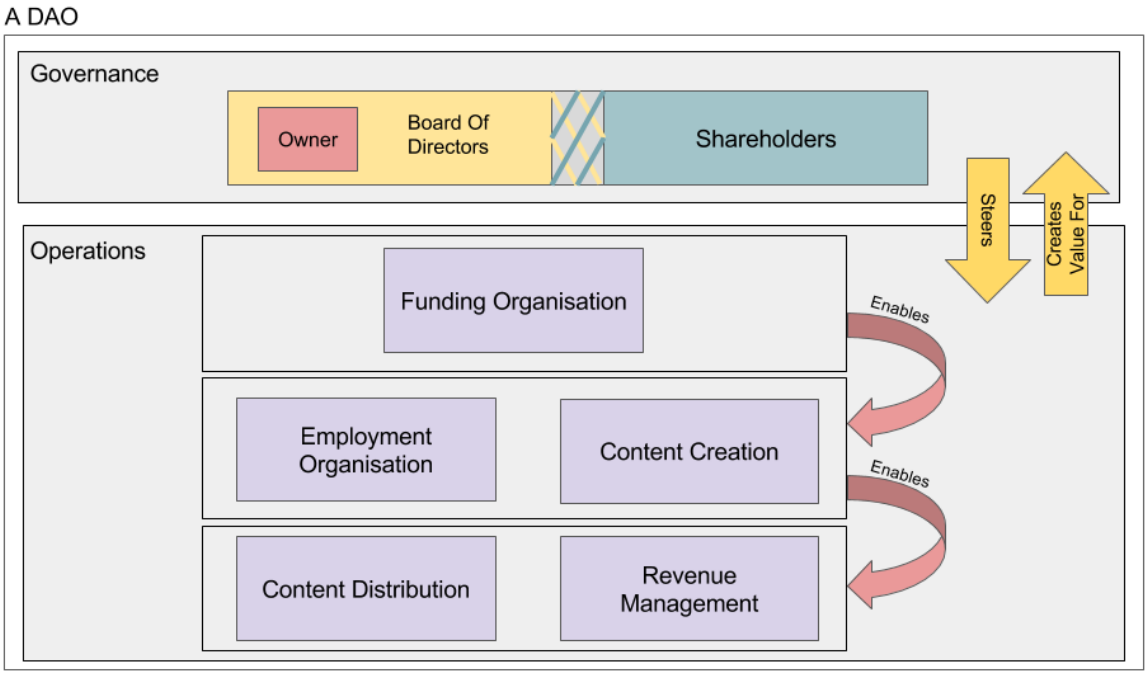
We aim to devise an environment that enables, monitors and facilitates the economic and commercial interaction of its participants. We envision such an environment as a digitalised ecosystem that would enable bringing together all different components of a healthy economy, from raising capital and investments to content creation and product distribution. In this perspective, DAOs are not viewed as individual entities that can stand on their own grounds, but rather as interconnected core components of a wider system that comes together in order to create value for all its participants.

Fig. 4.1 shows the conceptual representation of our solution, with its two main components representing different viewpoints: Fig. 4.1a displays an overview of the Decentralised Startups environment concept while Fig. 4.1b represents a zoomed-in, more precise perspective on the specific tasks of the DAOs within the system.

More precisely, Fig. 4.1a shows our system's participants with high-level interactions it enables them to engage in. As seen in the figure, the two main entities involved are users and DAOs. A user is essentially a person using the system, although conceptually users do not need to represent a single individual. Instead, groups of people or complete organisations can also decide to join the Decentralised Startups ecosystem under a single user's banner. DAOs are entities that are created from within the system, and are effectively the bodies that enable and organise the economic interaction between the users.



(a) Participants and system interactions.



(b) The role and organisation of a DAO.

Figure 4.1: Conceptual representation of the Decentralised Startups environment.

Fig. 4.1b shows the different components that make up our DAOs, indicating they way they organise the above-mentioned economic interaction. As we can see, there are two main components that become apparent:

1. DAO governance is the part of the DAO where decision making processes as well as the specific roles of the DAO's stakeholders are defined. Through the established rules and protocols, the stakeholders can guide the different parts of the DAO's operations. They can, for instance, post employment offers for contractors to create some content, decide on which specific contractor to hire and then launch the distribution of the product.
2. DAO operations refer to the part of the DAO that defines its different capabilities and workflows. Initially these consist of a set of funding operations that will enable content creation through various employment processes. Once content is produced, the DAO also defines the means by which (1) the content is sold and (2) the revenue is redistributed to its stakeholders.

An interesting perspective that emerges from this discussion is that DAOs do not need to replace traditional corporations or become their digital equivalent in our systems. They certainly can: if the users participating in the system group together independently under a governance scheme of a DAO, then the DAO essentially becomes a digitally self-managed corporation. That does not mean however that existing corporations must have their DAO digital equivalent. Corporations can also join the system as users, and use a DAO to enable their digital interaction with our ecosystem, while keeping their full governance and operations "off the chain". For instance, if a corporation only wishes to sell its products to our users it can do so through the following steps: (1) join as a user, (2)create a DAO of which it is the owner, (3) upload its products for sell within the environment. The corporation does not need to organise its operations through the DAO, and, from this perspective, the DAOs merely become a technological body that allows for commercial activity within Decentralised Startups.

It is finally worth placing our line of work in the current DAO landscape seen in Section 3.2. With reference to Fig. 4.1, Boardroom, Otonomo, "The DAO" and the foundation's model appear to principally be concerned with the governance and funding aspects of a DAO. Colony on the other hand seems to be more concerned about the content creation, employment and content distribution aspects while denying traditional governance structures (cf. Section 3.2). Our design philosophy is then similar to the mentality presented by Colony (creating a content production and interaction environment) but it conserves the more traditional view of economic and corporate organisation seen in the other solutions. As such, we believe that Decentralised Startups strikes midway between Colony and the other solutions in the field, thereby profiting from advantages of both perspectives.

4.1.2 Software Components

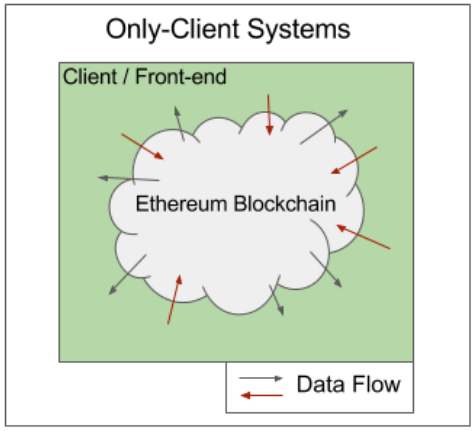
It is important at this stage to identify the different software components and lines of work that implementing our solution requires: the portal code and the contracts code. The portal code includes all the front-end and back-end functionality hosted on central servers. The contracts code consists of the Solidity smart contracts that empowers the DAOs (and that once deployed will reside on the Ethereum Blockchain.). In essence, the portal code will enable the interaction of the users with the contracts code.

Both components present interesting development challenges and opportunities. In developing contract code, one needs to creatively exploit the Ethereum technology to devise DAO workflows and capabilities that can cater to the complexities of real-life scenarios while not sacrificing user experience. In developing portal code, one needs to consider how to interface with the Blockchain in order to harness its potential and enable an efficient interaction with end-users in an intuitive and user-friendly manner.

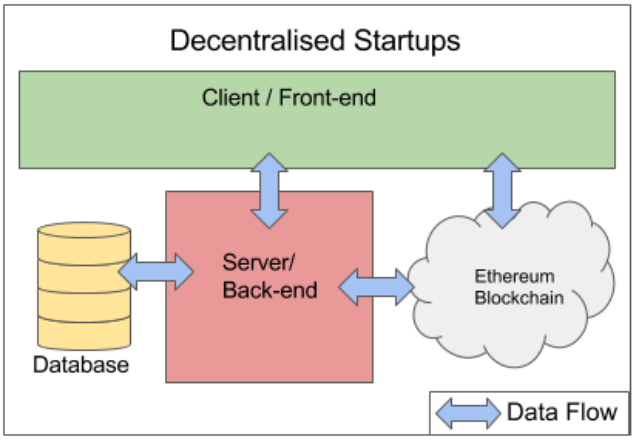
An important point here is that while we initially considered these as entirely distinct components, we found that the full power of the solution could only be obtained by considering them simultaneously. We then realised that our initial approach resulted from an existing preconception, namely that portal code was merely supposed to be a front-end support for smart contracts, and that the smart contracts should be self sufficient units representing the entirety of the required logic.

We have moreover realised that this idea originated from the previous work in the field. All of “The DAO”, Boardroom, the foundation’s model and Otonomos seem to treat their contracts code as an entirely independent unit making up a DAO, with some solutions such as Otonomos also offering a front-end to monitor this DAO [27, 57, 73].

In Decentralised Strartups, as illustrated in Fig. 4.2, we have decided that using a centralised server as a core part of the solution is necessary to create the envisioned environment. As it appears from the following sections, using some server-side computation while focusing on how we can create synergies with the Ethereum Blockchain will allow us to get the best out of both worlds. Our aim is to deliver the full power of the Blockchain to the end user in a user-friendly manner, and we believe that the traditional approach of just providing a front end for the Blockchain database is insufficient (cf. Section 4.1.3) .



(a) Client-Only architectures traditionally used [27, 57, 73].



(b) Decentralised Startups: server-side computation complements the Blockchain functionality.

Figure 4.2: Client-Only systems vs Decentralised Startups.

4.1.3 The need for a user friendly experience

Looking at "The DAO" [27], the most widely known DAO created in the Ethereum ecosystem, and the Foundation's familiarisation tutorials [16], we have realised that current smart contract and DAO implementations tend to have laborious user experiences that are less than adequate for the average user. In order to illustrate this point, we can take a closer look at the different steps a user would need to take to use such schemes.

At the current state of affairs, if one wants to connect or interact with an Ethereum contract, he needs to go through the following process. Other methods exist, but the outlined steps make up the least technical way we have identified, as it does not involve the use of the command line [57, 16, 55]:

1. Download the Ethereum Mist wallet [55].
2. Obtain the address of the DAO contract in question (through any means necessary).
3. If the contract creator provides it, get the contract's ABI (Application Binary Interface), a compact interface for the contract [56].
4. If not, obtain a copy of the original source code (again, by whatever means necessary), and use a Solidity compiler (such as the Online Solidity compiler [98]) in order to obtain the contract's ABI.
5. Go to the wallet in the "Watch Contract" [55] section and paste the obtained ABI and address to the corresponding fields.
6. Monitor the contract through the provided User Interface (UI).

In the particular case of DAOs (assuming that the foundation's model is followed), things become even more convoluted because of the way proposals are approached. In order to better illustrate this point, it is interesting to consider the workflow associated with the submission and monitoring of a proposal having some logic attached to it. It is also important to note that this information was not trivial to understand from the available documentation, and it required a fairly good understanding of how Ethereum operates [99, 57, 27]:

1. Connect to the contract of the relevant DAO as described above.

2. Write the contract (call it contract A) that will encompass the logic of the proposal in Solidity. Depending on the logic needed, this can be no trivial matter, even for experienced developers. An example proposal can be found on "The DAO"s white paper [27].

3. Compile and deploy contract A to the Ethereum network. The actual proposal that will be submitted will consist of a function (call it function "`foo`") in contract A. This can be any function in the contract and can contain any kind of logic.

4. Obtain the Bytecode of `foo`. As specified by the foundation's tutorial [57], this can be obtained by faking a transaction on the mist wallet that would call this function. Just before sending the transaction, the Bytecode of the function becomes visible and can be copied down, at which point this fake transaction can be stopped from being sent to the network.

5. Call the proposal submission function through Mist [55] by providing Bytecode and the other required parameters. The proposal will then be in the list of proposals to be voted for execution.

6. When the proposal is called for execution, `foo` is called and executed. As `foo` is just an arbitrary contract function, the proposal itself can actually consist of any logic that can be encoded in a Solidity contract.

7. In order for anyone to monitor the proposal after its initial execution, he would again need to connect to the relevant contract in the standard way.

The above workflows clearly lack a satisfying user experience, the illustrated DAO process in particular is fairly developers-only oriented. We do understand that some of these originate from restrictions related to the Homestead release (one of the development milestones set from the foundation) which is still not aimed at mass adoption [17, 100, 45].

Nonetheless, we believe that in order to truly illustrate the disruptive potential of the Blockchain, showcasing a more satisfying user experience is necessary. As such, in our project we will attempt to provide a system that simplifies the above workflows, making our solution much more accessible to the average user. As a side note, it is

important to point out that Colony, Otonomos, and perhaps at a lesser extent Boardroom appear to be aiming to such a goal as well, although it is still largely unclear how this is approached in their implementation designs and philosophies.

4.1.4 Summary of Design principles

Before moving forward to the more technical aspects and the implementation design of our solution, it is important to summarise here the different design principles that we have been following in our approach.

A full environment, not just a DAO

As seen in Section 4.1.1 we aim to create a full environment that organises, monitors and facilitates the economic interaction of its participants. We therefore aim for our DAOs to be functional as a part this wider ecosystem rather than individual self sufficient entities. This also reflects in our contract code development, where we sometimes make the concious decision to take some of the functionalities out of the contracts and transfer them to the environment (cf. Section 4.4.1 on User Accounts).

A tool for the masses

Another important driving point for many of our implementation decisions is that we wanted to create a solution as appealing to the average user as possible (cf. Section 4.1.3). We therefore always strive to provide the easiest and most familiar user experience.

Simplicity is key

We also always strive towards simplicity in our contracts and solutions. As seen with "The DAO", having big, complex logic in the contracts can lead to unexpected loopholes arising that can have catastrophic consequences [97, 96]. As also suggested by the Solidity documentation and by Peter Vessenes in his blog posts, keeping contracts as simple as possible helps in mitigating such a risk of exploitable bugs [97, 43].

Moreover we believe it is entirely possible for complex processes to lead to issues even in the case where there are no exploitable bugs in the contracts. In fact we expect that if the implemented workflows are too convoluted (like the examples seen in Section 4.1.3), it would be easy for malicious participants to cheat the system by relying on human overlooks or misunderstandings (rather than code exploits).

Types are essential

Related to the previous idea of simplicity is the notion that we will aim for typed and pre-defined interactions to take place. This is a notion that we have first seen in Boardroom as discussed in Section 3.2 . For instance, we will not allow arbitrary

proposals to take place (cf. Section 3.1.2), but rather provide a set of predetermined proposal types, therefore contributing to clearer workflows and capabilities. In the same spirit, the set of available DAO types is also predefined (cf. Section 4.2).

Some form of centralisation is acceptable

In many cases, enabling solutions that stick to the above-mentioned design principles required more than that could be offered by a fully client-side / Blockchain solution corresponding to a fully decentralised application (cf. Fig. 4.2a). We have therefore decided that having a central server providing functionality that complements the smart contracts is acceptable and desirable, as long as the key benefits inherent to a DLT system are conserved (cf. Section 4.1.2).

4.2 Platform components

4.2.1 General Workflow Organisation

Decentralised Startups in based on the idea that DAOs are empowered through Blockchain technology to organise the economic interaction between the system's participants. The general workflow organisation of our solution is represented in Fig. 4.3 below:
The easiest way to understand the structure of our solution is by following the different steps depicted in Fig. 4.3. Walking through this diagram, we will now describe how each step is achieved as well as the resulting benefits to our system's stakeholders.

Step 1 : Creation and Funding

Initially, a DAO can be created from any user on the ecosystem by selecting one of the predetermined DAO structures further described in Section 4.2.2. The initial governing body can fuel the DAO with funds or expose it to external capital raising. In the later case, the DAO will be advertised as looking for investment through the system's opportunity discovery mechanisms described in Section 4.2.5. Investors will then be able to become its shareholders in exchange for funding and, depending on the scheme used, become entitled to various benefits and powers.

This mechanism both benefits to entrepreneurs and investors looking for opportunities: From the startup founder's perspective, it allows for efficient and rapid access to funding, with the smart contract largely eliminating the need for tedious traditional procedures and paperworks. From the shareholders perspective on the other hand, this system allows to guarantee that the predetermined shareholder rights are enforced, as they would be encoded in the DAO's code and secured by the Blockchain.

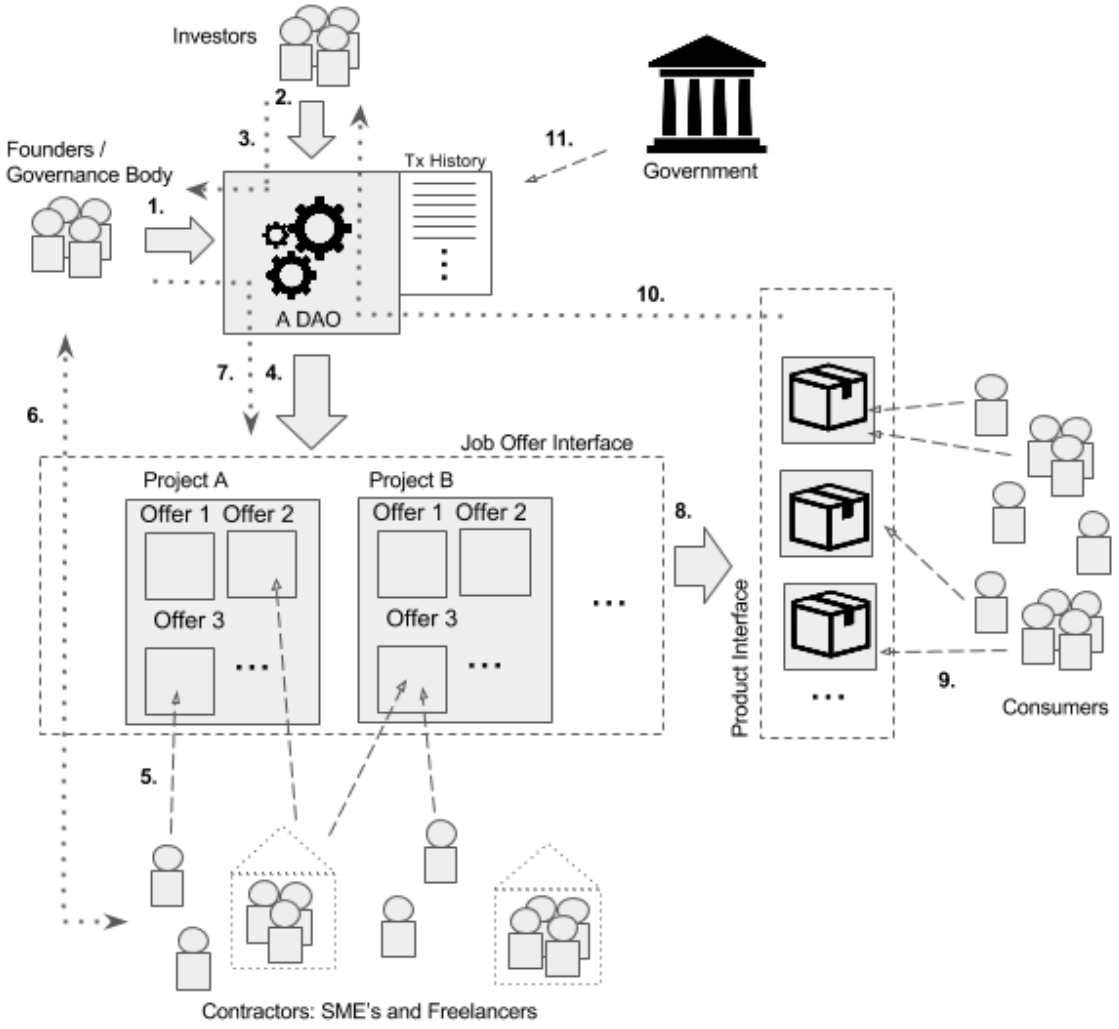


Figure 4.3: Workflow Diagram. *(1) Creation (2) Funding (3) Funding influences Governance structure (4) Employment Offer creation (5) Applications (6) Communication during selection process (7) Decision on candidate acceptance (8) Content creation and Distribution (9) Product Purchase (10) Revenue gathering and redistribution (11) Taxes based on transaction history.*

Step 2 : Contracting for Content creation

Once the necessary funds have been gathered and the DAO's balances are full, they can be used to create employment opportunities. Under the control of the governing body, the openings are created and aggregated together through the various projects the organisation engages in. The different offers concern a particular task and follow one of the employment schemes described in Section 4.2.4 . Once recruiting is enabled, any user in the system can find and apply for these opportunities (cf. Section 4.2.5).

During the applicant selection period, information and communication channels are available in order to guarantee that the successful candidates are adequate for the task at hand. A reputation system that complements the smart contract functionality also exists in this purpose. This system is built on a feedback loop mechanism as described in greater detail in the following sections.

Once employed, the contractor carries out the tasks described in his agreement. In case the organisation is dissatisfied, the contractor can be laid off, with the position becoming open again for new contestants. Dishonest behaviour in this context is again prevented thanks to the feedback and incentivisation mechanisms in place (cf. Sections 4.2.4 and 4.2.5).

This workflow creates a mutually beneficial environment for the involved parties, with outsourcing the different tasks becoming easy and efficient: On one hand, the concerned organisation benefits from an unbound pool of talent that can discover and engage in its projects. On the other hand, contractors such as freelancers and SMEs also benefit, as they get access to a channel on which they can discover opportunities, and through which they can compete with other actors on an equal footing.

Step 3: Selling and Miscellaneous

The projects carried out by the contracted users can result in the production of digital content. This content may in turn be made available to the users through digital distribution channels. Finally, potential customers can decide whether to purchase the products. Revenue streams from these purchases are then channelled back to the corporation's funds and distributed to the stakeholders based on the rules of the DAO smart contracts in place. These funds can also be re-invested to create new projects, thereby closing the cycle of the the DAO's operations.

Finally, throughout the process, and due to the inherent properties of Blockchain technology, the entire history of transactions and doings of the DAO is transparently available. This can be readily used in auditing processes by the Government or the investors. In the investors case, this auditing power can be used to monitor the financial progress and governance decisions of the DAO as a help for investment decisions. In the Government's case, a key benefit is that the transaction history can be used for tax auditing purposes in the combat against tax fraud.

In order to empower these operations, we are using various schemes which are described in the following sections. These schemes contain all the necessary logic for the operations while offering a discrete set of possibilities, in line with our principle of a type-based approach.

4.2.2 Governance Organisation

The DAOs created in the Decentralised Startups environment are part of a set of predetermined types. We are currently considering three main types, private, corporate and public, as defined by their associated governance schemes shown in Fig. 4.4. These are inspired from the examples and processes depicted in Boardroom, "The DAO" and the Ethereum foundation, and are elaborated upon and adapted to our designs [57, 27, 71].

