POLITECNICO DI TORINO

III Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Specialistica

BlockArt

A Blockchain dapp for collaborative artistic production

Supervisor: Laura Farinetti, Politecnico di Torino

Co-rapporteur: Gianni Corino, University of Plymouth

Co-rapporteur: Valentina Gatteschi, Politecnico di Torino

Candidate: Davide Grimaldi

Ottobre 2019

To my family

ABSTRACT

This work was developed through a mobility program between Politecnico di Torino and University of Plymouth, it proposes to develop a blockchain platform of collaborative community production for artists. The artist will be able to propose a draft that will be analyzed, discussed and implemented by the community acquiring a certain value on the platform. The first part of my thesis will have the intent to describe the main characteristics, both in general and in the artistic field, of peer communities (organization, production, governance and motivation). The second part will be an in-depth analysis of the world of the blockchain and on the Ethereum platform, then the working mechanisms of the blockchain and the Ethereum Virtual Machine (EVM) will be discussed. With the help of some platforms related to collaboration between artists, I focus on the main challenges these platforms have to face in the field of intellectual property. Finally, my platform will be illustrated by analyzing the main features and the smart contract.

INDEX

Contents

1 INTRODUCTION 1

1.1 CONTEXT 2

2 PEER PRODUCTION AND CROWDSOURCING 4

2.1 A GENERAL INTRODUCTION OF P2P 4

2.1.1 ORGANIZATIONAL MODELS 5

2.1.2 MOTIVATIONS 7

2.1.3 PEER GOVERNANCE AND PEER PROPERTY 9

2.1.4 WIKIPEDIA AND PEER PRODUCTION DESIGN 10

2.2 CROWDSOURCING AND ART 17

2.2.1 DEGREES OF PARTICIPATION 19

2.3 ALPHA ARTISTS AND ADMINISTRATORS 21

3 BLOCKCHAIN 23

3.1 BLOCKCHAIN. HOW DOES IT WORK 24

3.1.1 GENERAL DEFINITION 24

3.1.2 TRANSACTION 25

3.1.3 SECURITY CHALLENGES, HASHING AND CRYPTOGRAPHIC 26

3.1.4 BLOCK 31

3.1.5 HEADER AND MINING 32

3.1.6 COMPETITION 35

3.1.7 TRANSACTION HISTORY 37

4 SMART CONTRACT AND ETHEREUM 41

4.1 SMART CONTRACT 41

4.2 ETHEREUM 43

4.2.1 STATE ACCOUNT 43
4.2.2 MESSAGE AND TRANSACTION 44
4.2.3 WORD STATE AND EVM 46
4.2.4 ETHEREUM STATE TRANSACTION 49
4.2.5 BLOCK 51
4.2.6 GHOST CRITERION 54
4.3 DEPLOY SMART CONTRACT 56
4.3.1 SOLIDITY 56
4.4 UPGRADABLE SMART CONTRACTS 68
4.4.1 DATA SEPARATION PATTERN 69
4.4.2 PROXY PATTERN AND ZEPPELINOS 70

5 SIMILAR PLATFORMS **74**
5.1 CROWDBC 74
5.1.1 ROLES 74
5.1.2 KEY FEATURES 75
5.1.3 MAIN STEPS 76
5.1.4 CONTRACTS 77
5.2 BACKFEED 81
5.2.1 TECHNICAL ELEMENTS 82
5.2.2 SYSTEM MODEL 83

6 BLOCKART **85**
6.1 SYSTEM MODEL 86
6.1.1 ROLES 86
6.1.2 THREATS 87
6.2 BLOCKART: PEER PRODUCTION FRAMEWORK . 89
6.2.1 LAYERS 89
6.3 MAIN ALGORITHMS 91
6.3.1 STEP BY STEP 91
6.3.2 REPUTATION SYSTEM 92
6.3.3 DEPOSIT 95

6.3.4 MODERATION RULES 95
6.3.5 VOTING RULES 96
6.3.6 CONTRACTS 97
6.4 EVALUATION RESULTS AND ANALYSIS 103
6.4.1 SYSTEM DESIGN 103
6.4.2 RESULT EVALUATION 104

7 CONCLUSION **107**

A Table of smart contract costs **115**

1 INTRODUCTION

Nowadays there are many applications offering peer production services or crowdsourcing, all these applications have a decentralized view. Thus, the added value of these platforms is to get everyone involved in the discussion. In particular, when we talk about art, the concept of decentralization, although still little experimented, takes on an even stronger meaning. There are not many authors who deal with this subject, because artists are reluctant to participate in art. The few topics in the literature present generic descriptions of existing platforms. Starting from these points, the purpose of this work is to define a tool, BlockArt, able to present a completely decentralized application of support for the artistic production communities. Given that participatory art cannot be attributed to peer production or crowdsourcing, nowadays there are no platforms that offer a service similar to BlockArt. The development of blockchain technologies has allowed a greater development of decentralized technologies, this has generated a competitive advantage compared to centralized technologies. While the most important disadvantages are related to the lack of maturity of the blockchain technology. The purposes of this research are:

- makes artists more aware of the themes of participatory art;
- creates a proposed platform as a hybrid platform between peer production and crowdsourcing;
- realizes a completely decentralized application.

The most important challenge will be to be able to achieve all the goals listed through an efficient and safe application. So the goal of the research is to create a usable dapp. Starting from the hypothesis that a service like the one offered by BlockArt is absent in the dapps environment. In an attempt to demonstrate this hypothesis, the following research will begin to describe the main characteristics of peer communities (organization, production, governance, and motivation) in the artistic fields, and then there will be an in-depth analysis of the blockchain world and on the Ethereum platform . The final part will be a description of the BlockArt framework and an analysis of the tests performed.

1.1 CONTEXT

This project has been developed thanks to a mobility program between Politecnico di Torino and University of Plymouth. The name of Politecnico di Torino's program was "Mobilità per tesi su proposta del candidato". The first part of the research, reference research and idea prototyping, was developed at the School of Art, Design and Architecture of the University of Plymouth, inside i-DAT Open Research Lab. The university of Plymouth is a public university based predominantly in Plymouth, England. One of the most important feauture for the development of this research, gives from Plymouth university, was the possibility to access of the university library that is opened 24/7, and has a huge number of bookable books. i-DAT is an Open Research Lab for playful experimentation with creative technology and consists of two entangled parts: i-DAT Research and the i-DAT Collective. More specifically, the i-DAT fields are related to interdisciplinary research explores the idea of performative, embodied and social networks, in emergent technological practices. The i-DAT projects are provided by i-DAT Collective, a collaborative group of interdisciplinary artists, technologists and researchers. So, these projects are rooted in the main i-DAT research program that deals with making the "data" palpable, tangible and accessible.In this part the research was supervised by Dr. Gianni Corino Leader for MRes Digital Art and Technology at Plymouth University, who was able to direct and advise research in its preliminary phases. So, the opportunity to develop the research idea inside the Uni-

versity of Plymouth and i-DAT Collective allowed this research to form and develop across various disciplines (social, design, blockchain)

2 PEER PRODUCTION AND CROWDSOURCING

2.1 A GENERAL INTRODUCTION OF P2P