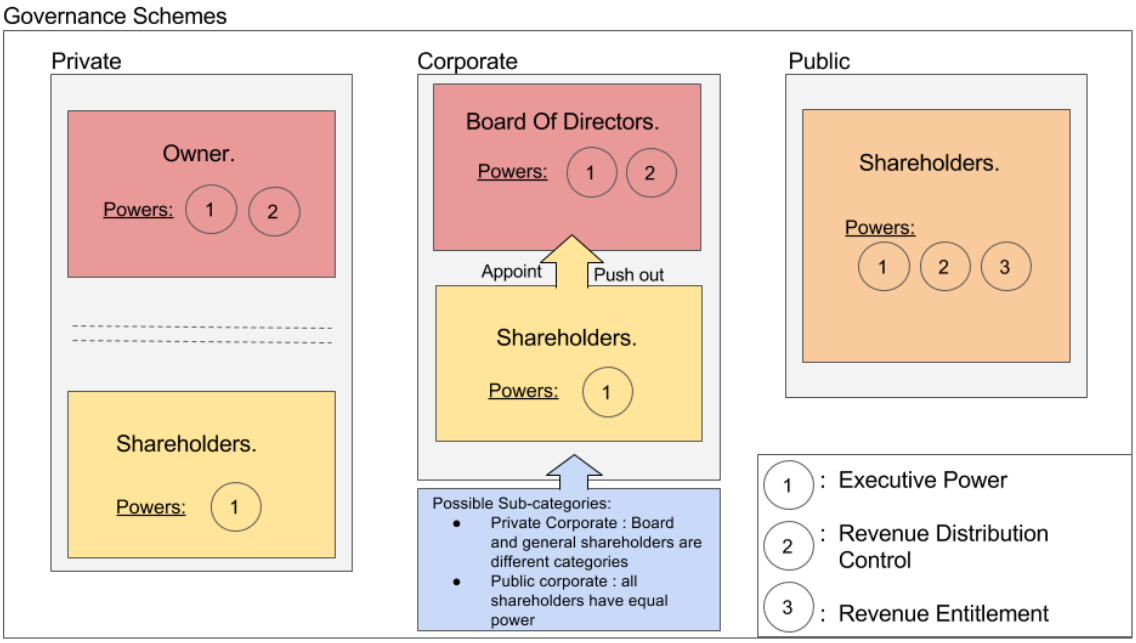


Figure 4.4: DAO Governance Schemes.

The three schemes mentioned mainly defer by the way their executive decision process works as well as the way their funding mechanisms affect their governing bodies. Once an executive decision has been taken, most of the remaining workflow depicted in Fig. 4.3 remains unchanged under each scheme.

The main executive decisions, which are mostly embodied and enforced by smart contract functionality, are the following:

- Enable/Disable funding and shares generation.
- Enable/Disable the recruiting processes.

- Create/Remove proposals (employment offers) as part of projects.
- Hire/layoff contractors for the advertised offers.
- Rate and comment on contractors as defined by the feedback mechanism processes (cf Section 4.2.5).
- Enable/Disable selling capabilities and expose products for sell.

The Private scheme

The private scheme essentially refers to a DAO with a single owner that has all governance power. As such the owner takes executive decisions on his own with no other user having the power to mitigate or alter the decisions.

A form of shareholding does exist under this scheme but does not grand any executive power. Users that send funds to a private DAO become its shareholders and are entitled to the revenue redistribution of the products sold by the DAO. The total amount redistributed is determined by the owner, and the specific quantity received by each user also depends on the number of shares they own.

Consequently, users should be aware of the risk represented by investing in a private DAO. In order to make an informed decision, one is able to use the built-in rating mechanism and the transaction history record to assess the DAO, as well as investigate its owner's records through his user profile.

Possible use cases for a private DAO are : (1) An "off-the-chain" SME that wants to sell its products through our portal can create a user account and a private DAO that it uses as a shell for vending its products. (2) A freelance entrepreneur that would require some quick funds to accomplish a specific project but does not necessarily intend to create a fully fledged corporate body.

The Corporate scheme

The corporate scheme is based on the idea that users sending funds to the DAO become shareholders, and then shareholders assign the Board of Directors that will have the executive power. Each member of the Board is elected through voting of the shareholders, with each vote being weighted by the amount of shares it represents. Through a similar process, Board members can also be voted to step down.

Once the Board is in place, each member can vote for the different executive decisions, with each vote having equal weight. Revenues are distributed in a way similar with the private DAOs, but shareholders have the power to intervene if they are unsatisfied with their governance by changing Board members.

This scheme allows for the biggest flexibility. A standard use case would be a group of co-founders that initially come together to create a DAO, and then require further funding in order to grow their company.

At this stage it would be possible to imagine different subcategories, somewhat filling the gap between the functionalities offered by the private and corporate schemes. For instance, there can be a system where there are two categories of shareholders, with one being entitled to executive power through the above mentioned mechanism and the other being similar to private DAO shareholders. Interestingly, it seems that Boardroom tries to account for most possibilities of Board and governance structures, and their tools could perhaps be used in the future. At this stage however, such level of complexity considered out of scope and therefore the concept is not further analysed (cf. Section 6.1.2).

The public scheme

The public scheme relies on the idea that shareholders have direct power and control over the executive decisions. This is vastly based on the models used in the Ethereum foundation's tutorials and "The DAO" [27, 57].

Similarly to the previous cases, users become shareholders under this scheme by sending Ether to the corresponding DAO. They then have voting power that is proportional to their number of shares, and each decision is made through a voting process.

Possible use cases that could fit this governance model would be the following: (1) A crowd-fund with humanitarian purposes where users come together to gather capital and then vote on which humanitarian missions to donate. (2) A scholarship system where donors would create a capital pool and then vote on which student candidates should receive scholarships.

4.2.3 Detailed Employment Workflow

There are two main aspects of how employment and contracting is organised through our DAOs. The first consists of the different contract types that define the possible employment schemes. The second, briefly mentioned in Section 4.2.1, consists of the protocol that that is attached to the employment cycle. In this section we will explore more in detail the latter while the available schemes are discussed in Section 4.2.4.

A more precise representation of the employment protocol under Decentralised Startups is represented in the flow chart in Fig. 4.5.

As seen in the figure, all employment contract life-cycles start with a candidate selection process. If an escrow scheme is used (cf. Section 4.2.4), the process is more complex, as an escrow needs to be chosen on top of the candidate. The escrow also

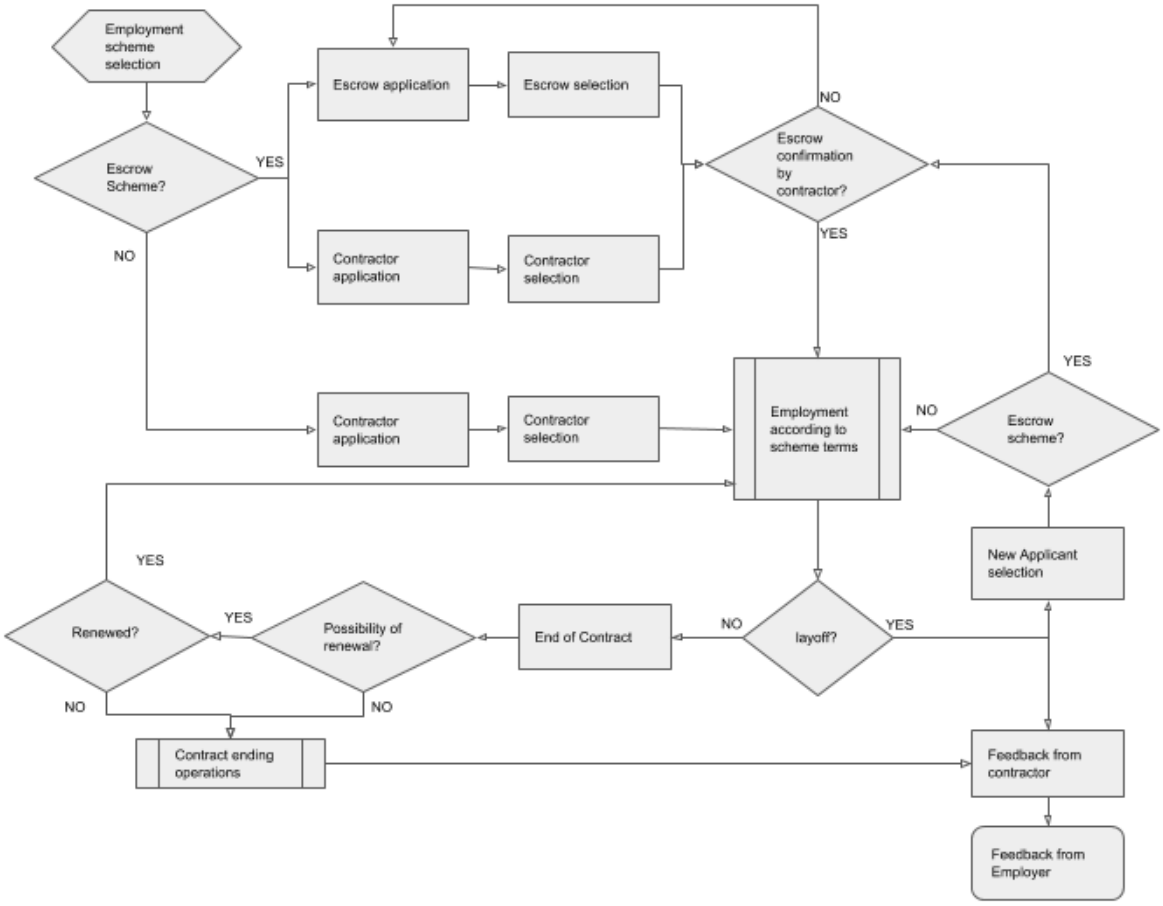


Figure 4.5: Employment organisation flow-chart.

needs to be confirmed by the selected candidate before the contract gets validated in order to mitigate the risk of escrow bias.

Once the different parties are confirmed the contract enters its main execution phase, with the execution details dependent on the selected scheme. In all cases there is the option of early termination of the contract, and some cases also allow for an easy renewal of the contract before its expiry (cf. Section 4.2.4).

At the end of the contract life-cycle, all parties involved need to supply feedback on the system about their counterparts which builds into the reputation system discussed in Section 4.2.5.

4.2.4 Employment Schemes

As seen in the previous section, our employment protocol is heavily reliant on the contracting scheme used. The main different schemes (or types) available to DAOs are the One-off scheme, the Salary scheme and their Escrow-based variations. An employment offer under one of our schemes is the typed equivalent of proposals in the "traditional" DAO structures seen in Chapter 3. These open-ended proposals conceptually represent a more ground-up approach where the contractors make up the proposals and submit the for debate. We take the stance of a completely typed system, where a more top-down approach is used (perhaps also more similar to traditional corporate structures): The company makes up the contracting proposals in offer and contestants apply to the different slots.

Fig. 4.6 gives an overview of what these entail.

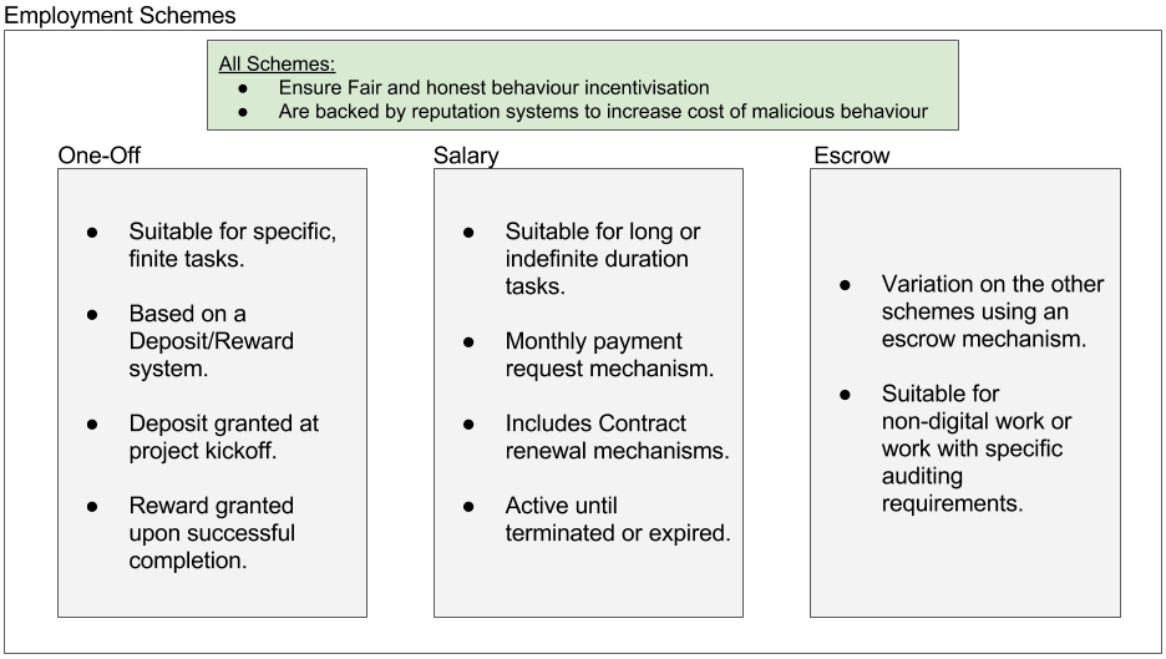


Figure 4.6: Available employment schemes.

One-off

A One-off contract as the name indicates is a type of proposal that consists of a one-off employment offer. It specifies a task to be accomplished, a deposit amount and a reward amount. The basic idea revolves around a two stage payment process: At the beginning of the contract, the contractor receives a deposit to cover any running costs and/or living expenses. At the end, upon consent of both parties for the successful completion of the task, the reward is also released to the contractor.

With this system, possible abuses are (1) contractors that take the deposit and never complete the task and (2) companies that pay the deposit but never release the reward. In order to protect against such abuses, the safeguards in place are the reputation system described in Section 4.2.5 and potentially selling restrictions on products resulting from unpaid contractor's work.

Good use cases for this system would be any finite short term tasks with a tangible outcome at the end. For instance, if an SME would like to increase its digital exposure, it can use this scheme to contract a web developer for making a website. Upon contracting, the developer will receive the deposit for his running costs during the development phase and upon completion of the website he will receive the remaining reward.

Salary

A salary contract represents a more traditional employer-employee relation between the DAO and the contractor. The contract defines a given salary the employee can withdraw every month as well as well as the task the employee needs to fulfil. These contracts have an expiry date, after which they can be renewed or terminated, but may also be terminated at an earlier date if necessary.

Possible abuses are very similar to the previous case, with the company that can potentially run out of funds to pay the salaries and contractors that can receive the salaries without completing their tasks. Nevertheless, it is important to note that this scheme is overall safer than the previous one. Contractor abuse may be discovered at an early stage which would result in a lay-off without loosing a full deposit worth of Ether. Company abuse is also prevented because, as long as enough funds are available in the DAO, payment is guaranteed by the contract. Nonetheless, the safeguards discussed previously are kept in place for an extra layer of security.

Use cases for this scheme are any open ended or long-term work to be fulfilled. Continuing with our previous example, if the SME with its new website ends up growing significantly, it might require a web portal maintenance engineer, an open-ended position which can easily be represented through a salary scheme.

Escrow

Escrow schemes are variations of the two previous types. An escrow is an unrelated third party that will serve as an arbitrator and confirm whether the terms of the contract are being fulfilled [101]. In exchange for this service, the escrow is entitled to a small fee. In a One-Off escrow contract, the escrow decides when the task of the contractor was successfully completed and releases the funds to the contractor. In a salary escrow agreement, the escrow has the power to stop the contractor's ability to withdraw salary if the contractual terms are not being fulfilled and holds a certain amount of salaries worth of funds available for release in case the DAO runs out of funds.

Escrows add more generality and an extra layer of complexity and security in the workflows. Through an escrow, the parties engaged in the agreement further eliminate the element of trust for keeping onto their contractual agreements. Nevertheless, escrows still represent a risk of abuse through bribing methods, which is why we believe a rating system needs to be in place for escrows as well.

Possible cases where escrows might be useful are (1) where a company would like to ensure that its salary agreements are being fulfilled without having to watch over them closely, (2) where the parties involved feel an arbitrator is more appropriate than mutual consent and (3) where expert arbitration is necessary in order to determine if a task is adequately completed. For instance, if a party is contracted for producing software by a DAO with non technical expertise, a escrow arbitration scheme might be used in order to ensure the software produced is of good quality.

4.2.5 Miscellaneous

The reputation system

As briefly mentioned in the previous sections, our ecosystem is reliant on a reputation system in order to ensure the integrity of its participants. A similar idea also seems to be in place with Colony [66].

A key reason for such a system is that although Ethereum contracts are great in enforcing rules and regulations that are expressed directly within the contracts themselves, they are often lacking for enforcing "off-the-chain" behaviour. A premium example of this is, as mentioned in the previous section, a contractor that receives the deposit for accomplishing a certain task, and then immediately disappears.

The reputation system is in place so that, if such behaviour occurs, it will punishable by bad reviews and ratings which will then disable the malicious user form performing such a stunt again. In fact, we believe that the reputation system will result in the following dynamic, which will eventually completely nullify the cases of fraudulent behaviour:

- The more reputation and positive reviews a user will accumulate, the more likely he is to be picked as a contractor.

- Important, well paid tasks will attract and select adequately reputed users which will then have proven their integrity and have spent significant hours working on the system.

- New users which wish to become contractors will therefore be constrained to start with smaller jobs on average in order to build a reputation.

- If these users decide to engage in fraudulent behaviour they will get penalised by the reputation system before they can access the important sums of well paid positions.

The main idea followed then is that if someone has built a strong enough reputation that would allow him to compete for large, well paid task (that would make fraud appealing) he would be dependent enough on the system that engaging in fraud would not be beneficial for him.

The three main identified types of reputations that then need to be maintained by the system are the following: (1) DAO reputation, (2) Contractor reputation, (3) Escrow reputation. Practically, such a system could take the form of a rating mechanism where 0 stars is the minimum score and 5 stars is the maximum. The star ratings also need to be accompanied by comments in order to give context and more information about the rating. As seen in Fig. 4.5, these data are supplied to the system by contractors and DAOs when an employment contract terminates.

Opportunity discovery system

An important advantage of the Decentralised Startups ecosystem is that it creates an environment where SMEs and freelancers can cooperate with each other and reach consumers and investors interested in their business. In order for such a beneficial dynamic to be established however the need for information scouting channels where the opportunities can be easily found and accessed is paramount.

As such, our solution uses three main opportunity discovery mechanisms which are depicted in Fig. 4.7: (1) Investment discovery, (2) Employment Offer navigation and (3) Product browsing. All of these three systems aim to provide easy access to their corresponding information, thereby providing support to the full ecosystem.

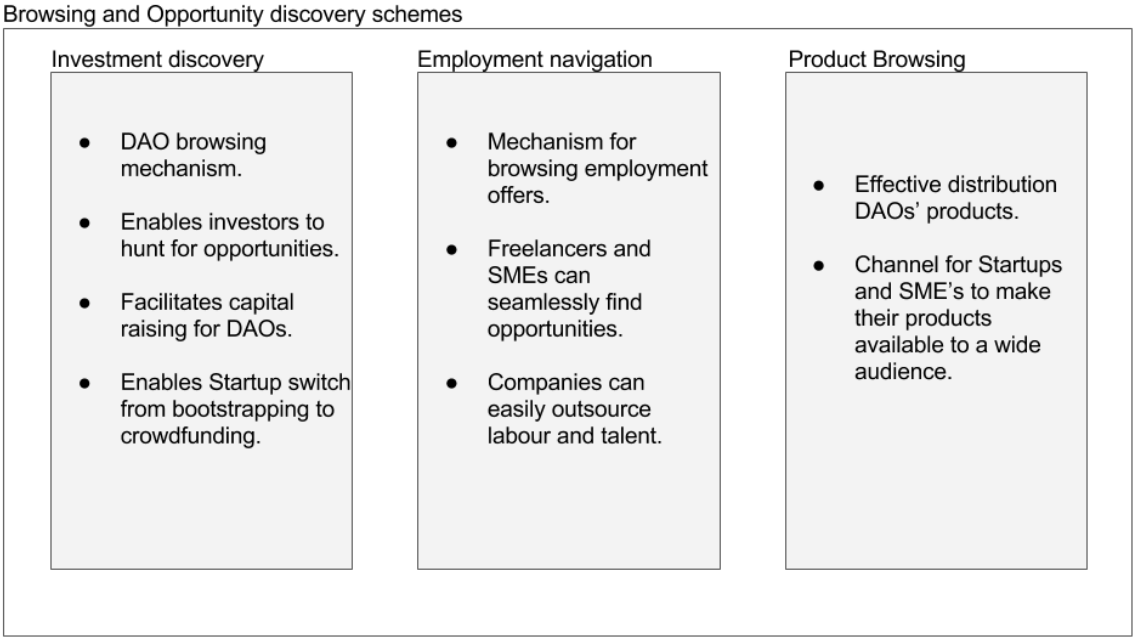


Figure 4.7: Opportunity Discovery mechanisms.

4.3 Constructing the Contracts

4.3.1 Overview

Smart contracts (or more precisely, smart contract code -cf. Section 2.3.1-) are at the heart of our solution. It is the benefits of Blockchain technology inherent to the contracts that allow for the secure and transparent execution of the complex economic interactions offered by Decentralised Startups.

The contracts in our system then consist of Solidity contracts for the Ethereum Blockchain. Their role is to define the core DAO functionality that is reflected and enhanced by the web portal. We also believe that an important guideline to follow is that all mission critical interaction should be securely executed from within the Blockchain while all other interaction should be ensured by the rest of the system. Our goal through this approach is to ensure secure and transparent interaction of our participants while (1) minimising gas costs associated with running a DAO and (2) minimising the risk of complex exploitable loopholes that could arise.

An important takeaway from this discussion is also that contracts discussed in a sense represent the Blockchain projection of the full DAO construct. As also mentioned previously, the role of this projection is securing the key commercial interactions that can occur throughout the ecosystem rather than being fully independent products. A different perspective on the same point is also that if someone were to attempt interacting with these contracts without going through our web portal he should be able to, but he would only be able to access the primitive , raw economic interactions of the DAOs.

In reflecting upon and investigating different contract structures and designs our starting point and main inspiration were the DAO contracts presented by the DAO foundation as well as "The DAO" contract [57, 27, 59, 60, 61] (cf. Chapter 3). In the following sections we present some of our resulting thoughts.

4.3.2 Design and structure

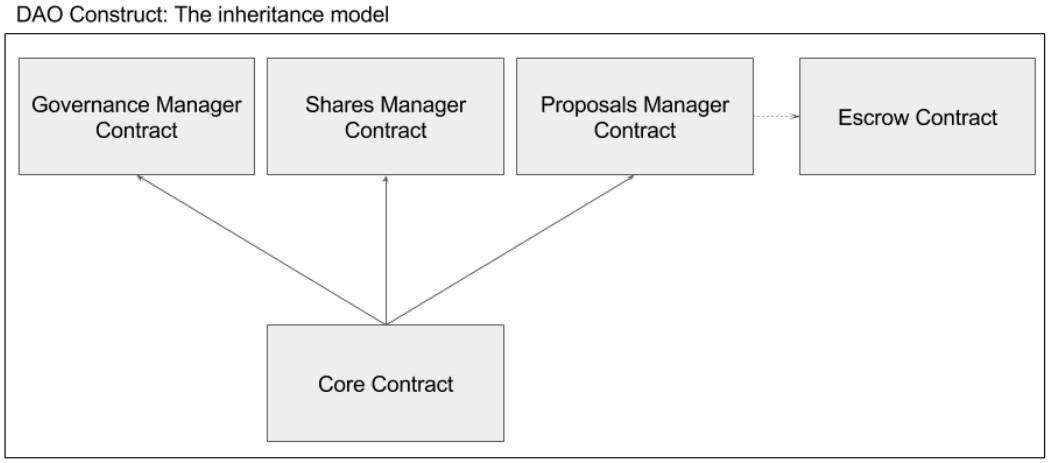
In constructing the DAO contracts, we have identified two main ways of organising their structure, with one based on inheritance mechanisms and the other relying on composition with other contracts. Diagrams representing these two structures are shown in Fig. 4.8.

As reflected in the figure, both of these contract designs need to cater to a certain amount of functionality which, perhaps unsurprisingly, touches on all the different parts of the workflow presented in the previous section. In both constructs we have included units in the form of contracts that are clustering together the functionality of the different parts of the workflow:

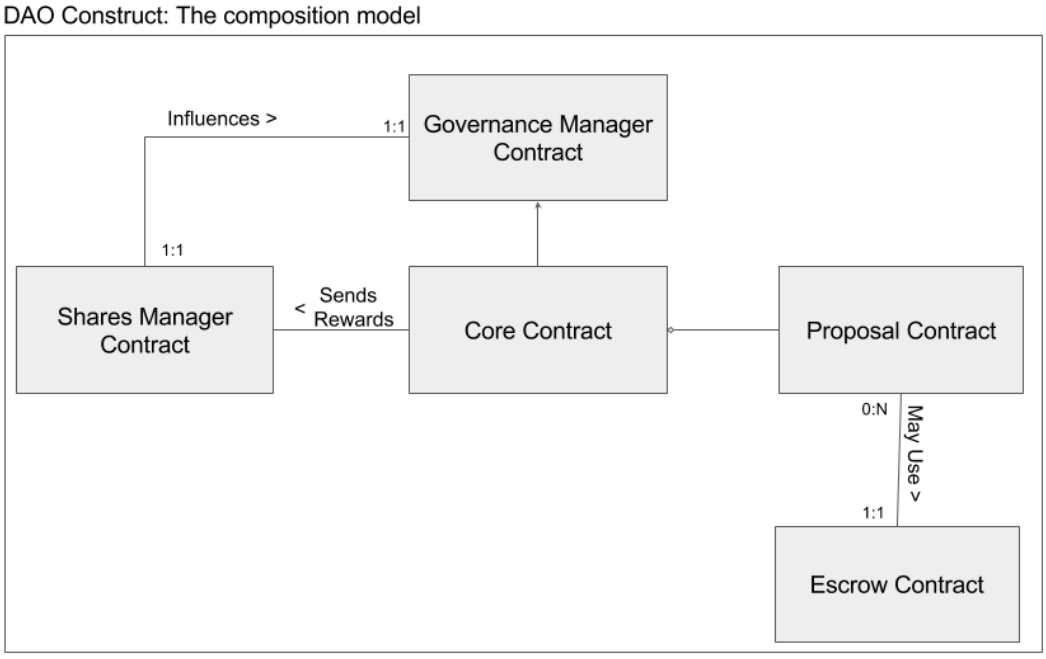
- The Governance contract embodies the different governance schemes (cf. Section 4.2.2).
- The Shares Managing contract represents the functionality related with investments and shareholding.
- The Proposal and Escrow contracts organise the employment workflows (cf. Section 4.2.4).
- The core contract then brings all the functionality together and adds operations management.

In the inheritance construct (cf. Fig. 4.8a), the "specialised" contracts are made to be parents of the core contract. The core then inherits all of their functions and has the task of organising them by supplying extra operational methods. The specialised contracts themselves do not have a direct connection, and as such any functionality involving cooperation of the different units will need to be part of the core. This has the consequence that units are strongly tied with the core contract and the type of DAO they represent.

In using the inheritance structure simplicity gains are non-negligible. All functionality is represented by a single Ethreum Address for the DAO which significantly simplifies technical workflows. In fund redistribution for example, having only one Address allows for a much clearer set-up, with funds either flowing in or flowing out of this address. If entirely separate contracts were involved, which is the case



(a) Inheritance-Based DAO contract structure.



(b) Composition-Based DAO contract structure.

Figure 4.8: Ways of structuring DAO contracts.

with the structure in Fig. 4.8b, then a multitude of Blockchain-related questions arise such as (1) which contract will be responsible for the fund redistribution, (2) how are the DAO funds allocated in the first place, (3) at which point are messages sent between the contracts and (4) how are data conserved in the inter-contract message communications.

Even though the inheritance structure is then more in line with our simplicity principle, the composition structure does appear to be the more scalable and maintainable option when the amount of functionality the DAOs need to cater for increases. The primary reason for this is that in the composition structure units are much more independent and are allowed to communicate between each other, making the structure more loose. As a consequence, each of the units in the composition structure can be made to interact in different ways with the other units by providing a sort of API, while with the inheritance structure such a feature is much harder to achieve due to the lack of communication between the units.

In order to illustrate why such a characteristic might contribute to the scalability of the solution, it is possible to consider the scenario of a grown ecosystem with many nuanced functionalities in the DAOs being offered to the users. In such a scenario, the inheritance based model is likely to require each nuance to be represented by an entirely new set of DAO contracts. With the composition model on the other hand, the nuances could be represented by changes in the unit in which they apply. DAOs would then be put together by assembling the different units that create the full functionality required, overall resulting in a much more maintainable and scalable system.

4.4 A Mixed Architecture

As we mentioned in Section 4.1.2, our system architecture is based on a collaboration between the block-chain and our front and back-end functionalities. We also choose to make use of a server database in order to enhance and enable some of our features. In this section we discuss the details of our architecture and the synergies it creates.

4.4.1 Centralised Components

The need for a Back-end Database

The first time when the need for a centralised server became apparent was while considering the browsing and opportunity discovery functionalities. What we needed was a way to search the Ethereum Blockchain on demand, querying for the contracts that are part of our ecosystem, based on attributes and other criteria on those contracts. We have not found an elegant and efficient way of doing so using only

functionality provided by Ethrereum. As such, keeping the idea of a client only solution depicted in Fig. 4.2a meant denying one of the fundamental aspects of our product and a regression to a mentality where creating DAO contracts is the end-goal by itself.

The solution of creating a Back-end Database then became quite self-evident. Upon deciding to go down this route, a dilemma appears for how much of the contract Data should get mirrored in the database. One possibility is to keep only the absolutely minimum information necessary for enabling the otherwise non attainable functionalities. Another possibility is denormalise most of the contract data by mirroring them on the central back-end. At this stage of development, both solutions seem viable with different advantages to each. The former design has the advantage that the system would potentially be more scalable as there would be a bigger split over the computation done on the client and on the server. The later on the other hand has the advantage that a consistent interface would be kept across all users as well as having the elegance and simplicity of all data being treated in the same way.

The User Accounts dilemma

Another important point of dilemma in developing or our solution was whether or not to make use of user accounts. The purely "decentralised" way of dealing with the issue would be that each user makes use of his Ethereum node's accounts and the web3 javascript object (cf. Section 2.3.6) in order to interact with the Blockchain.

In our system however, an issue arises primarily because of the rating system. In order for the system to work, we need metadata such as the ratings and comments to be associated with each account. In order to keep the solution fully decentralised, this meant that these metadata needed to be kept in the form of contracts which would be updated by the new comments and feedback during the employment contract life-cycle as seen in Fig. 4.5. This solution then creates very complex work flows where each user that interacts with our system would need to have an associated user contract with his metadata. The added complexity in the system not only creates opportunities for loopholes and hence system abuses to take place, but also goes against the simplicity principle we expressed in Section 4.1.4 .

Such an approach is consequently not entirely desirable in our solution, and given that we are making use of a central system regardless, we have decided to look for a more centralised approach. The first option that then comes in mind is to entirely host the users' Ethereum accounts on our servers. Whenever a user creates an account, this would create an Ethereum account in our server node and associate it with this particular user. This solution would have arguably been the easiest to implement. Nevertheless, unlike in the other instances of centralisation in our system, we feel that adopting such an architecture would correspond to denying the fundamental benefits of Blockchain technology as our central servers would effectively have full control over the users' accounts. Consequently, this architecture was also set aside.

The solution finally adopted was somewhat of a hybrid of the two approaches. Users still keep their Ethereum accounts in their local nodes, but need to register in our platform by providing their Ethereum account's public key and some metadata such as a username. The platform then uses these accounts in order to keep track of all the reputation system metadata, while Ethereum transactions are still done using the users local node.

4.4.2 Technical Considerations

The main challenge that arose from adopting the approach of a mixed architecture has been finding a reliable way of keeping the denormalised Blockchain data synchronised with their server counterparts. For instance, if someone makes a transaction from a Decentralised Startups DAO without going through our system, the question of how to still capture and reflect it through our portal arises.

One possible architecture that was then explored consisted in forcing all contracts that relate to our ecosystem to be interacted with through our platform. Ways of achieving this could have been through smart contracts, by having an extra step for all transactions involving DAOs where a cryptographic hash set by our servers would need to be verified before proceeding. Similarly to user accounts however, this structure created (1) a form of centralisation that extends beyond information display and (2) convoluted and hence potentially exploitable processes. This solution was therefore not pursued.

The architecture finally endorsed is depicted in Fig. 4.9. It includes a protocol with series of checks and safeguards, that have both pre-emptive and reactive measures in order to ensure a maximum level consistency of the data stored in our servers.

As seen in the figure, data are initially submitted by the users through the UI. Transaction requests are then processed on the client before being sent to the Blockchain. If the corresponding information is needed on our database, the transaction data are also sent to the server. On the back-end it is then confirmed that the transactions have been successfully processed on the Blockchain and then added to the database. Through this process the integrity of the data is maintained, and what users of the system view at the end is a combination of the database information and direct Ethereum data.

This protocol involves 5 levels of integrity checks:

Levels 1 and 2.

As it appears in Fig. 4.9, the first two levels are carried out on the client side. They consist of pre-emptive measures that should prevent most of the issues relating to the transactions sent through our portal.

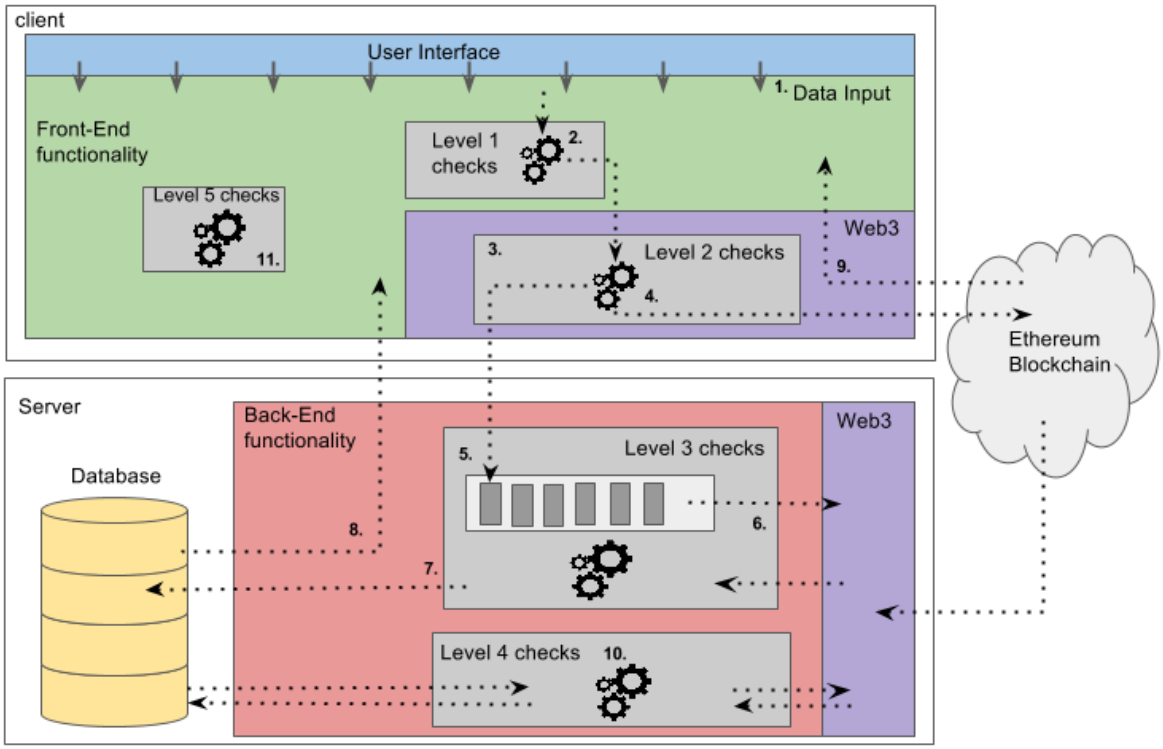


Figure 4.9: Main System Architecture. *(1) Data input through the User Interface (2) Sanity Check on the client (3) Transaction verification on the local node (4) Transaction sent to the Blockchain and to the server (5) Transactions enter a queue on the server (6) Confirmation of transactions on the server (7) Confirmed information written on the Database (8) Database information displayed to the front-end (9) Direct Ethereum data displayed in the front-end (10) Routine maintenance checks for Database consistency (11) User induced consistency checks.*

At level 1, checks aim to verify the sanity of the transaction that is about to get processed. More precisely, they independently verify that the transaction will be allowed to occur on the Blockchain. In the simple example of a transaction for a DAO sending 20 Ether to a user, it is verified that the DAO currently possesses more than 20 ether before proceeding.

Level 2 checks are made by the local node and make sure that the transaction is sound and well formed. For instance, if a transaction is made and the specified sender of the transaction does not exist, level two checks will be able to catch the issue and prevent further processing [45].

No information is sent on the server(or the Blockchain) if these tests don't pass which is the reason they enforce consistency. Nevertheless, it is not impossible to imagine situations where the tests pass and issues arise at a later stage, or where the tests are circumvented altogether. One of these situations is where a transaction that is executing on the Blockchain runs out of gas during execution and is forced to roll back all changes it initiated. Another problematic scenario is when a person engages with the DAOs code directly, without going through our portal. In both of these cases, our server needs a way to be notified of events that occurred on the Ethereum Blockchain, without occurring within our system.

Level 3.

The level 3 protocol is in place in order to cater for such cases. It largely based on Paul S.'s method for confirming if transactions run out of gas, as described in a Stack Exchange post [102]. This is illustrated in the pseudocode in Protocol 1 and discussed in more detail below.

Protocol 1: LEVEL3-CHECK

```
1:  Initialize a set S of transaction structures
2:  while S is not empty do
3:    for each structure in S do
4:      if transaction receipt is null then
5:        Sleep for a given period δt
6:      else if transaction receipt and appropriate event log exist then
7:        Reflect transaction in central Database
8:        Remove transaction from S
9:        Notify the client of the outcome
10:     else
11:       Remove transaction from S
12:       Notify the client of the outcome
13:     end if
14:   end for
15: end while
```

Once a transaction is sent to Ethereum, the information it carries and its transaction hash are also sent to the server, where they enter a set of transaction structures. The server then continuously probes the Blockchain through the `getTransactionReceipt` function of the web3 API [53], watching for these transactions. If the receipt is received then the transaction has been mined, and if it contains the correct events associated with the transaction in its logs then the execution succeeded (cf. Section 2.3.6). At this stage the the transaction is either considered as confirmed and the associated information is synchronised with the central database, or it is not confirmed and no action is taken. This process then also implies that each relevant contract function will need to have a corresponding event in the contract implementation.

Levels 4 and 5.

Levels 4 and 5 add a last layer of security in keeping a clean and consistent database state.

The level 4 checks essentially consist of periodic database maintenance, which verifies all the existing information is consistent with the ethereum Blockchain data. The checks can either review the full database state, or the state of a particular DAO. In either case, the maintenance steps are as follows:

1. All transactions through the portal that influence the information being maintained need to stop.
2. All pending transactions in set S (cf. Protocol 1) are verified.
3. All information of interest is verified through the Blockchain.
4. Normal transaction activity is re-enabled.

Finally, level 5 consists of user-triggered checks. The idea behind this mechanism is that users can through their UI verify the sanity of the information presented to them. When carrying a level 5 check, the information of the current database state, the current Ethereum state and the current state of the set S (cf. Protocol 1) should be displayed to the user.

An added synergy between these two last levels can also be created, where users that spot inconsistencies through the level 5 process can trigger or push for a level 4 check on the DAO of concern.

Chapter 5

System Prototype

5.1 Overview

In order to provide a practical proof-of-concept of our solution, we have created a prototype implementation. This proof of concept needs to embody the key principles of Decentralised Startups while effectively limiting the scope of the features to be implemented.

In this proof-of concept, we have decided to focus our attention at a particular use case of Decentralised Startups. Specifically, users will be able to create DAOs, and contract other users (or groups joining as a single user) in order to accomplish concrete, tangible and self-contained tasks which result in some content creation. Examples of such tasks would be the creation of a computer game that results in an executable file, the writing of an ebook or the recording of a podcast.

In order to illustrate this use case, we can consider the example of a firm distributing specialised educational content. The firm would create a DAO on our system and create proposals for the creation of PDF ebooks as well as pod-casts on the various subjects of interest. It would then look to contract experts in the corresponding fields that would take on the task of creating that content. Once the different products are ready, they can be sold by the DAO through the associated distribution mechanisms.

With this goal in mind we have created a set of precise specifications for this prototype, which are discussed in the following section.

5.2 Specifications

5.2.1 User Accounts

User accounts are useful to both contractors looking for opportunities and owners of DAOs. Their primary purpose is to enable and facilitate the rating system of the solution as described in Section 4.4.1. User accounts come with an associated public user page that interested parties can visit to determine the legitimacy of the

user through his past activity and description.Overall the user accounts provide the following information:

- A list of the past and present contracted jobs the user has been engaged in. This list provides direct insight on the user's past and present history. Employers wishing to contract this user can then see through this list the amount of work he currently has and the types of projects he has carried to successful completion.
- A list of the DAOs the user owns. This can again be used in order to measure the activity of the user and how successful are the ventures he engages in.
- The overall rating and different feedbacks received by the past employers of the user (cf. Section 4.2.5).
- A public self description of the user. In this section the user can provide a description of himself and his skills, as well as any relevant links and contact details that he wishes to share with the other users. This is useful for the users to be able to sell themselves, and for other interested parties to get more material for forming an opinion about the user. Example use cases of this feature are investors looking into owners of DAOs or employers investigating employment offer applicants.

5.2.2 DAO Creation

The DAO creation is one on the most important features of our system. DAOs are the centrepiece of the solution, and all significant functionality of the solution originates from them. Creating a DAO corresponds to deploying the DAO contract on the Blockchain as well as recording this DAO to our database (cf. Section 4.4). Once the DAO contract is deployed, it enables all of the commercial interactions defined in its functions and becomes an integrated actor of the economic environment created by our platform.

We will only use the private type of DAO (cf. Section 4.2.2) here, which relies in an owner having all of the executive powers. The owner of the DAO needs to be in possession of a user account. This is necessary as (1) it provides a means for authenticating the feedback provider of the DAO contractors and (2)enables investors to investigate the activities of DAO owners.

We have decided to make DAO creation a very seeming-less process, with only the crucial information required for deploying the DAO contract being requested. As such, DAO creation involves the following:

- Provide an owner, which the creator can set as himself. We have decided to restrict the DAO creation to logged-in users, as practically only them would benefit from doing so.
- Provide a title which can be seen as the brand name of the corporation.

- Provide an official declaration for the DAO. This declaration is very important and consists of a description of the intended DAO activities that will be stored in the Blockchain (cf. Fig. 5.3). The aim of this declaration is to define the scope of activities of the DAO and restrict owners from abusing stakeholders. Given this description is stored on the Blockchain, it is permanent and immutable. As such, it can safely be used, along with the Blockchain recorded transaction history of the DAO, for potential investors to be warned of rogue behaviour of the owner, or possibly in some cases for existing shareholders to take legal action against him.

5.2.3 Opportunity Discovery

As seen in Section 4.2.5 , opportunity discovery is one of the pillars Decentralised Startups uses for truly enabling the digital economy and providing value to freelancers, SME's and investors altogether. Our prototype directly implements browsing for DAOs and Employment offers. Product browsing and discovering is done indirectly through the DAOs, where a user requiring a certain productcan find the DAOs in the corresponding industry and then subsequently look at their products. The option of directly connecting to a DAO is also offered, so that a user knowing the address of the DAO he is interested in does not have to go through the trouble of searching for it.

As such, the browsing features are the following:

- DAOs can be browsed through keywords or by specifying the different activity flags of interest. Activity flags are : Recruiting, Selling Products, Looking for Investments and Building new products. The search results contain a link to the main monitoring page of the corresponding DAOs and some of their essential information . Particularly, the title, the owner, the available funds, the activity flags, the address and the current rewards on revenue for investing are displayed. The purpose of this information is to help investors quickly making an opinion for which DAOs are worth further investigation.

- Employment offers are browsed through keywords and remuneration. The option of also displaying already assigned tasks is offered, which can be useful in case there is a lay-off in one of these positions. Again, essential information about the positions and a link to the corresponding pages are offered in the search results.

5.2.4 Investing

As seen in Section 4.2.2 , investing is available to users under the private type and as such it is also possible in our prototype. Investing rules are the following:

- DAO owners have to enable investing for the DAO before users can invest in it.

- A user becomes a shareholder of the DAO by sending funds to it through the corresponding form (which activates the share creation contract function -cf. Section 5.5-).
- Users can transfer any amount of their shares to other users.
- Shareholders can verify the amount of shares they own in the DAO.
- Shareholders are entitled to a certain percent on the revenue of the DAO. This amount is set by the owner, and fund distribution happens automatically through the DAO contract (cf. Section 5.5).

5.2.5 Product selling

DAOs make revenue by selling the contents produced by their contractors. The features of the selling process are the following:

- Owners can set the selling activity flag of their DAO in order to allow the selling of its products. This activity flag also serves for browsing purposes.
- In order to protect the contractors, a product is only available for sale once the corresponding proposal has been finalised (cf. Section 5.5).
- Owners can change the information displayed for each of the products in offer. Available information fields are: (1) The name of the product, (2) a brief description of the product, (3) the price of the product, (4) a link towards an external page with more information about the product.

5.2.6 DAO monitoring

Each of the DAOs in the system has their dedicated control room page through where all the functionalities relating to those DAOs can be accessed. This monitoring page is therefore the portal's main hub and consists of the following:

- A display of the main information regarding the DAO as well as its financial and historical data. This includes the history of all the transactions made since the creation of the DAO and that originate from it. Investors and potential contractors alike can use this information in order to establish the legitimacy of the DAO. Investors in particular can see the current returns on revenue from the DAO, and combine it with the Official statement and the change history of this figure in order to establish the risk on potentially low returns from this investment.
- A list of the Employment offers made by the DAO. Each offer further has a dedicated page where the owner, applicants and eventual contractor will be able to interact together through the different employment rules and workflows (cf. Section 5.4.2). Each offer page then has all the commands necessary to this end, including facilities for file uploading of the content produced.

- A panel with the controls available to the owner, including mechanisms for adding new employment offers and changing the activity flags of the DAO. Most of the owner actions are available through this panel, except the specific actions concerning each employment contract, which are part of the employment's dedicated page.

- A panel with the controls available to the investors (cf. Section 5.2.4).

5.3 Technologies used

A significant amount of effort has been put into finding and choosing the correct tool-kit (Integrated Development Environments (IDEs), frameworks, languages) for our implementation. There are two types of tools that needed to be used, namely Ethereum development tools and web development tools. Specifically, an important part of our efforts have been directed towards creating synergies between the different parts of our tool-kit as well as finding efficient development workflows.

5.3.1 Web Development tool-kit

The first and most critical step in choosing our web development tool-kit consisted in finding a web development framework for our application. The three main candidates were Django [103] which uses Python [104] as its main language, Ruby on Rails(RoR) [105], which is based on Ruby [106] and Meteor [107], which uses Javascript [108] . Django was rapidly disregarded due to a previous experience with the framework which was not entirely satisfying. Regarding the remaining two frameworks, RoR had the advantage of being a strong framework that is well established [109]. On the other hand, Meteor is well suited for real-time responsive applications, which is an interesting as it adds demonstrating power to our application [110]. Additionally, Meteor is recommended in Ethereum's Github wiki page and also features some packages for Ethereum Dapp development [45].

As a result of the above considerations, Meteor was chosen. Meteor is a full stack framework based on node.js, that uses MongoDB [111, 107]as its back-end database. One of its most advertised features is that it has built-in reactivity, meaning that changes on its database objects are immediately reflected on the front-end without the need for refreshing the page. Another very important characteristic of Meteor is that its native language is Javascript, both for the front-end and back-end functionalities [107, 112]. This creates a particularly helpful synergies with Ethereum, given that an Ethereum node is accessed through a javascript object, web3 (cf. Section 2.3.6) [53]. An example of such a synergy is that meteor makes it very easy to set up communication of data acquired through the web3 object to any other part of the application.

Other helpful features of the meteor framework is that it has a very flexible file structure and it benefits from a large and active community. This is reflected through Meteor's package portal, atmosphere, which features more than $11,000$ packages [113]. Some of the main packages we used in our project are the following:

- **ethereum:web3.** This is the main Ethereum package that defines the web3 object allowing to connect to an Ethereum node [114, 53].
- **twbs:bootstrap.** This package allows us to use twitter Bootstrap [115], which we used as the main front-end library in our implementation [116].
- **iron:router.** Iron router is one of the most popular routing packages for meteor [117, 118].
- **Session.** Session is a very helpful package allowing for data transfer between different parts of a meteor application [117, 112].

The second essential tool that we needed to choose was an IDE for developing our product. The question then arose on whether to choose a Javascript/web development dedicated tool [119] or one of the Ethereum optimised IDEs such as Mix [45] or EtherCamp's Ethereum Studios [120]. We have finally settled on Webstorm [121], a javascript IDE that has native support for meteor [122]. This choice is justified by the fact that we have separated our contract and web development tasks as seen in more detail in Section 5.3.3.

5.3.2 Ethereum Development tool-kit

In choosing our Ethereum tools, two immediate choices where Geth [123] as our client and Solidity as the language to use, given that those are the most popular and well supported at the time of writing [17]. Following those two fundamental choices, we next decided to use Truffle [124, 125], as our Dapp development framework. The main benefit from using Truffle was the access to its integrated testing workflows and tool-kit. Specifically, Truffle uses Mocha [126], Chai, [127] and pudding [128] for making unit tests, which greatly facilitates contract testing.

We have next used testrpc as our development ethereum node. Testrpc creates an artificial environment that simulates the functioning of the Ethereum Blockchain without truly connecting to it. As such there is no need for mining, Ether or polluting the real Blockchain. All transactions are sent from one of the 10 unlocked accounts provided by each instance of testrpc and are executed instantly. This is ideal for development or demonstration purposes and hence testrpc has truly become a core component of our development tools [129].

The final parts of the Ethereum tool-kit we used are the Atom text editor [130], which has useful packages for the Solidity language, and the on-line Solidity compiler [98] which allows us to easily get the compiled versions of our contracts as well as their ABI (cf. Section 4.1.3).

5.3.3 Development Organisation

As might have become apparent from the previous sections, we have chosen to entirely separate our contract development and web interface development processes rather than attempt to integrate them together. One of the reasons for this decision was that Truffle can also be used as the web framework supporting the Dapp, but we decided to use Meteor [125]. As such, Truffle was solely used in order to streamline our contract development and testing processes.

Overall, our development workflow is as follows:

- Use the testrpc to simulate an Ethereum node.
- Develop and test the DAO related contracts using Truffle through Atom.
- Develop the web platform using Meteor and Webstorm.
- Get the compiled versions of the contracts using the on-line solidity compiler.
- Transfer the compiled contracts over for use in the web application.

5.4 Portal Implementation

5.4.1 The Architecture

Our implementation follows the principles of the Decentralised Startups architecture, with most of the contract data denormalised to our mongoDB database (cf. Section 4.4). We also enforce consistency using level 1 and level 2 checks as discussed in Section 4.4.2.

The organisation of the different components in our system is shown in the entity-relationship (ER) diagram below (Fig. 5.1). Specifically, this ER representation uses the *Look-Here* cardinality and allows for *Nested Relationships* and *Multivalued Attributes*.

As seen from the ER diagram in Fig. 5.1, A **DAO** is identified by an *Address*, and includes (1) a *Title* which represents of the organisation, (2) a *Description* which is the official declaration, (3) a *balance* which are the current funds of the DAO, (4) a *Revenue Distribution Rate* entitled to shareholders, (5) *Activity Indicators* which is any combination of {*Recruiting, Selling Products, Building Products, Looking for Investment*} and (6) A set of *Shares* with their associated Shareholders.

A **User** is identified by the address of his externally owned Ethereum account. He possesses (1) a *Username* as a means for a human readable identification of the user, (2) a self-supplied *Description* ads contact details and Curriculum Vitae(CV), (3) an overall *Rating* from all the employment tasks he accomplished. Users can be DAO **Owners** and/or **Contestants** in employment positions. **Contractors** are a further

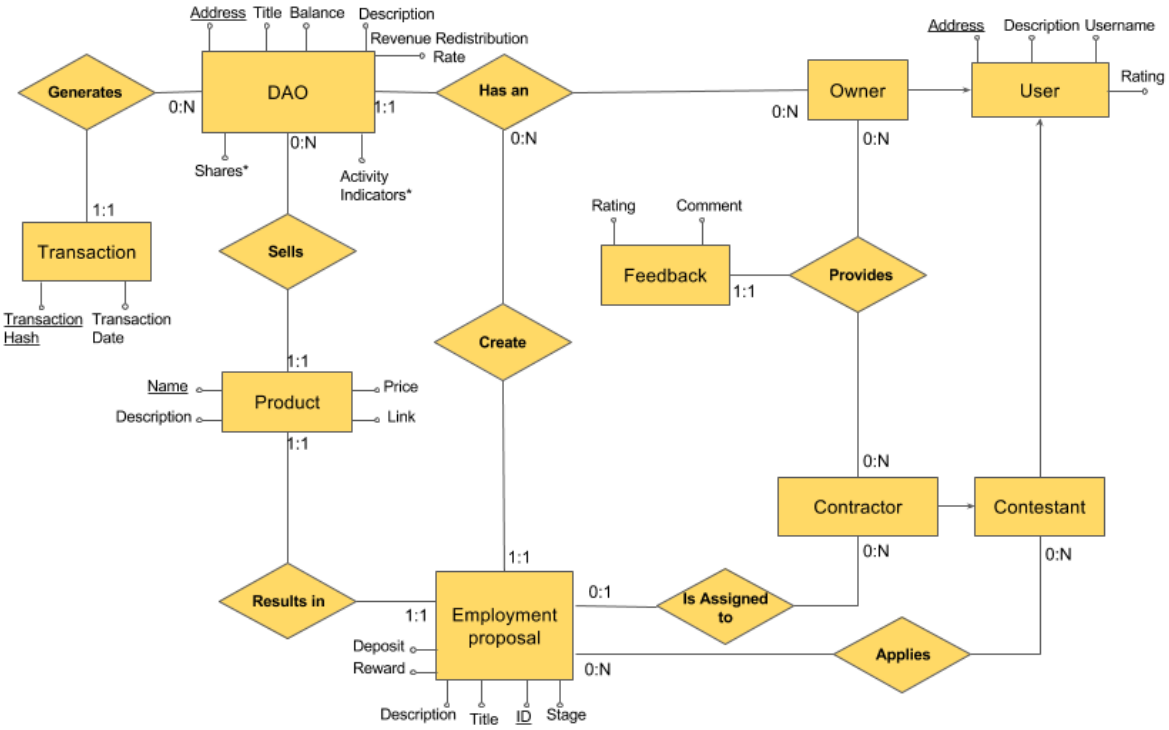


Figure 5.1: Entity-Relationship Diagram for our System.

specialisation made of the **Contestants** that have been selected to carry the corresponding post.

Each DAO has a unique **Owner**, and under his instructions creates any number of **Employment Proposals**. Those are identified by their unique *ID* within the DAO and include (1) a *Title*, which summarises the position, (2) a *Description* which defines the task at hand, and (3) *Deposit, Reward* figures indicating the amounts awarded for the task.

Any number of **Contestants** can apply to any number of proposals and only one contestant is chosen as a contractor for the post. The associated owner can then provide **Feedback** - that includes a *Rating* and a *Comment* - to the contractor.

Employment tasks result in **Products** being produced and having the possibility of being sold through the DAO. Each product has (1) a *Name* through which it is identified, (2) a *Description* that contains a summary of what the product is, (3) a *Price* at which it is being sold and (4) a *Link* with any URL, usually towards a page with a fully featured description of the product.

Finally, all interactions with the DAO generate **Transactions** of which we record the *Transaction Hash* and the *Transaction Date* in order to the users to be able to easily investigate the behaviour and history of the DAO through Ethereum Blockchain explorers such as EtherCamp [120].

5.4.2 Workflows

The prototype's workflow follows the organisation shown in the previous chapter in Fig. 4.3. In this section we take a closer look at the different parts of this general organisation, specifically looking at how they are embodied in our prototype. The implemented workflows are then reflected through the three flow charts of Fig. 5.2.

Investing Workflow

Fig. 5.2b shows the an overview of the different steps involved in the investment process.

No investments can be made to the DAO unless the owner initially indicates that the DAO is looking for funding. In doing so the owner switches the investment flag -or activity indicator- of the DAO contract to ON which (1) allows for shares to be created by the DAO in exchange for funds (cf. Section 5.5) and (2) places this DAO in the investment opportunity searches.

The DAO can then be found by potential investors through the browsing facilities. Moreover at that stage any user can send Ether to the DAO and become a shareholder. Once a shareholder, the user is (1) entitled to a certain amount of revenue as is defined in the corresponding Ethereum contracts (cf. Section 5.5) and (2) can use the investor section of the web interface in order to transfer some of his shares or to verify how many he owns.

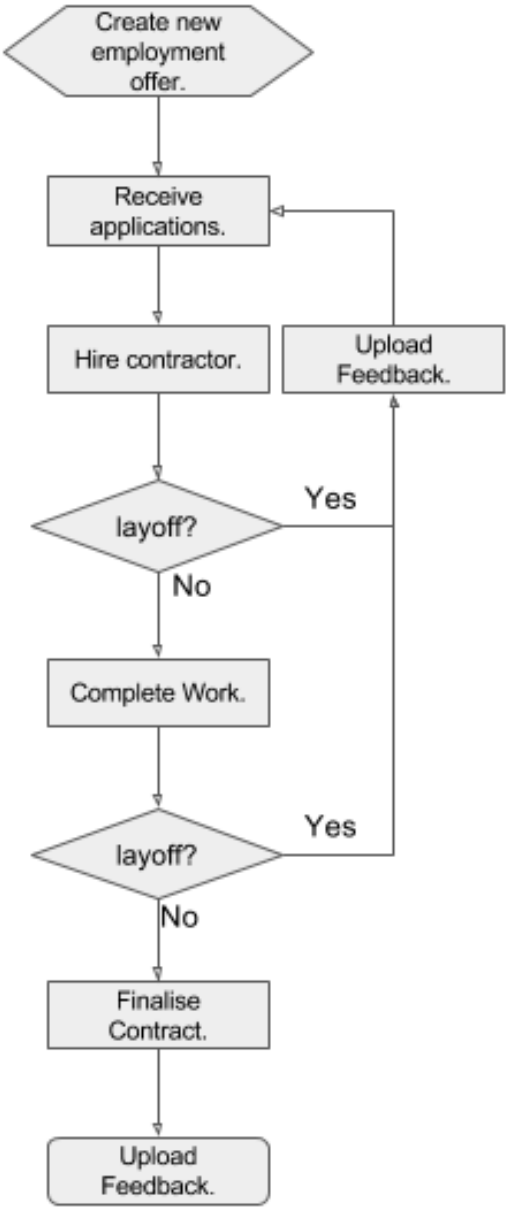
Employment Workflow

The employment workflow shown here follows mostly the same structure as the one seen in Fig. 4.5 with nevertheless a more linear progression.

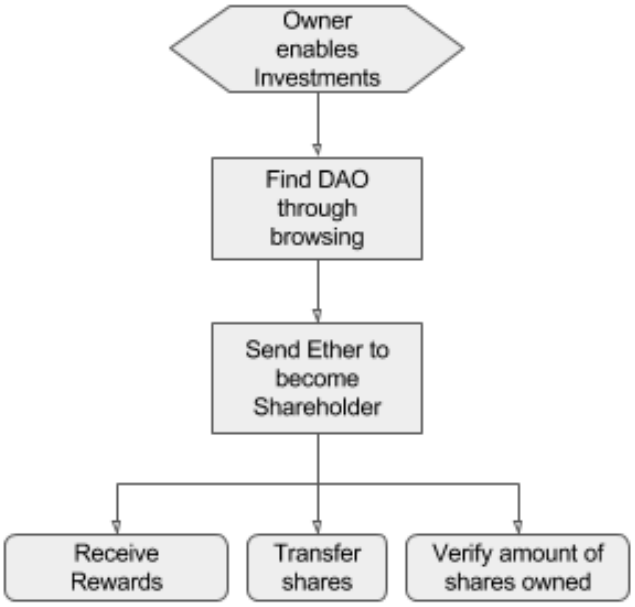
In order for the process to start, the owner of the DAO needs to enable recruiting, at which stage employment proposals can be created (cf. Section 5.5). He then uses the portal facilities in order to create these proposals, which then become searchable through the browsing facilities. Potential contractors can then find these opportunities and apply for the positions.

If the owner is satisfied with one applicant, he can hire him for the task. This involves the automatic transfer of the deposit specified in the employment offer to the contractor. Other candidates can still deposit applications at that stage in the eventuality that the contractor is let-go.

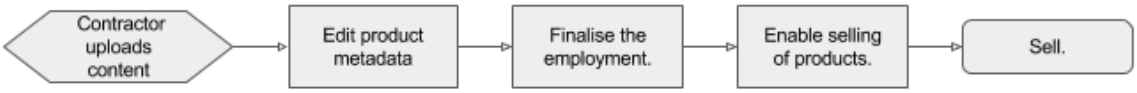
Once the contractor completes his work he can upload it through the portal. The owner can then finalise the employment, which (1) automatically sends the rewards associated with the proposal to the contractor (2) freezes any other interaction with that particular post and (3) enables the selling of the product.



(a) Employment steps flow-chart.



(b) Investing steps flow-chart.



(c) Product distribution steps flow-chart.

Figure 5.2: System overview.

At any stage until the finalisation of the post, the owner can let go of the contractor, in which case he looses his deposit and has to choose a new contractor from the applicant list. When either conducting a lay-off or finalising the proposal, the owner also uploads feedback (a comment and a rating value) for the contractor to the system.

From the above it becomes apparent that honest behaviour of all parties involved is obtained through a combination of the automated rules of the smart contract and the implemented workflows. On one hand contractor's behaviour is regulated via the feedback mechanism, as discussed in Section 4.2.5. On the other hand, malicious behaviour form the owners is prevented via the deposit and finalisation mechanisms: The products cannot be sold if the owner has not finalised the post (hence rewarded the contractor), and abusive lay-off of contractors is discouraged through the associated costs.

The Product distribution workflow

The product distribution workflow partly becomes apparent through its employment counterpart discussed above. The process starts through the upload of the content created by the contractor. The owner can then edit the metadata (such as the price or the name) of the product associated with that content. Finally, the owner needs to finalise the corresponding proposal and enable selling for the DAO.

The products for which the above actions are successfully completed can be sold through the portal. When a user buys one of these products, part of the revenue generated is automatically redistributed to the shareholders. The amount that redistributed is managed through the smart contracts, and is set by the owner through the redistribution rate (cf. Section 5.5).

5.5 Contract Implementation

5.5.1 Overview

Our prototype contracts have been implemented using Solidity, and follow the general design principles as well as inheritance structure specified in Section 4.3. In developing those contracts, we have started with the Ethereum Foundation's Shareholder Association model [57] along with elements of "The DAO" [27] and have constructed our model on top, via adapting the contracts to our design principles and building in our system's workflows. Overall, our contract implementation design is shown in the UML (Universal Modelling Language) class diagram in Fig. 5.3, and our full contract implementation appears in Appendix B.

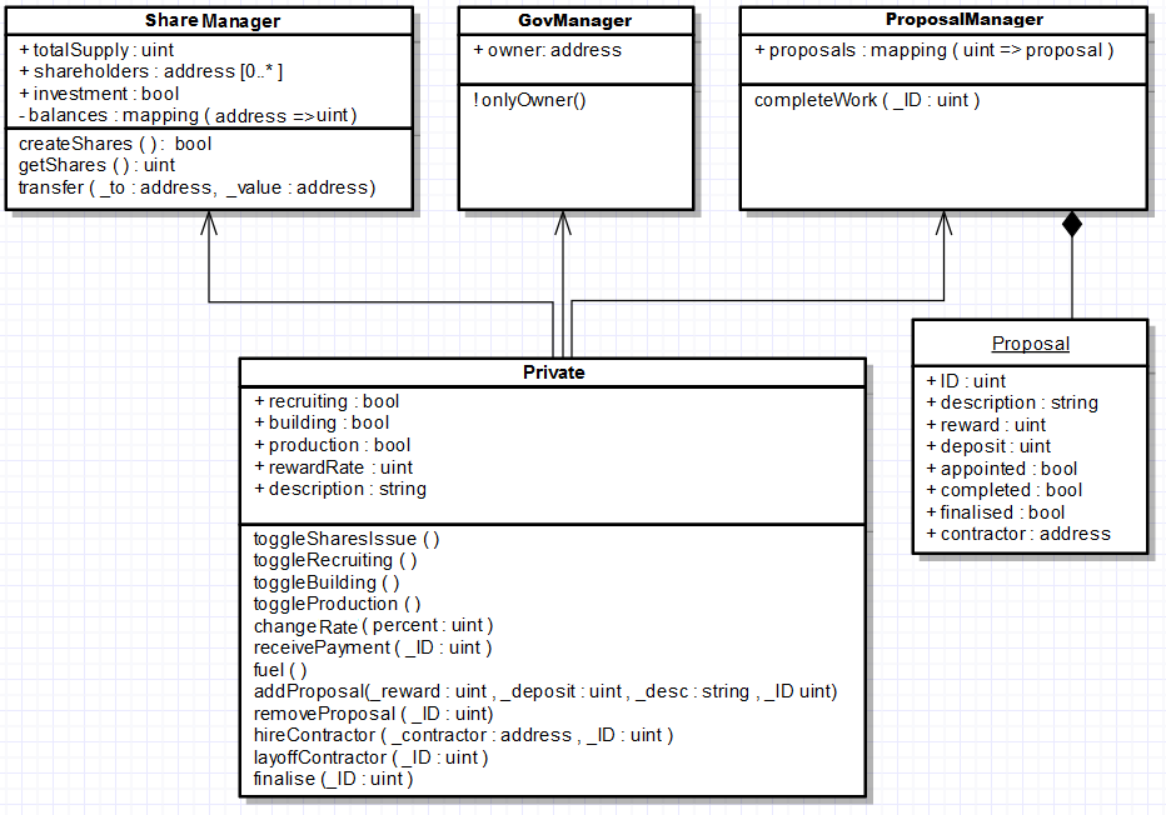


Figure 5.3: UML class diagram of our contract implementation. *Methods Preceded by "!" indicate function modifiers*.

Consistently with the inheritance design, our DAO construct consists of several contracts representing the different units of operation of the DAO. As the names indicate, **ShareManager** organises shareholding and investments in the DAO, **GovManager** defines the governance rules and **ProposalManager** relates to employment management.

Our core contract is **Private**, and organises the operations of the DAO and the functionalities involving more than one unit of operation. For instance, most of the functions in the Private contract have the `onlyOwner` modifier, which enforces to the other units the core governance rules of this DAO type, namely that operations can only be carried by the user behind the `owner` address.

5.5.2 DAO operations

The DAO operations are controlled by a set of central flags that define which actions can be carried out by the DAO. These correspond to the Activity Indicators mentioned in the previous sections and are as follows:

1. The `recruiting` flag indicates whether the DAO is in a recruiting phase or not and is set on and off by the `toggleRecruiting` function. Its functionality consists of allowing and disallowing (1) new proposal (employment offer) addition to the DAO and (2) hiring contractors. This ensures that if a proposal

is looking for a contractor, its corresponding DAO will appear in the browsing facilities of our portal and hence promotes equal opportunity for all the participants.

2. The `building` and `production` flags (and the corresponding `toggleBuilding` and `toggleProduction` functions) relate to the content that is being produced by the DAO, indicating that it builds new products or that it already possesses production content on the market respectively.

3. The `investment` flag in the **sharesManager** contract indicates whether the DAO is looking for investment and enables share issuance for the DAO. It is enabled and disabled by the `toggleSharesIssue` function on the Private contract.

Most of the other operations of the DAO then relate to shareholding and revenue management or to Employment management, which we discuss in more detail below. Apart from these, the owner of the DAO also has the option to `fuel` his organisation, which serves as a simple funds transfer mechanism.

5.5.3 Shareholding and Revenue Management

As suggested above, the primary consequence of turning the `investment` flag on is gaining the ability of using the `createShares` function which issues new shares. Any person with an Ethereum address can then become a shareholder of the DAO by making a transaction that calls this function and attaching a certain amount of Ether to it (this is done automatically through our portal). The shares creation mechanism is also relatively straight forward, with 1 share created per Ether sent. This has the interesting implication that as new shares are created, the value of existing shares is diluted, and hence the owner needs to take this point into consideration when deciding to enable shares creation. Once created, the shareholders can view how many shares they own using the `getShares` function, or can transfer their shares using `transfer` function.

All shareholders are also entitled to revenue redistribution from the DAO, at a rate set by the owner. This functionality is ensured by the `receivePayment` function and works as follows:

1. The owner sets the `rewardRate` (which is the revenue redistribution rate mentioned in the previous sections) using the `changeRate` function.

2. Every time an item is purchased from the DAO, the payment processing is ensured by the `receivePayment` function.

3. The amount of the purchase is then split between the DAO and the shareholders, at a rate determined by `rewardRate`, with each shareholder receiving an amount proportional to the quantity of shares he possesses. Precisely, if $rewardRate = X$, and the DAO receives an amount A of revenue, then a shareholder with Y shares will receive $X\% * \frac{Y}{TotalShares} * A$.

5.5.4 Employment Management

The central component of employment management in the DAOs are proposals (employment offers). A proposal takes the form of a structure in the **ProposalManager** contract. The `proposals` mapping then keeps track of all the proposals in the DAO, with the integer key of the mapping representing the unique ID of the proposal. A new proposal can be added to the DAO by the owner using the `addProposal` function, subject to the state of the `recruiting` flag. The owner can also remove a proposal using the `removeProposal` function, only however if the proposal is at its initial state (cf. Fig. 5.4).

The structure of the proposal then keeps track of all the necessary meta-data associated with the employment offer: `ID`, `description`, `reward`, `deposit`, `contractor`. It also keeps track of three vital flags for the management of the employment workflow: `appointed`, `completed`, `finalised`. The state of these flags then corresponds to the global state of the proposal itself which greatly influences the possible actions that can occur with regards to the proposal. This creates a state transition system that is illustrated in Fig. 5.4.

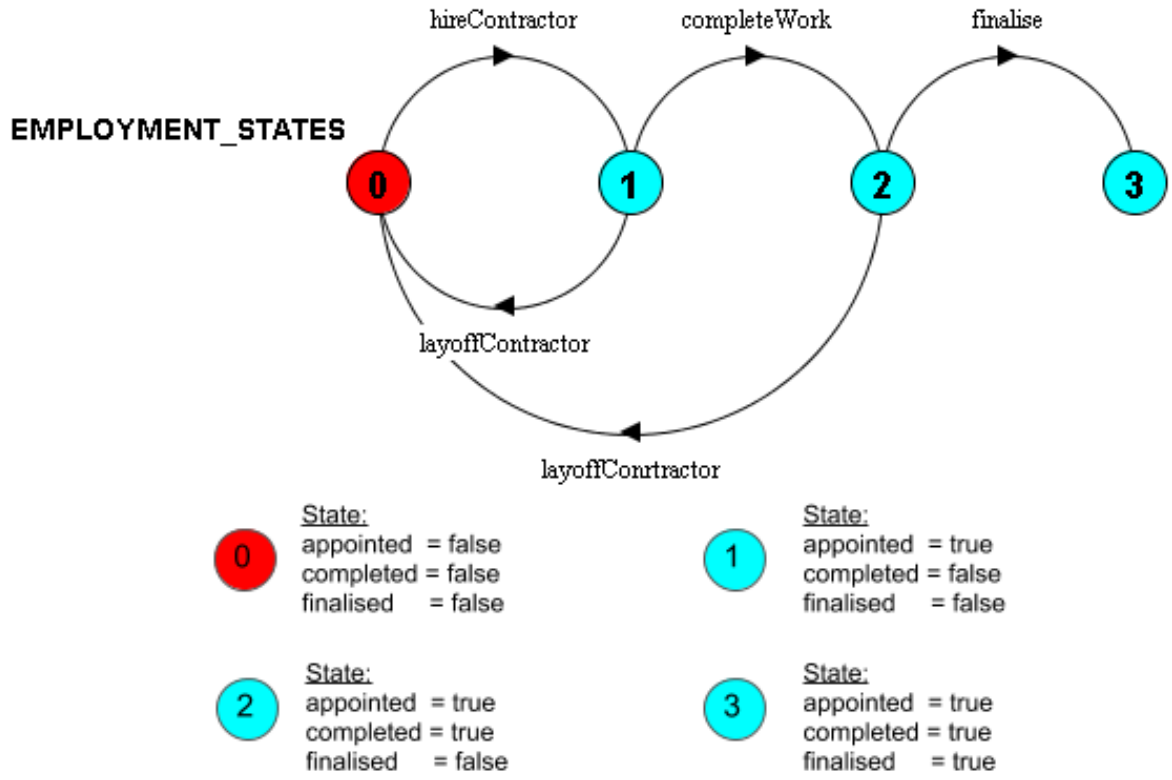


Figure 5.4: State Transition Diagram of the contract proposals using the Labelled Transition System Analyser(LTSA) tool [131].

In fact, as seen from the state transition diagram, each of the functions associated with the employment workflow corresponds to a transition function between states of the associated proposals. More precisely, the following transition functions are available:

1. `hireContractor` can only be used at state (0) and be fired off by the owner. It assigns a contractor to the proposal structure, marks the appointed flag as true and sends the amount recorded in `deposit` to the contractor.

2. `completeWork` can only used in state (1) and called by the contractor. Once executed, the function makes the transition between states (1) and (2) by setting the completed flag to true in the proposal structure.

3. `layoffContractor` can be used in both states (1) and (2) by the owner. It resets the state of the proposal to (0) and discharges the contractor.

4. `finalise` can only be used in state (2) by the owner. It takes the proposal to the terminal state by turning the `finalised` flag on, and sends the amount indicated by the *reward* variable to the contractor.

Finally, interesting parallel to draw is that the state transition diagram in Fig. 5.4 (and its transition functions) look quite similar to the flow chart in Fig. 5.2a. This is to be expected, as the state transition diagram corresponds the low-level contract implementation representation of the same high-level workflow depicted by the flow-chart.

5.5.5 Contract Security Evaluation

In the matter of contract security, given the dramatic events in "The DAO" phenomenon, we have been quite careful when constructing our contracts. We have, for instance, been mindful of negating the "recursive call bug" discussed in Section 3.3.3 by using the `send` function instead of the `.call` function and by updating any conditions in the correct order. Nevertheless, in our latest contract security review, we have discovered a possible attack vector in our `receivePayment` method. The vulnerability can cause a "Denial of Service" attack as described by Peter Vessenes in his blog post "Ethereum Griefing Wallets: Send w/Throw Is Dangerous" [132]. The attack then arises from the part of the function shown in Listing 5.1 [132].

As Vessenes points out, the vulnerability arises from using the `.send` function along with `throw` if send fails. The attack can be carried by putting a contract as one of the shareholder addresses, and making sure the fallback function of that contract is very expensive in gas, so that it is guaranteed to consume more than what `send` allows. If this is the case, then every time the `receivepayment` function is called, `send` will fail for this shareholder and the full execution will `throw`, thereby never sending funds to any shareholder. As the end result, the attack effectively succeeds in "denying the service" of payment for all the shareholders [132].

```
1   function receivePayment(uint _ID){
2         ...
3         for (uint i=0; i<nbShareholders; ++i){
4
5           address holder = shareholders[i];
6           uint shares = balances[holder];
7
8           if(!holder.send(((msg.value*rewardRate)/100)*(shares/
                totalSupply))){
9             throw;
10          }
11        }
12    }
```

Listing 5.1: Code showcasing the receivePayment function and the combined use of send and throw.

Interestingly, this example reiterates how important it is to exercise extreme caution while developing Ethereum contracts. It also shows that seemingly innocuous code (in this case the fact of rolling back changes if an operation fails) can be used in Ethereum in order to construct complex attacks with varying consequences.

In order to negate this attack, a possible approach then would be to change the handling of a failed `send`. Instead of throwing, the function should record which addresses have failed to receive their payment along with the amount that is due [132]. Another function should then be provided for individual attempts to retrieve these missed payments and a portal interface should be implemented to inform individual investors when such situations arise (and the corresponding actions they can take) [132]. In terms of the contracts, one possible fix is then shown in Listing **??** below [132].

```
1   contract Private is owned, SharesManager, hasProposals {
2
3     ...
4
5   mapping (address=>uint) missedRewards;
6
7     ...
8
9     function retrieveMissedRewards ( address _shareholder ){
10        uint sum = missedRewards[_shareholder];
11        if(sum <=  0 ){
12         throw;
13        }
14       if(! _shareholder.send(sum)){
15           throw;
16       }
17      }
18
19     ...
```

```
20
21    function receivePayment(uint _ID){
22        ...
23          for (uint i=0; i<nbShareholders; ++i){
24
25            address holder = shareholders[i];
26            uint shares= balances[holder];
27
28            if(!holder.send(((msg.value*rewardRate)/100)*(shares/
                  totalSupply))){
29
30             missedRewards[holder] = ((msg.value*rewardRate)/100)*(
                   shares/totalSupply);
31
32            }
33          }
34    }
35 }
```

Listing 5.2: Solution to the send attack vector.

5.6 Web Interface and Navigation Map

The end product of our implementation is a web portal that demonstrates how it is possible to use DLT in order to enable the Digital Economy. It's centre piece consists of the dedicated DAO monitoring hub, with all other significant parts of the portal either converging to or emanating from this space. This is illustrated in the portal Navigation Map in Fig. 5.5. Appendix A contains screenshots of the different pages appearing in this map, with the more interesting ones also described below.

The homepage is shown in Screenshot A.1. A very simple design was chosen with a single central option for newcomers: *Browse the DAOs*. For acquainted users, the navigation bar (Navbar) also contains all the links necessary to access the other parts of the website. The *Current DAO* link is special in the sense that it remembers the last DAO visited by the user and can be used to access it directly. If no DAO was yet visited, the link will prompt the user to connect to a DAO.

When accessing the Browsing space, the user has the choice between browsing for DAOs and Browsing for Jobs (Employment Proposals) as shown in Screenshots A.5 and A.6. As we can see from these screenshots, the users also have a range of search options in both cases and can directly navigate to the corresponding offer or DAO spaces from the search results. For users that know the address of the specific DAO they wish to connect to, there is also the option to do so directly through the connect form shown in Screenshot A.4.

User sign-up and login facilities are offered by the panel appearing on the right of the Navbar, as shown in Screenshot A.2. We have made it very accessible for users to sign up to the system, with only their Ethereum address and a user name necessary.

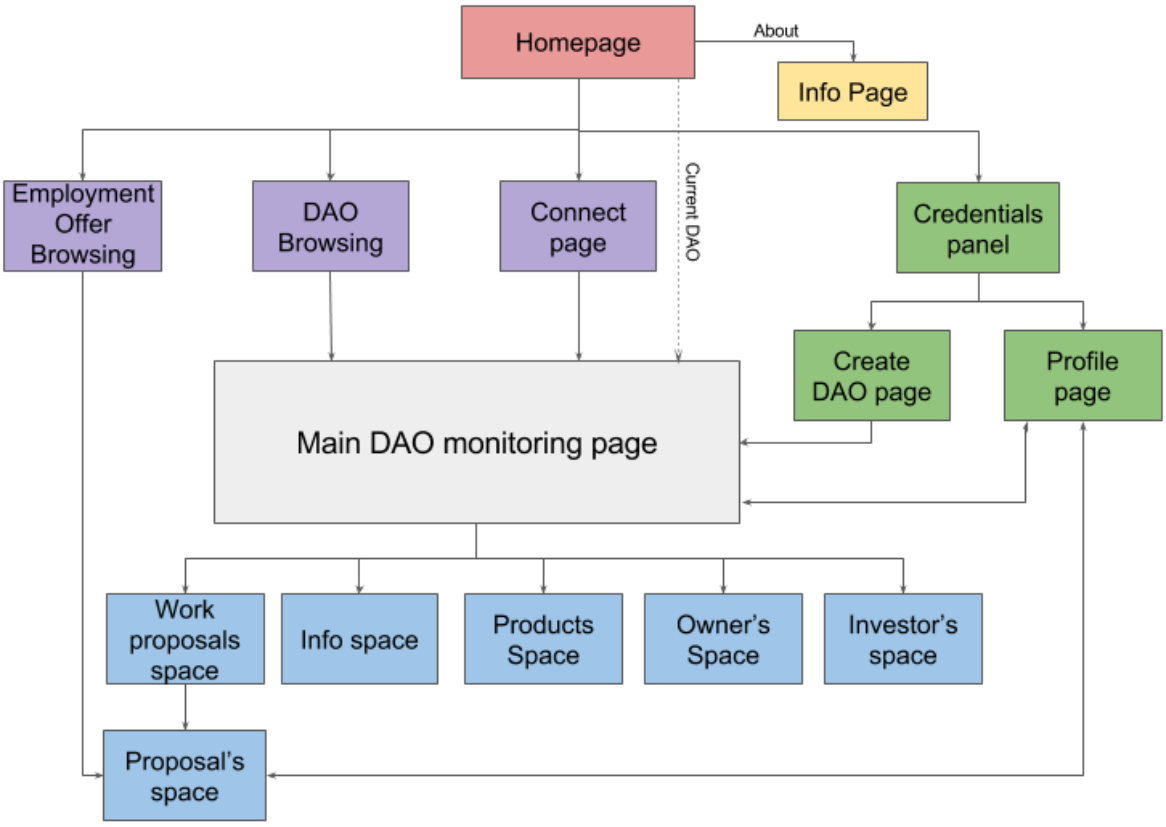


Figure 5.5: Navigation Map of our portal.

We believe this is in line with the decentralised/cryptocurrency spirit of accessibility with no control of identity by central authorities, while our system and the smart contracts in place make sure that malicious behaviour is prevented (users do have subsequently the option to provide as much personal information they see fit through the description facilities).

Once signed up, our users have the possibility of becoming owners of DAOs and contractors on employment offers. They further get access to the create new DAO form shown in Screenshot A.3 and their profile space shown in Screenshot A.14. On their profile page, users can add and then *Edit* their *Description*, which serves as a type of personal statement and gives a space for user to provide their contact details. The DAOs they own and their contractor tasks are also shown, with the possibility to navigate directly to the corresponding pages.

The main DAO monitoring page is shown in Screenshot A.7. This page is composed of a central panel with the main information concerning the DAO, and the choice between 5 different spaces through a navigation bar.

The information space shown in Screenshot A.7 contains the history of all past transactions of the DAO. Each of the transactions displayed also provides a link to is representation on an Ethereum Blockchain explorer.

The Employment proposal space is shown in Screenshot A.9. This space lists all the offers from this DAO, along with some essential information about each. As can be seen from the screenshot, the owner has the option to remove an offer, as long as it still remains in the recruiting phase. Through the provided link, a user can then access the dedicated proposal management dashboard shown in Screenshot A.10. This dashboard includes all the actions necessary to perform the full employment life-cycle relating to that particular proposal (cf. Fig. 5.2a).

The Owner's space, where most of the owner actions can be executed, is shown in Screenshot A.11. In particular, the owner can add employment offers through this space, which is done through the form shown in Screenshot A.12.

Finally, the Investor's space (shown in Screenshot A.8) contains the different possible investor actions and the Product's space (shown in Screenshot A.13) consists of the product shelf where the created content is sold and through which the owner can modify the associated product attributes.

Chapter 6

Review and Conclusions

6.1 Evaluation and Future Work

6.1.1 Evaluation

Our solution successfully depicts a complete model, which we named "Decentralised Startups", for enabling the economic interaction of system participants through a cryptocurrency empowered digital platform. This model succeeds in capturing the entire chain of the corporate life-cycle, namely funding, employment, content creation, content distribution and revenue management. Although our designs might still represent somewhat simplified workflows, we have captured the essence of the necessary features required to enable the vision of a novel economic system centred around Decentralised Autonomous Organisations. As such, our model can be used as a base framework for continued investigation in this direction, as we discuss in Section 6.1.2.

Our investigation also led to the implementation of a system prototype that constitutes a powerful demonstration of our principles. This product succeeds in exhibiting the core ideas of the full Decentralised Startups environment described in Chapter 4. The full workflow depicted in Fig.4.3 is represented, albeit with a smaller range of functionalities at each stage. In the following section we discuss improvements that would move our prototype away from the proof-of-concept realm, making it into a a full, production ready Minimum Viable Product (MVP).

6.1.2 Future Work

Research into the Decentralised Startups Ecosystem

We have identified some of the lines of work that can be carried out as a continuation of our investigations. All these contribute to the vision of creating a functional, complete and accessible ecosystem utilising Blockchain technology for empowering the Digital Economy, supporting new operational models for content creation and creating value in the greater context of worldwide economic development.

Future lines of work then contain the following:

Further development of the DAO processes, either by addition of new systems or expansion and improvement of the existing ones. Featured examples are the following:

- Further development of the governance schemes to increasingly mirror existing corporate processes. Such an approach could involve collaboration with the Boardroom team which seems to specialise in this domain [70].
- Addition of alternative types of DAOs not representing corporate constructs. These can include NGOs, humanitarian campaigns or personal crowd-funding support systems, and their addition would involve further research in the corresponding processes, workflows and the role of DAO smart contracts in their creation.
- Research into specialised DAOs according to industries. This line of work would require an interdisciplinary approach for defining processes inherent to each sector and translating those to smart contract workflows manageable through a digital platform.
- Investigation of shareholding mechanisms that mirror real stock logistics of supply and demand related value fluctuations.

Research into the contracting and content distribution workflows that focuses on maintaining the security and safeguards provided by Blockchain technology while delivering a user friendly and familiar experience to the users. Such research may include:

- Investigation of additional contracting schemes to cater more closely to various use cases.
- Further inquiry into how to maintain and promote honesty in the ecosystem, including with tasks that may be inherently manual (hence non-digital). This could involve the extension escrow workflows and contracts.
- Additional development of communication channels and feedback processes necessary in the realisation of the ecosystem. Included in this line of work could be the definition of a complex reward-based rating system (analogous for instance to the one in place on stack Exchange [133]). Interestingly, Colony seems to develop such an idea in its processes [66].

Extension of the horizons of the platform to interface with and include third party services. These may include:

- Investigation into Government and Public service inclusion mechanisms as part of the workflows. This would represent complex interdisciplinary research that might involve (1) considering a Government contract or address that would

automatically collect due Tax on the different operations of the DAOs and (2) considering a legal recourse mechanism where a Tribunal related address could impose sanctions or intervene to enforce justice.

- Interfacing with decentralised storage mechanisms such as Storj [134] for the digital contents created through the platform.
- Interfacing with distributed asset trading platforms such as BitShares [135] in order to enable trading of DAO shares.

Prototype Expansion

With regard to our implemented prototype we have also identified two possible future lines of work : (1) Moving away from a proof-of-concept towards a fully featured MVP and (2) Expanding the horizons of the product by adopting more of the features described in Chapter 4. Some of the tasks they would involve are the following:

From Prototype to MVP:

- Include communication channels between contractors/candidates and owners.
- Create a DAO feedback mechanism where contractors and investors can rate and comment on a given DAO.
- Include a final confirmation from the contractors before the hiring process completes and give them the ability to terminate the contact.
- Include the possibility for owners to preview the completed work through the portal without gaining full access to it.
- Carefully re-examine all contracts and possible security breaches they would contain. This includes the implementation of the fix presented in Section 5.5.5, and any interface functionality necessary to support it.
- Make user interface improvements such as guides for the product use or indications during transaction mining.
- Include the remaining integrity check mechanisms to the implementation (cf. Section 4.4.2).
- Carefully examine and cater for real network demands such as the approximate 15 second [17] delay between sending a transaction and completing the mining operation.

Into the digital Economy:

- Expand the employment workflow by adding the possibility for salary-based and escrow-based proposals.
- Add the other types of governance schemes (cf. Section 4.2.2).
- Move away from the inheritance-based contract system towards a composition-based structure (cf. Section 4.3).

6.2 Concluding Remarks

In our project we created a model for the digitalisation of economic interactions through smart contracts and Decentralised Autonomous Organisations. Our design utilises the Ethereum Blockchain as the backbone for building an ecosystem that includes every aspect of the corporate life-cycle: funding, operational governance, employment management, content creation and distribution and revenue management.

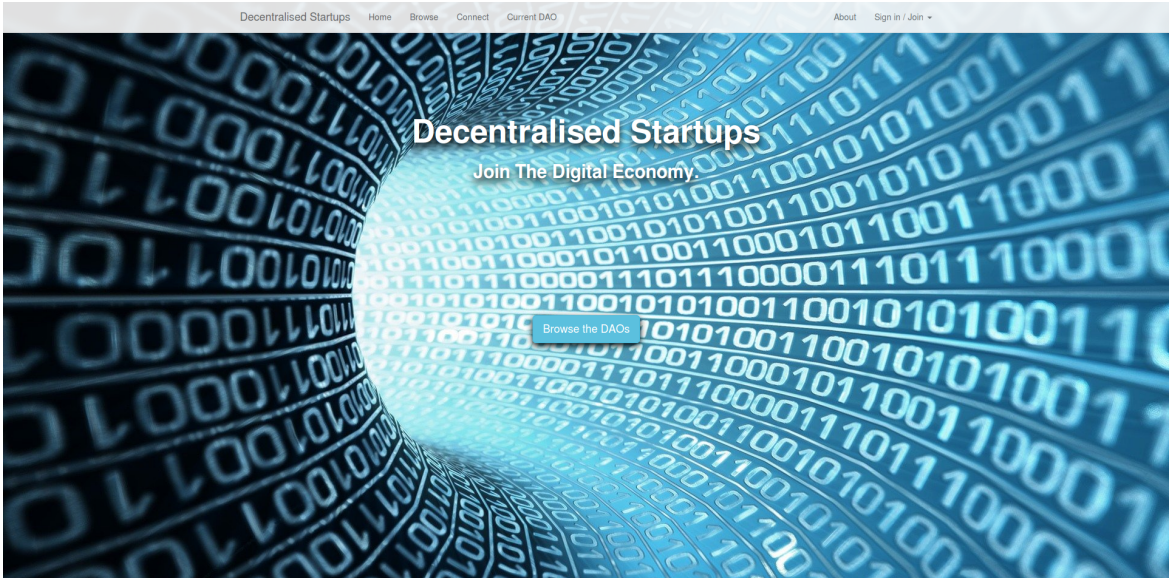
Regarding funding and operational governance, we have identified three different schemes corresponding essentially to three distinct types of DAO, namely a privately held organisation, a corporate entity and a fully public organisation directly controlled by its members. For content creation and employment we also cater to different use cases via associated schemes. Main distinctions between those are (1) whether they consist of a salary-based agreement or a one-off, result driven contract and (2) whether the task completion arbitration is made via mutual agreement or through the intermediary of an escrow contract. Finally, our workflows also define product sale and revenue redistribution channels. These are used to manage revenue in a secure and predetermined fashion, ensuring that qualified users receive their entitlements in a timely manner according to the agreed upon rules.

Our project also provides a proof-of-concept prototype platform for enabling such an ecosystem. This implementation includes the Private governance scheme and One-Off employment agreements. A company under this model would make use of freelance entrepreneurs or independent SME's in order to create its content and vend it through the portal. This proof of concept successfully demonstrates the core ideas put forward in this project, namely that Ethereum smart contracts and Decentralised Autonomous Organisations can be used in order to enable and organise the Digital Economy.

Having completed a base framework for the use of cryptocurrencies and smart contracts as supports for the Digital Economy, we have opened the doors to a vast amount of prospective research. Investigations can then aim to produce a more general depiction of reality, to increasing the connectivity of the created ecosystem with third-party systems or to creating production ready platforms that would aim to introduce and spread the solution to a general audience.

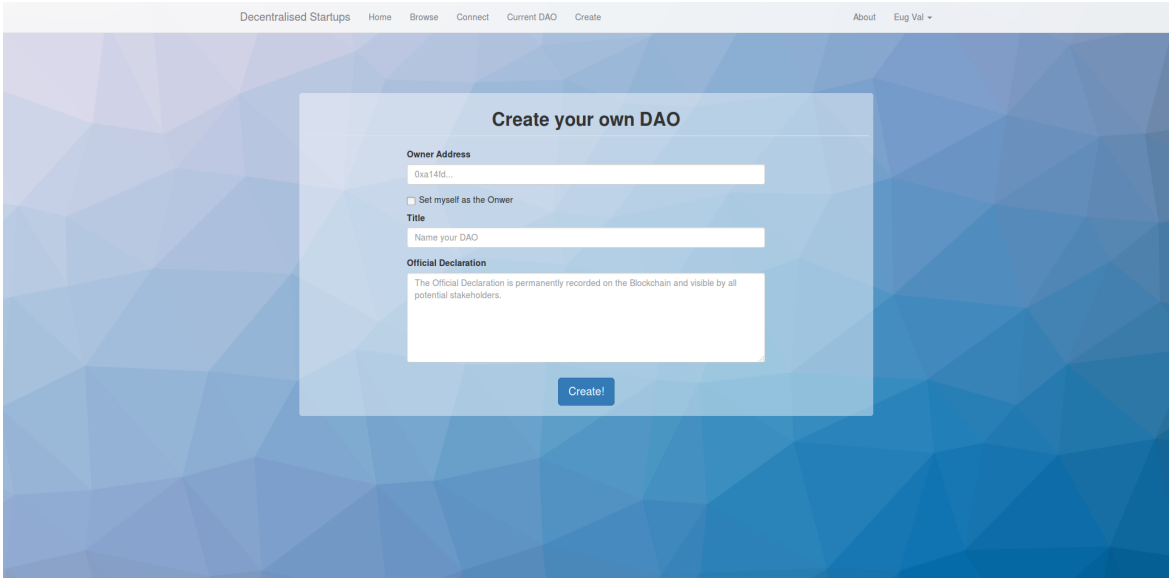
Appendix A

Prototype Screenshots

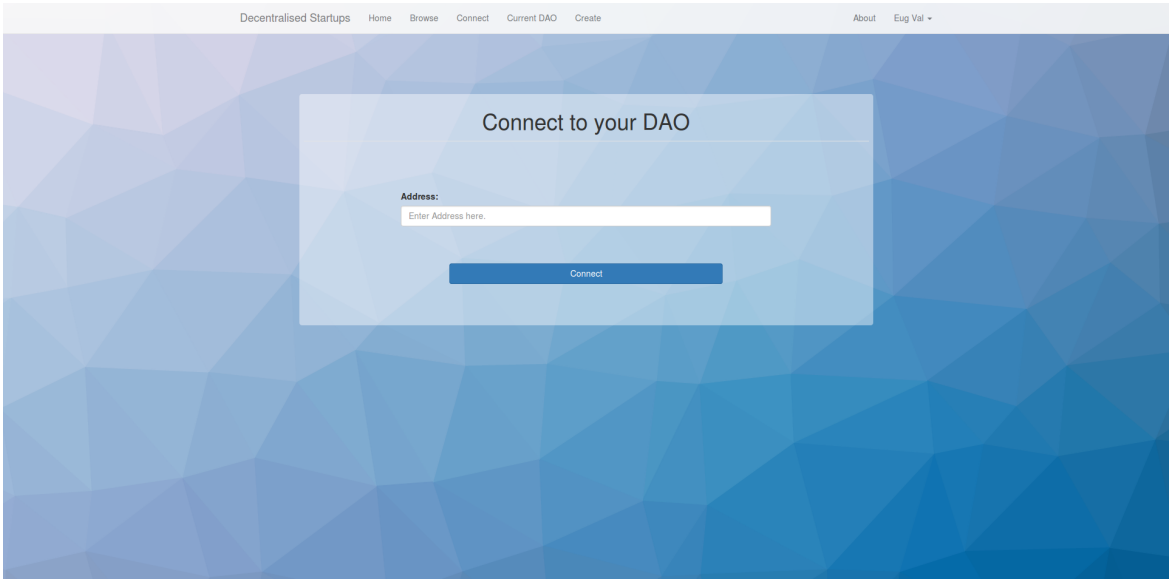


Screenshot A.1: Homepage (the Background image's source is [136]).

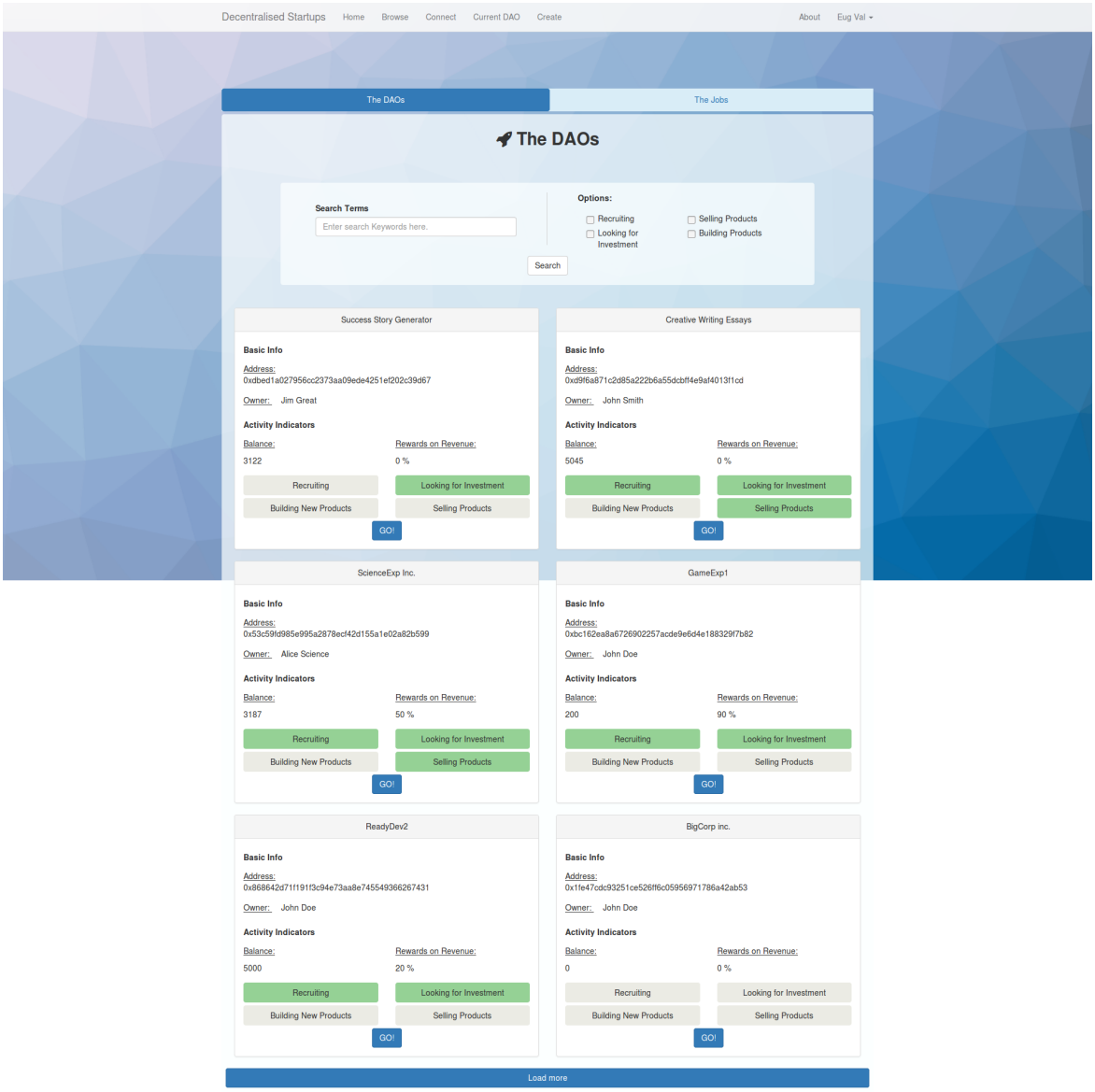
Screenshot A.2: Sign-up panel.



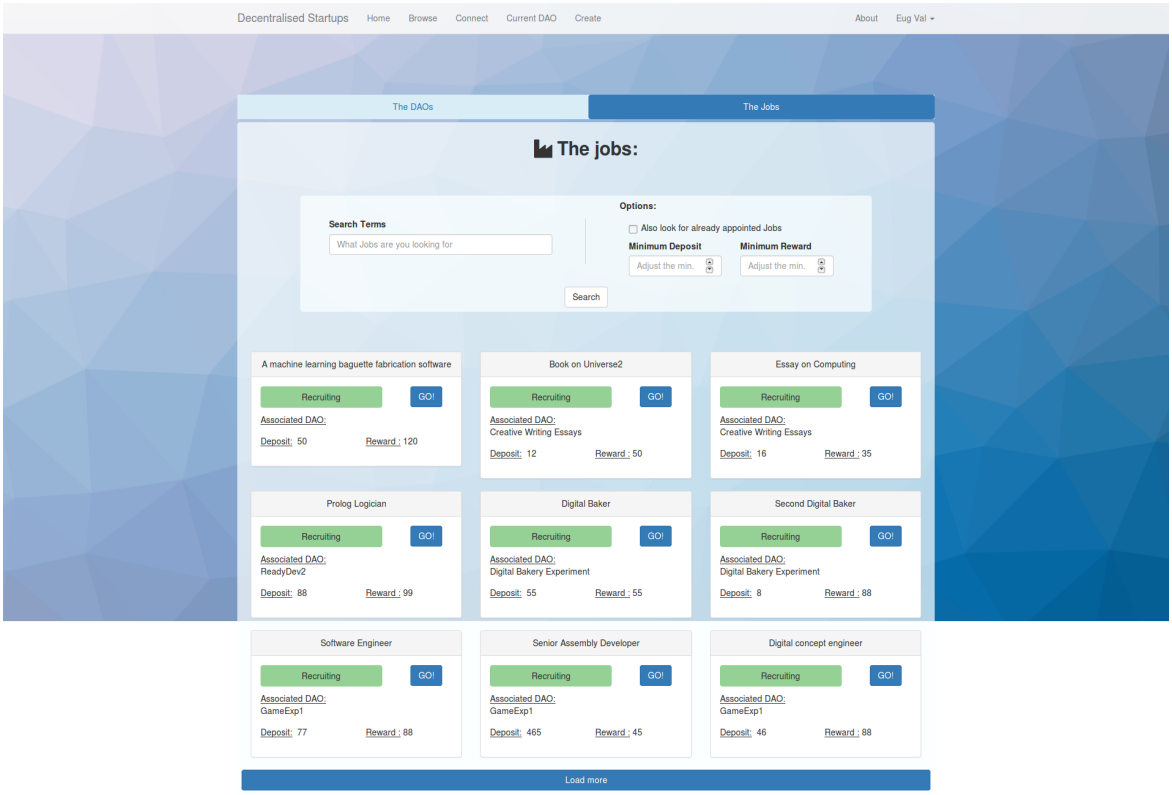
Screenshot A.3: Form for creating a new DAO.



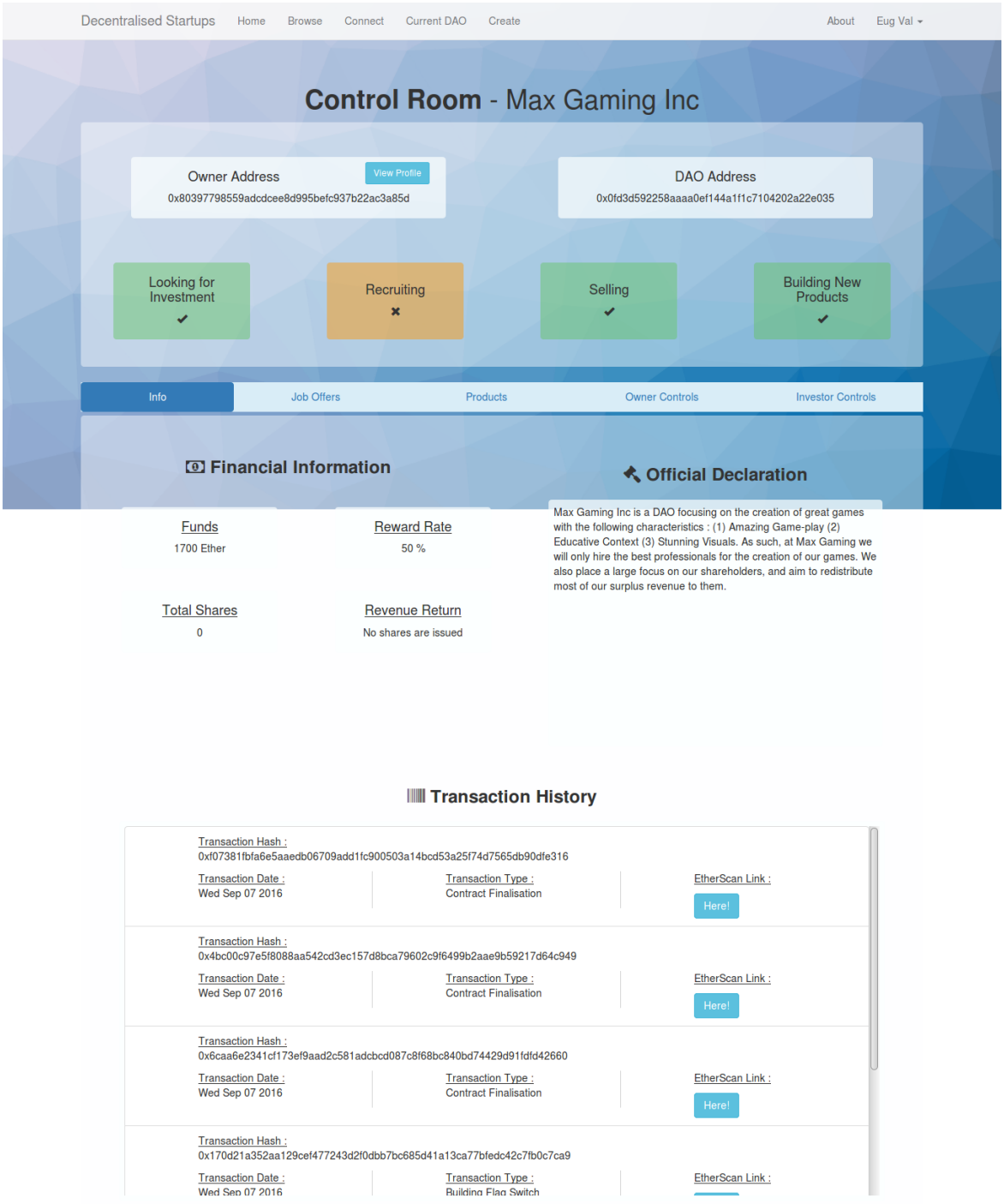
Screenshot A.4: Connecting to a DAO through it's address.



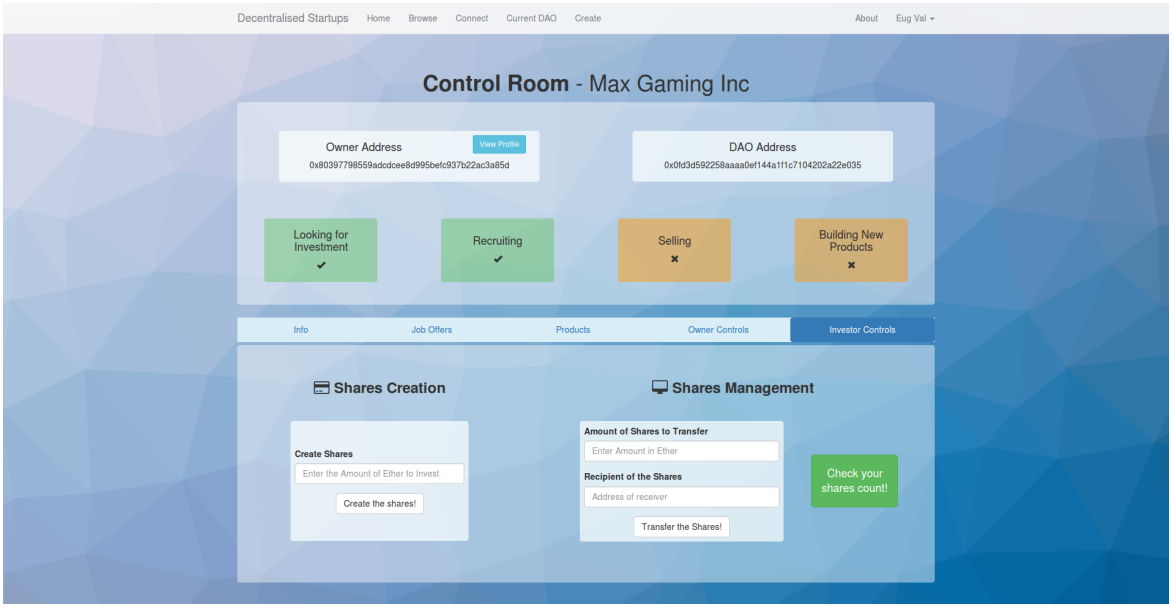
Screenshot A.5: Space for Browsing DAOs (Blue Background is fixed and the rest of the page scrolls on top).



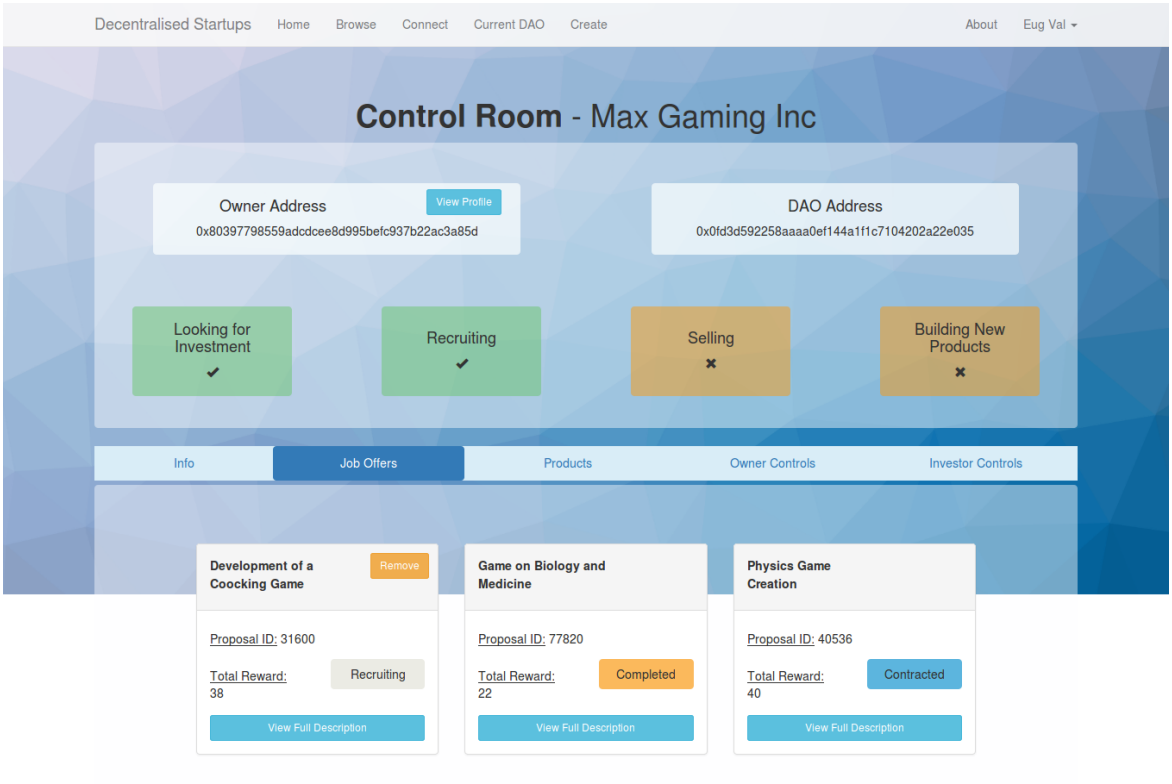
Screenshot A.6: Space for browsing employment offers (Blue Background is fixed and the rest of the page scrolls on top).



Screenshot A.7: Main Monitoring page: Information Display (Blue Background is fixed and the rest of the page scrolls on top).

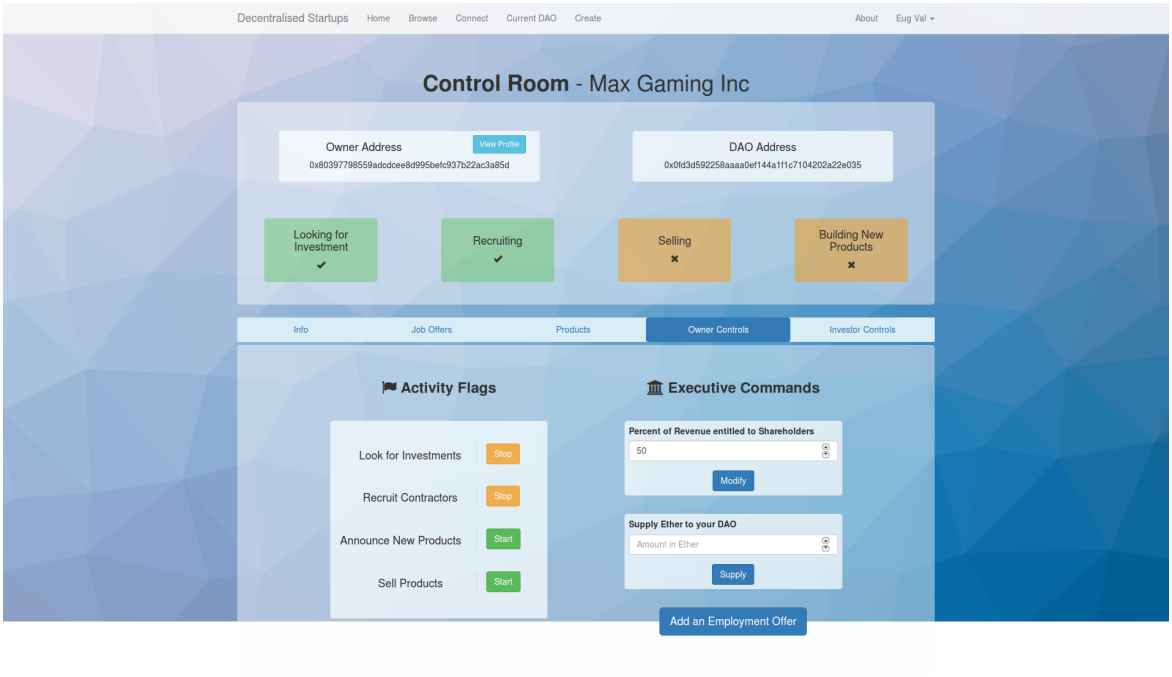


Screenshot A.8: Main Monitoring page: Investor Space.

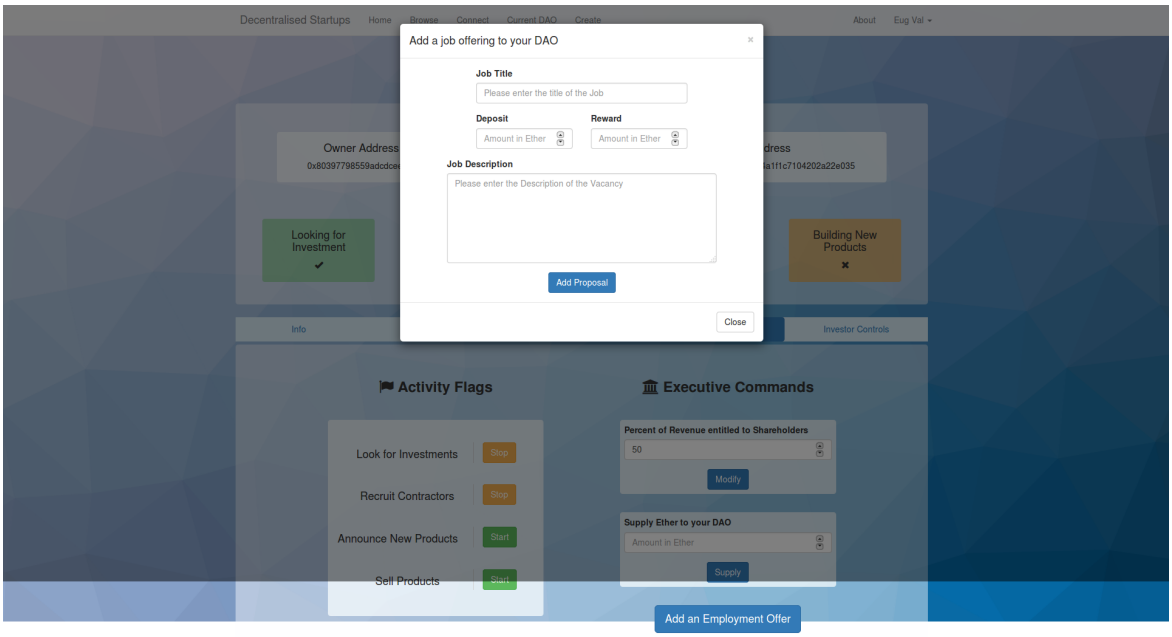


Screenshot A.9: Main Monitoring page: Employment offers.

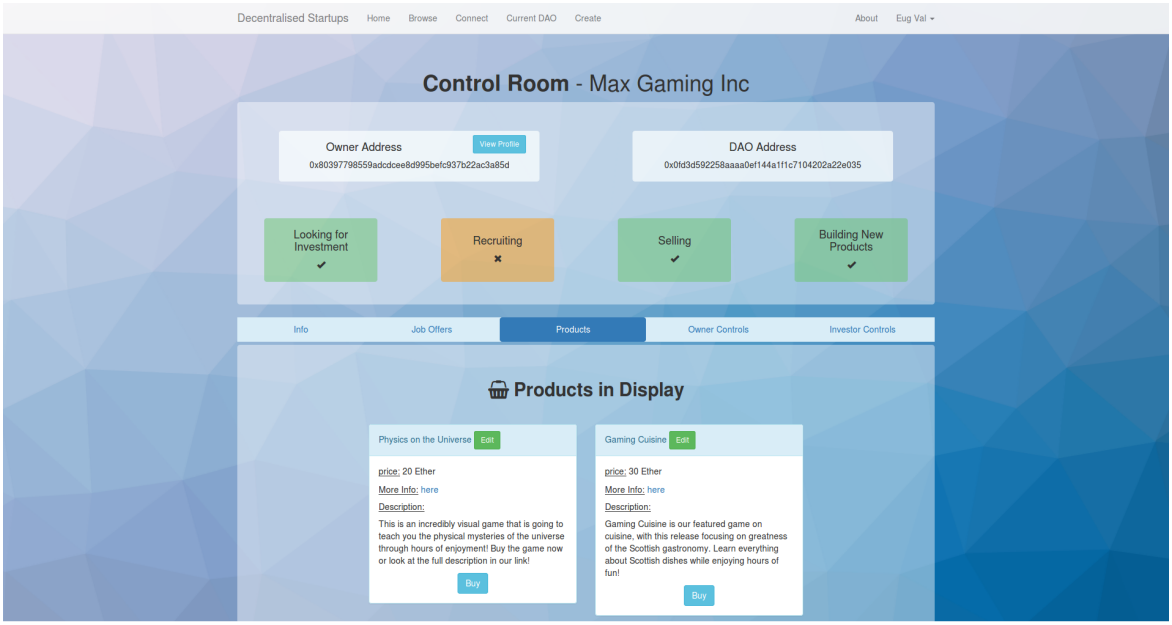
Screenshot A.10: Employment management space (Blue Background is fixed and the rest of the page scrolls on top).



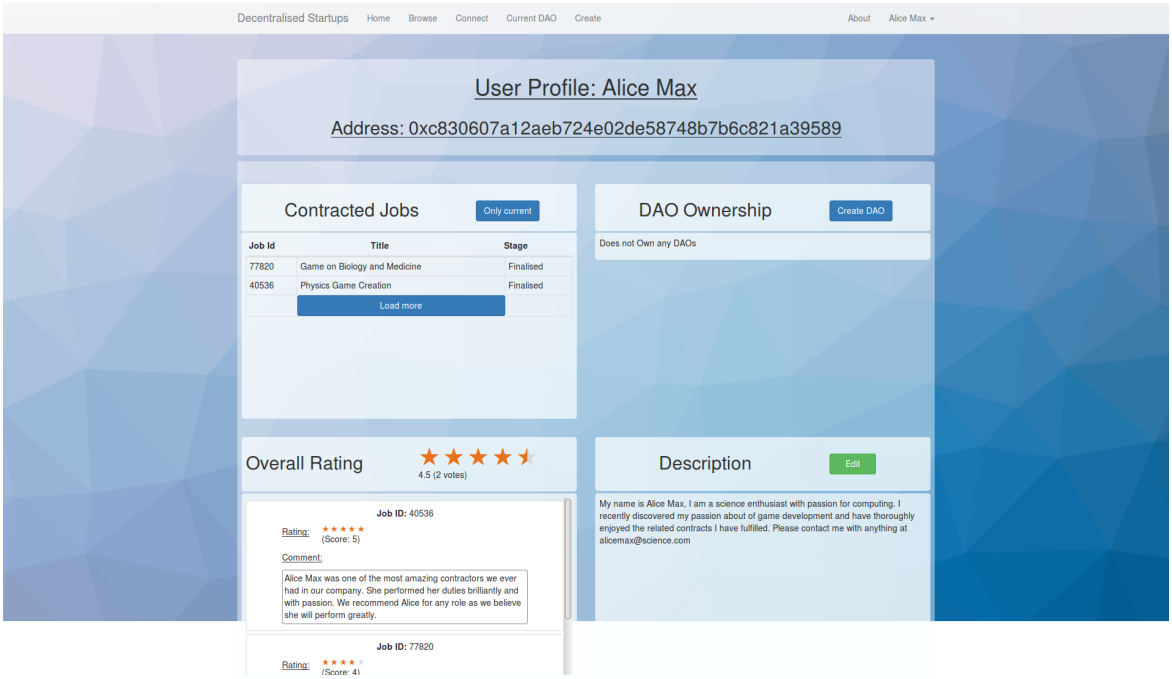
Screenshot A.11: Main Monitoring page: Owner Space.



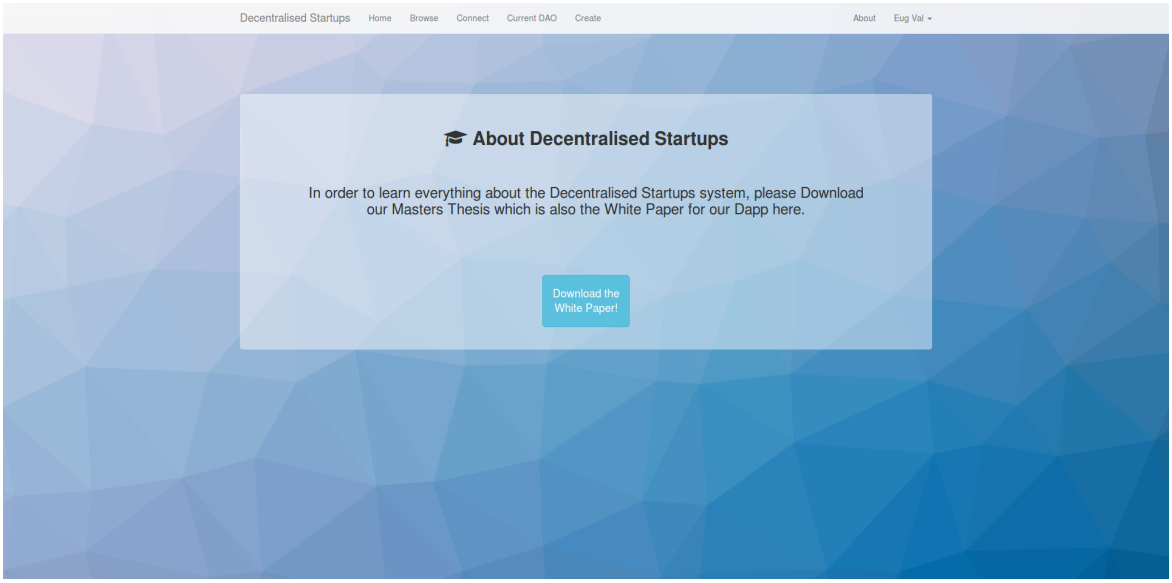
Screenshot A.12: Form for creating new proposals.



Screenshot A.13: Main Monitoring page: Product space.



Screenshot A.14: User profile page.



Screenshot A.15: Information page where our White paper (this report) can be downloaded.

Appendix B

Prototype Contracts

```
contract GovManager {
  address public owner;
  /*Constructor*/
  function GovManager() {
      owner = msg.sender;
  }

  /*Governance Modifier*/
  modifier onlyOwner {
    if (msg.sender != owner){
      throw;
    }
    _
  }

}
```

Listing B.1: Prototype Contracts: GovManager.

```
contract ProposalManager{
  mapping (uint => proposal)  public proposals;
  struct proposal{
    uint ID;
    string  description;
    uint  reward;
    uint deposit;
    bool completed;
    bool appointed;
    address  contractor;
    bool finalised;
  }

  /*Constructor*/
  function ProposalManager(){}

  /*Function for completing proposal*/
  function completeWork (uint _ID){
    if(proposals[_ID].ID == 0){
      throw;
    }
```

```
22
23      proposal p = proposals[_ID];
24
25      if(p.appointed == false || p.finalised==true || p.completed ==
            true){
26        throw;
27      }
28
29      if(msg.sender != p.contractor){
30        throw;
31      }
32      p.completed = true;
33    }
34
35  }
```

Listing B.2: Prototype Contracts: ProposalManager.

```
1   contract ShareManager {
2
3     uint public totalSupply;
4     address[] public shareholders;
5     mapping (address => uint) balances;
6     bool public investment;
7
8     /*Constructor*/
9     function ShareManager() {
10      totalSupply = 0;
11      msg.sender.send(msg.value);
12    }
13
14    /*function for creating shares*/
15    function createShares() returns (bool success){
16      if(!investment){
17        throw;
18      }
19
20      if(balances[msg.sender] == 0){
21        uint shareholderID = shareholders.length++;
22        shareholders[shareholderID] = msg.sender;
23      }
24      uint shares = msg.value / 1 ether;
25      balances[msg.sender] = balances[msg.sender] + shares;
26      totalSupply = totalSupply+ shares;
27      return true;
28    }
29
30    /*function for verifying owned amount of shares*/
31    function getShares() returns (uint shares){
32      return balances[msg.sender];
33    }
34
35  /*function for transfering shares*/
36    function transfer(address _to, uint _value) {
37      if(balances[_to] == 0){
38        uint shareholderID = shareholders.length++;
```

```
39        shareholders[shareholderID] = _to;
40      }
41
42      if (balances[msg.sender] < _value){
43        throw;
44      }
45
46      if (balances[_to] + _value < balances[_to]){
47         throw;
48      }
49
50      balances[msg.sender] -= _value;
51      balances[_to] += _value;
52    }
53
54  }
```

Listing B.3: Prototype Contracts: ShareManager.

```
1   contract Private is GovManager, ShareManager, ProposalManager {
2
3     bool public recruiting;
4     bool public building;
5     bool public production;
6     uint public rewardRate;
7     string public description;
8
9       /*Constructor*/
10
11      function Private(address _owner, string _desc) {
12        if(_owner != 0){
13          owner = _owner;
14        }else{
15          owner = msg.sender;
16        }
17
18        description = _desc;
19
20        investment = false;
21        recruiting = false;
22        building = false;
23        production = false;
24        rewardRate = 0;
25      }
26
27      /*Activity indicator manipulation*/
28
29      function toggleSharesIssue() onlyOwner  {
30        if(investment){
31          investment = false;
32        }else{
33          investment = true;
34        }
35      }
36
37      function toggleRecruiting() onlyOwner  {
```

```
38        if(recruiting){
39          recruiting = false;
40
41        }else{
42          recruiting = true;
43
44        }
45      }
46
47      function toggleBuilding() onlyOwner  {
48        if(building){
49          building = false;
50
51        }else{
52          building = true;
53
54        }
55      }
56
57      function toggleProduction() onlyOwner  {
58        if(production){
59          production = false;
60
61        }else{
62          production = true;
63
64        }
65      }
66
67      /*functions for Revenue Redistribution*/
68
69      function changeRate(uint percent)onlyOwner{
70          if(percent > 100){
71            throw;
72          }
73          if(percent<0){
74            throw;
75          }
76          rewardRate = percent;
77      }
78
79
80      function receivePayment(uint _ID){
81        if(!production){
82          throw;
83        }
84
85        if(proposals[_ID].ID == 0){
86          throw;
87        }
88
89        proposal p = proposals[_ID];
90
91        if(p.finalised == false){
92          throw;
```

```
93        }
94
95        if(totalSupply != 0){
96          uint nbShareholders = shareholders.length;
97          for (uint i=0; i < nbShareholders; ++i){
98            address holder = shareholders[i];
99            uint shares = balances[holder];
100           if(!holder.send(((msg.value*rewardRate)/100)*(shares/
                totalSupply))){
101             throw;
102           }
103         }
104     }
105     }
106
107
108
109     function fuel() onlyOwner{}
110
111
112     /*functions for Employment Management*/
113
114     function addProposal(uint _reward, uint _deposit, string _desc,
           uint _ID) onlyOwner {
115       if(_ID<0){
116         throw;
117       }
118
119       if(!recruiting || proposals[_ID].ID!=0 ){
120         throw;
121       }
122
123       proposal p = proposals[_ID];
124       p.ID = _ID;
125       p.reward = _reward;
126       p.deposit = _deposit;
127       p.completed = false;
128       p.appointed = false;
129       p.finalised = false;
130       p.contractor = 0;
131       p.description = _desc;
132     }
133
134     function removeProposal(uint _ID) onlyOwner{
135       if(proposals[_ID].ID == 0){
136         throw;
137       }
138
139       proposal p = proposals[_ID];
140
141       if(p.appointed == true){
142         throw;
143       }
144
145       p.ID = 0;
```

```
146
147     }
148
149
150     function hireContractor (address _contractor, uint _ID) onlyOwner
            {
151
152       if(!recruiting){
153         throw;
154       }
155
156       if(proposals[_ID].ID == 0){
157         throw;
158       }
159
160       proposal p = proposals[_ID];
161       if(p.appointed == true){
162         throw;
163       }
164
165       p.appointed = true;
166       p.contractor = _contractor;
167       if(!_contractor.send(p.deposit * 1 ether)){
168         throw;
169       }
170
171     }
172
173     function layoffContractor (uint _ID) onlyOwner{
174       if(proposals[_ID].ID == 0){
175         throw;
176       }
177
178       proposal p = proposals[_ID];
179
180       if(p.appointed ==false || p.finalised==true){
181         throw;
182       }
183
184       p.appointed = false;
185       p.completed= false;
186       p.contractor = 0;
187
188     }
189
190
191     function finalise(uint _ID) onlyOwner{
192       if(proposals[_ID].ID==0){
193         throw;
194       }
195
196       proposal p = proposals[_ID];
197
198       if(p.appointed == false || p.completed ==false || p.finalised
              == true){
```

```
199           throw;
200         }
201
202         uint payment = p.reward;
203         p.finalised = true;
204
205         if(!p.contractor.send(payment * 1 ether)){
206           throw;
207         }
208
209       }
210
211       /*fallback Function*/
212
213       function () {
214        throw;
215       }
216
217   }
```

Listing B.4: Prototype Contracts: Private.

Bibliography

[1] Steven White. Statistical release, business population estimates for the uk and regions 2015. Technical report, Department for Business, Innovation & Skills, UK Government, October 2015. Crown copyright 2015, https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/467443/bpe_2015_statistical_release.pdf. pages 1

[2] John Kitching. Exploring the uk freelance workforce in 2015. Technical report, IPSE (The Association of Independent Professionals and Self Employed) and the Small Business Research Centre at Kingston University. https://www.ipse.co.uk/sites/default/files/documents/research/Exploring-uk-freelance-workforce-2015-report-v1.pdf. pages 1

[3] freelance:UK. What are the challenges of freelancing? http://www.freelanceuk.com/news/3359.shtml, 27 November 2009. [Accessed: August 2016]. pages 1

[4] Business Reporter. The debate: What will be the main challenges facing smes in 2016? [Online Discussion], http://business-reporter.co.uk/2015/11/15/the-debate-what-will-be-the-main-challenges-facing-smes-in-2016/, 15 November2015. [Accessed: 28 August 2016]. pages 1

[5] Sarah Watts. What are the biggest problems for small businesses? Director Publications Ltd, http://www.director.co.uk/biggest-problems-facing-smes-sarah-watts-blog-22-may-2015/, 22 May 2015. [Accessed:23 August 2016]. pages 1

[6] Phil Smith. What challenges will smes face in 2016? Moorfields Advisory Ltd t/a Moorfields, http://www.moorfieldscr.com/media-centre/business-insights/what-challenges-will-smes-face-in-2016/, 19 December 2015. [Accessed:23 August 2016]. pages 1

[7] Technology Strategy Board. Digitial economy strategy. Technical report, Innovate UK. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/404743/Digital_Economy_Strategy_2015-18_Web_Final2.pdf. pages
1

[8] Government Chief Scientific Adviser Sir Mark Walport. Distributed ledger technology: beyond block chain. Technical report, Government Office for Science. Crown copyright, https://www.gov.uk/government/uploads/system/uploads/attachment_data /file/492972/gs-16-1-distributed-ledger-technology.pdf. pages 1, 5, 7

[9] European Commission. The importance of the digital economy. https://ec.europa.eu/growth/sectors/digital-economy/importance_en. [Accessed: 23 August 2016]. pages 1, 2

[10] European Commission. Digital economy policy. https://ec.europa.eu/growth/sectors/digital-economy/policy_en. [Accessed: 23 August 2016]. pages 1, 2

[11] Bill Wood. The impact of the digital economy on your industry. *Capgemini*, March 2015. https://www.capgemini.com/blog/capping-it-off/2015/03/the-impact-of-the-digital-economy-on-your-industry ,[Accessed: 24 August 2016]. pages 1, 2

[12] Simon Taylor. Blockchain: understanding the potential. Technical report, Barclays Bank PLC, July 2015. Available at : https://www.barclayscorporate.com/content/dam/corppublic/corporate /Documents/insight/blockchain_understanding_the_potential.pdf, [Accessed: 16 May 2016]. pages 1

[13] Stan Higgins. Jpmorgan unveils 'juno' project at hyperledger blockchain meeting. CoinDesk, http://www.coindesk.com/jpmorgan-juno-hyperledger-blockchain/, 03 March 2016. [Accessed: 12 June 2016]. pages 1

[14] Bitcoin. Bitcoin Project , https://bitcoin.org/en/, 2009-2016. [Accessed: 22 March 2016]. pages 1, 7

[15] Blockchain Technologies. Blockchain technology explained. BlockchainTechnologies.com , http://www.blockchaintechnologies.com/blockchain-definition. [Accessed: June 2016]. pages 1, 7

[16] Ethereum Foundation. Ethereum project. https://www.ethereum.org/. [Accessed: February 2016]. pages 1, 7, 16, 23, 38

[17] Ethereum community. Homestead documentation. http://www.ethdocs.org/en/latest/index.html. [Accessed: 28 March 2016]. pages 1, 16, 17, 18, 19, 20, 21, 23, 24, 25, 39, 66, 82

[18] Josh Stark. Making sense of blockchain smart contracts. CoinDesk, http://www.coindesk.com/making-sense-smart-contracts/, 4 June 2016. [Accessed: 24 July 2016]. pages 1, 16

[19] Nick Szabo. Smart contracts. http://szabo.best.vwh.net/smart.contracts.html, 1994. [Accessed: 3 July 2016]. pages 1, 16

[20] Jay Cassano. What are smart contracts? cryptocurrency's killer app. Fast Company & Inc, Mansueto Ventures, http://www.fastcompany.com/3035723/app-economy/smart-contracts-could-be-cryptocurrencys-killer-app, 17 September 2014. [Accessed: 05 June 2016]. pages 1

[21] Vitalik Buterin. Daos, dacs, das and more: An incomplete terminology guide. Ethereum Blog, https://blog.ethereum.org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide/, 6 May 2014. [Accessed: 18 May 2016]. pages 1, 26

[22] DJ Pangburn. The humans who dream of companies that won't need us. Fast Company & Inc, Mansueto Ventures, LLC http://www.fastcompany.com/3047462/the-humans-who-dream-of-companies-that-wont-need-them, 19 June 2015. [Accessed: 05 June 2016]. pages 1, 2, 26

[23] Stephan Tual. A primer to decentralized autonomous organizations (daos). Slock.it Blog, https://blog.slock.it/a-primer-to-the-decentralized-autonomous-organization-dao-69fb125bd3cd#.44bc30pub, 03 March 2016. [Accessed: 02 June 2016]. pages 2

[24] Vitalik Buterin. A next-generation smart contract and decentralized application platform. Technical report. [Accessed:16 March 2016]. pages 2, 9, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 24, 26

[25] Ian Allison. 'decentralised uber' arcade city wins europe's biggest blockchain competition. *International Business Times*, May 2016. http://www.ibtimes.co.uk/decentralised-uber-arcade-city-wins-europes-biggest-blockchain-competition-1560910,[Accessed: 05 September 2016]. pages 2

[26] Uber Technologies Inc. Uber. https://www.uber.com/ ,[Accessed: 24 August 2016]. pages 2

[27] Christoph Jentzsch. Decentralized autonomous organization to automate governance final draft - under review. Github, https://github.com/slockit/DAO/tree/master/paper. [Accessed: 28 May 2016]. pages 4, 26, 31, 32, 33, 37, 38, 39, 44, 46, 53, 71

[28] Kate. What is the dao and why is it the biggest crowdfunding project in history? LTP, https://letstalkpayments.com/what-is-the-dao-and-why-is-it-the-biggest-crowdfunding-project-in-the-history/, 21 May 2016. [Accessed: 05 August 2016]. pages 4, 30

[29] FirstPartner. 2016 blockchain ecosystem market map. http://firstpartner.net/content/2016-blockchain-ecosystem-market-map, July 2016. [Accessed: 04 August 2016]. pages 6

[30] Tim Swanson. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. http://www.ofnumbers.com/wp-content/uploads/2015/04/Permissioned-distributed-ledgers.pdf, 6 April 2015. [Accessed: 04 August 2016]. pages 7

[31] Bitscan. Permissioned and unpermissioned blockchains, part 1. http://bitscan.com/articles/permissioned-and-unpermissioned-blockchains-part-1, 02 February 2016. [Accessed: 04 August 2016]. pages 7

[32] Ripple. https://ripple.com. [Accessed: 04 August 2016]. pages 7

[33] Linux Foundation Collaborative Project. Hyperledger project. The Linux Foundation, https://www.hyperledger.org. [Accessed: 23 July 2016]. pages 7

[34] CuriousInventor. How bitcoin works under the hood. Youtube, https://www.youtube.com/watch?v=Lx9zgZCMqXE, 14 July 2013. [Accessed: 04 February 2016]. pages 7, 8, 9, 11, 13, 15

[35] Zulfikar Ramzan. Bitcoin: What is it? Khan Academy, https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-what-is-it. [Accessed: 05 June 2016]. pages 7, 8, 9, 11, 13, 15

[36] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Bitcoin Project, https://bitcoin.org/bitcoin.pdf, 2008. [Accessed: 26 May 2016]. pages 7

[37] Fraida Fund. Bitcoin: reaching consensus in distributed systems. DISCUS, http://witestlab.poly.edu/blog/get-rich-on-fake-bitcoins/, 07 March, 2016. [Accessed: 23 May 2016]. pages 8

[38] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media Inc., United States of America, first edition edition, 2014. pages 8, 9, 10, 11, 12, 13, 14, 15, 16

[39] Bitcoin developer guide. Bitcoin Project, https://bitcoin.org/en/developer-guide#block-chain. [Accessed: 26 July 2016]. pages 9, 11, 12, 13, 14

[40] BitcoinWiki. Secp256k1. https://en.bitcoin.it/wiki/Secp256k1. [Accessed: 28 July 2016]. pages 9

[41] Chapter 4. keys, addresses, wallets. OReilly Media, Inc, http://chimera.labs.oreilly.com/books/1234000001802/ch04.html. [Accessed: 17 August 2016]. pages 10

[42] bicoinmining.com. Hesiod Services LLC, https://www.bitcoinmining.com/bitcoin-mining-pools/. [Accessed: 16 August 2016]. pages 15

[43] Ethereum. Introduction to smart contracts. https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html. [Accessed: 05 April 2016]. pages 16, 21, 22, 23, 24, 25, 27, 32, 40

[44] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf. [Accessed: 3 July 2016]. pages 17, 18, 20

[45] Ethereum. Ethereum wiki. Github, https://github.com/ethereum/wiki/wiki. [Accessed: 14 March 2016]. pages 18, 20, 21, 39, 59, 65, 66

[46] work2heat. Understanding the ethereum trie. https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/, 4 June 2014. [Accessed: 28 July 2016]. pages 19

[47] Vitalik Buterin. State tree pruning. Ethereum Blog, https://blog.ethereum.org/2015/06/26/state-tree-pruning/, 26 June 2015. [Accessed: 28 July 2016]. pages 19

[48] Ethereum. Design rationale. Github, https://github.com/ethereum/wiki/wiki/Design-Rationale. [Accessed: 02 July 2016]. pages 19, 20

[49] Vitalik Buterin. Merkling in ethereum. Ethereum Blog, https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/, 15 November 2015. [Accessed: 28 July 2016]. pages 19, 20

[50] Ethereum. Ethash. Github, https://github.com/ethereum/wiki/wiki/Ethash. [Accessed: 02 July 2016]. pages 20

[51] Ethereum. Mining. Github, https://github.com/ethereum/wiki/wiki/Mining. [Accessed: 02 July 2016]. pages 20

[52] Refsnes Data. W3schools javascript. http://www.w3schools.com/js/, 1999-2016. [Accessed: 01 June 2016]. pages 21

[53] Ethereum. Web3 javascript dapp api. Ethereum Wiki, Github, https://github.com/ethereum/wiki/wiki/JavaScript-API. [Accessed: 24 March 2016]. pages 21, 23, 24, 25, 60, 65, 66

[54] jimkberry. What is the ethereum transaction data structure? Stack Exchange, [Online Discussion], http://ethereum.stackexchange.com/questions/1990/what-is-the-ethereum-transaction-data-structure, March 2016. [Accessed: 12 July 2016]. pages 21

[55] ethereum/mist. Ethereum wallet and mist beta 0.8.2. Github, https://github.com/ethereum/mist/releases. [Accessed: 15 June 2016]. pages 23, 38, 39

[56] Ethereum Wiki. Ethereum contract abi. Github, https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI. [Accessed: 10 July 2016]. pages 23, 38

[57] Ethereum Foundation. How to build a democracy on the blockchain. https://www.ethereum.org/dao. [Accessed: 3 July 2016]. pages 26, 27, 28, 37, 38, 39, 44, 46, 53, 71

[58] Ethereum Community. The ethereum foundation. https://ethereum-homestead.readthedocs.io/en/latest/introduction/foundation.html#foundation. [Accessed: 12 July 2016]. pages 27

[59] Alex Van de Sande. Ethereum in practice part 1: how to build your own cryptocurrency without touching a line of code. Ethereum Blog, https://blog.ethereum.org/2015/12/03/how-to-build-your-own-cryptocurrency/, 03 December 2015. [Accessed: 23 May 2016]. pages 27, 53

[60] Alex Van de Sande. Ethereum in practice part 2: how to build a better democracy in under a 100 lines of code. Ethereum Blog, https://blog.ethereum.org/2015/12/04/ethereum-in-practice-part-2-how-to-build-a-better-democracy-in-under-a-100-lines-of-code/, 04 December 2015. [Accessed: 23 May 2016]. pages 27, 53

[61] Alex Van de Sande. Ethereum in practice part 3: how to build your own transparent bank on the blockchain. Ethereum Blog, https://blog.ethereum.org/2015/12/07/ethereum-in-practice-part-3-how-to-build-your-own-transparent-bank-on-the-blockchain/, 07 December 2015. [Accessed: 23 May 2016]. pages 27, 53

[62] Ethereum Foundation. Create your own crypto-currency with ethereum. https://www.ethereum.org/token. [Accessed: 9 July 2016]. pages 27

[63] Ethereum. Solidity documentation faq. https://solidity.readthedocs.io/en/latest/frequently-asked-questions.html. [Accessed: July 2016]. pages 27

[64] tcimoli. How does solidity ".call" work? Ethereum Community Forum, [Online Discussion],

https://forum.ethereum.org/discussion/6822/how-does-solidity-call-work, May 2016. [Accessed: 12 July 2016]. pages 27

[65] Bilthon. what does 'p.recipient.call.value(p.amount * 1 ether)(transactionbytecode);' do? Stack Exchange, [Online Discussion], http://ethereum.stackexchange.com/questions/2370/what-does-p-recipient-call-valuep-amount-1-ethertransactionbytecode-do, March 2016. [Accessed: 12 July 2016]. pages 27

[66] Colony. https://colony.io. [Accessed: 27 June 2016]. pages 28, 50, 81

[67] Jack du Rose. Colony (decentralized autonomous organisation). Devcon1 - Ethereum Developer Conference, London, September 2015. [Talk Accessed on Youtube], https://www.youtube.com/watch?v=ZvK3ZI1JX0Q. [Accessed: 03 June 2016]. pages 28

[68] HN London. Jack du rose- colony: A platform for decentralised autonomous companies using the ethereum blockchain. Vimeo, https://vimeo.com/148340526. [Accessed: 3 Aug 2016]. pages 28

[69] DavidJohnstonCEO. The general theory of decentralized applications, dapps. Github, https://github.com/DavidJohnstonCEO/DecentralizedApplications. [Accessed: 08 June 2016]. pages 29

[70] Nick Dodson. Boardroom. Empowered by Consensys, http://boardroom.to/. [Accessed: 28 May 2016]. pages 29, 81

[71] Nick Dodson. Boardroom: A next-generation decentralized governance apparatus. http://boardroom.to/BoardRoom_WhitePaper.pdf. [Accessed: 01 June 2016]. pages 29, 44

[72] Nick Dodson. Devcon1: Weifund & boardroom - nick dodson. Devcon1 - Ethereum Developer Conference, London, September 2015. [Talk Accessed on Youtube], https://www.youtube.com/watch?v=miaxf6BI6Wc. [Accessed: 02 June 2016]. pages 29

[73] Otonomos BCC Pte. Ltd. Otonomos dashboard. http://otonomos.com/dashboard/. [Accessed: 08 August 2016]. pages 29, 37, 38

[74] Otonomos. Otonomos x the dao. Reddit, [Online Discussion], https://www.reddit.com/r/ethereum/comments/4jkstu/otonomos_x_the_dao/. [Accessed: 08 August 2016]. pages 29

[75] Slock.it. Slock.it UG, https://slock.it/index.html. [Accessed: 28 May 2016]. pages 29

[76] The New York Times Nathaniel Popper. A venture fund with plenty of virtual capital, but no capitalist. The New York Times,

http://www.nytimes.com/2016/05/22/business/dealbook/crypto-ether-bitcoin-currency.html?_r=4&referer=, 21 May 2016. [Accessed: 04 August 2016]. pages 30

[77] Michael del Castillo. The dao: Or how a leaderless ethereum-based organization raised $50 million. CoinDesk, http://www.coindesk.com/the-dao-just-raised-50-million-but-what-is-it/, 12 May 2016. [Accessed: 05 August 2016]. pages 30

[78] Stephan Tual. No dao funds at risk following the ethereum smart contract recursive call bug discovery. Slock.it Blog, https://blog.slock.it/no-dao-funds-at-risk-following-the-ethereum-smart-contract-recursive-call-bug-discovery-29f482d348b#.vr9yrewne, 12 June 2016. [Accessed: 25 July 2016]. pages 30, 31

[79] Michael del Castillo. The dao attacked: Code issue leads to $60 million ether theft. CoinDesk, http://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft/, 17 June 2016. [Accessed: 06 August 2016]. pages 30

[80] Vitalik Buterin. Critical update re: Dao vulnerability. Ethereum Blog, https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/, 17 June 2016. [Accessed: 06 August 2016]. pages 30

[81] David Siegel. Understanding the dao attack. CoinDesk, http://www.coindesk.com/understanding-dao-hack-journalists/, 25 June 2016. [Accessed: 06 August 2016]. pages 30

[82] Edan Yago. Why the wrong response to the dao attack could kill ethereum. CoinDesk, http://www.coindesk.com/ethereum-response-dao-kill/, June 20,2016. [Accessed: 06 August 2016]. pages 30

[83] Peter Vessenes. Point / counterpoint: Ethereum miners should blacklist thedao theft. http://vessenes.com/point-counterpoint-ethereum-miners-should-blacklist-thedao-theft/, 18 June 2016. [Accessed: 06 August 2016]. pages 30

[84] Michael del Castillo. Ethereum executes blockchain hard fork to return dao funds. CoinDesk, http://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds/, 20 July 2016. [Accessed: 07 August 2016]. pages 30

[85] Peter Szilagyi. Dao wars: Your voice on the soft-fork dilemma. Ethereum Blog, https://blog.ethereum.org/2016/06/24/dao-wars-youre-voice-soft-fork-dilemma/, 24 June 2016. [Accessed: 07 August 2016]. pages 30, 33

[86] luke jr and CodeShark. bips/bip-0123.mediawiki. Github, https://github.com/bitcoin/bips/blob/master/bip-0123.mediawiki#1_Consensus_Layer. [Accessed: 07 August 2016]. pages 30

[87] Aaron van Wirdum. Why some changes to bitcoin require consensus: Bitcoins 4 layers. Bitcoin Magazine, BTC Inc, https://bitcoinmagazine.com/articles/why-some-changes-to-bitcoin-require-consensus-bitcoin-s-layers-1456512578, 26 February 2016. [Accessed: 07 August 2016]. pages 30

[88] BitcoinCore. Bitcoin core version history. Bitcoin Project, https://bitcoin.org/en/version-history. [Accessed:07 August 2016]. pages 30

[89] CoinDesk Stan Higgins. Will ethereum fork? dao attack prompts heated debate. CoinDesk, http://www.coindesk.com/will-ethereum-hard-fork/, 17 June, 2016. [Accessed:06 August 2016]. pages 30

[90] Anonymous. An open letter. Pastebin http://pastebin.com/CcGUBgDG, June 2016. [Accessed: 07 August 2016]. pages 30

[91] Stan Higgins. Ethereum developers launch white hat counter-attack on the dao. CoinDesk, http://www.coindesk.com/ethereum-developers-draining-dao/, 21 June 2016. [Accessed: 08 August 2016]. pages 30

[92] Stan Higgins. Dao debacle escalates: Attacker counter-attacks ethereum developers. CoinDesk, http://www.coindesk.com/dao-counter-attack-ethereum/, 22 June 2016. [Accessed: 07 August 2016]. pages 30, 31

[93] CryptoBond. What is ethereum classic. CryptoCompare, https://www.cryptocompare.com/coins/guides/what-is-ethereum-classic/, 03 Aug 2016. [Accessed: 08 August 2016]. pages 30

[94] Vitalik Buterin. Hard fork completed. Ethereum Blog, https://blog.ethereum.org/2016/07/20/hard-fork-completed/, July 20, 2016. [Accessed: 07 August 2016]. pages 30

[95] Peter Vessenes. More ethereum attacks: Race-to-empty is the real deal. http://vessenes.com/more-ethereum-attacks-race-to-empty-is-the-real-deal/, June 2016. [Accessed: 16 August 2016]. pages 31, 32, 33

[96] Phil Daian. Analysis of the dao exploit. *Hacking, Distributed*, June 2016. http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/ ,[Accessed: 07 August 2016]. pages 32, 33, 40

[97] Peter Vessenes. Advice for dao 2.0s. http://vessenes.com/advice-for-dao-2-0s/, 22 July 2016. [Accessed: 08 August 2016]. pages 33, 40

[98] Solidity realtime compiler and runtime. Github Pages, https://ethereum.github.io/browser-solidity/#version=soljson-latest.js. [Accessed: 01 June 2016]. pages 38, 66

[99] slock.it. The dao, about. https://daohub.org/about.html. [Accessed: 28 May 2016]. pages 39

[100] Eva Shon. A 101 noob intro to programming smart contracts on ethereum. Consensys, https://medium.com/@ConsenSys/a-101-noob-intro-to-programming-smart-contracts-on-ethereum-695d15c1dab4#.yoqwxgma4, 29 October 2015. [Accessed: 15 May 2016]. pages 39

[101] ESCROW.COM. What is escrow? how does escrow work? https://www.escrow.com/what-is-escrow. [Accessed: 16 August 2016]. pages 50

[102] Paul S. How do i know when i've run out of gas programmatically? Stack Exchange, [Online Discussion], http://ethereum.stackexchange.com/questions/1181/how-do-i-know-when-ive-run-out-of-gas-programmatically?noredirect=1&lq=1, 9 February 206. [Accessed: 16 July 2016]. pages 59

[103] Django Software Corporation. Django. https://www.djangoproject.com/, 2005-2016. [Accessed: 18 February 2016]. pages 65

[104] Python Software Foundation. Python. https://www.python.org, 2001-2016. pages 65

[105] Rails. http://rubyonrails.org. [Accessed: 19 March 2016]. pages 65

[106] The Ruby community. Ruby. https://www.ruby-lang.org/en/. [Accessed: 20 May 2016]. pages 65

[107] Meteor Development Group Inc. Meteor. https://www.meteor.com, 2016. [Accessed: 02 June 2016]. pages 65

[108] Mozilla Developer Network and individual contributors. Javascript. https://developer.mozilla.org/en-US/docs/Web/JavaScript, 2005-2016. [Accessed: 02 June 2016]. pages 65

[109] HotFrameworks. Find your new favorite web framework. http://hotframeworks.com. [Accessed June 2016]. pages 65

[110] Sashko Stubailo. Is meteor.js today's ruby on rails? Quora, [Online Discussion], https://www.quora.com/Is-Meteor-js-todays-Ruby-on-Rails. [Accessed: 26 May 2016]. pages 65

[111] mongoDB. Mongodb documentation. https://docs.mongodb.com/, 2016. [Accessed: 23 June 2016]. pages 65

[112] Introducing meteor api docs. http://docs.meteor.com/#/full/, 2016. [Accessed: 02 June 2016]. pages 65, 66

[113] Percolate Studio. Atmosphere. https://atmospherejs.com/. [Accessed: 03 June 2016]. pages 66

[114] ethereum. ethereum:web3. Athmoshere, https://atmospherejs.com/ethereum/web3. [Accessed: 06 June 2016]. pages 66

[115] @mdo and @fat. Bootstrap. http://getbootstrap.com/. [Accessed: 28 June 2016]. pages 66

[116] dandv twbs and publishbot. twbs:bootstrap. Athmosphere, https://atmospherejs.com/twbs/bootstrap. [Accessed: 28 June 2016]. pages 66

[117] Greif Sacha Coleman Tom. Discover meteor. Version 1.3, https://www.discovermeteor.com/, 2016. [Accessed: 02 June 2016]. pages 66

[118] iron. iron:router. Athmosphere, https://atmospherejs.com/iron/router. [Accessed: 04 June 2016]. pages 66

[119] Slant. What are the best javascript ides and editors? http://www.slant.co/topics/1686/ javascript-ides-and-editors. [Accessed: 25 May 2016]. pages 66

[120] Ethercamp. https://live.ether.camp/. [Accessed: 28 May 2016]. pages 66, 68

[121] JetBrains s.r.o. Webstorm. https://www.jetbrains.com/webstorm/. [Accessed: 02 June 2016]. pages 66

[122] Rishi Goomar. A round-up of text editors for meteor development. Discover Meteor, https://www.discovermeteor.com/blog/text-editors-meteor-development/, 11 December 2014. [Accessed: June 2016]. pages 66

[123] Go-ethereum. Geth. Github, https://github.com/ethereum/go-ethereum/wiki/geth. [Accessed: 02 June 2016]. pages 66

[124] mKoeppelmann. What are the development tools for ethereum? Stack Exchange, [Online Discussion], http://ethereum.stackexchange.com/questions/2064/what-are-the-development-tools-for-ethereum?rq=1, 14 March 2016. [Accessed: 24 May 2016]. pages 66

[125] ConsenSys. Truffle. Github, https://github.com/ConsenSys/truffle. [Accessed:June 2016]. pages 66, 67

[126] Mocha. https://mochajs.org/. [Accessed: 26 June 2016]. pages 66

[127] Chai assertion library. http://chaijs.com/. [Accessed: 26 June 2016]. pages 66

[128] ConsenSys. Ether pudding. Github, https://github.com/ConsenSys/ether-pudding. [Accessed: 26 June 2016]. pages 66

[129] ethereumjs. Welcome to testrpc. Github, https://github.com/ethereumjs/testrpc. [Accessed: 10 June 2016]. pages 66

[130] GitHub. Atom. https://atom.io/. [Accessed: 02 July 2016]. pages 66

[131] Jeff Magee and Jeff Kramer. Labelled transition system analyser v3.0. https://www.doc.ic.ac.uk/ jnm/book/ltsa/LTSA_applet.html. [Accessed: 04 February 2016]. pages 74

[132] Peter Vessenes. Ethereum griefing wallets: Send w/throw is dangerous. http://vessenes.com/ethereum-griefing-wallets-send-w-throw-considered-harmful/, June 2016. [Accessed: 16 August 2016]. pages 75, 76

[133] Stack exchange. http://stackexchange.com/. [Accessed: 14 February 2016]. pages 81

[134] Storj Labs Inc. Storj. https://storj.io/. [Accessed: 23 August 2016]. pages 82

[135] Bitshares. Bitshares - your share in the decentralized exchange. https://bitshares.org. [Accessed: 08 June 2016]. pages 82

[136] Curitiba in English. binary-number-tunnel-1080p-hd-wallpaper. http://curitibainenglish.com.br/wp-content/uploads/2013/01/binary-number-tunnel-1080p-hd-wallpaper.jpg, January 2013. [Accessed: 27 August 2016]. pages 84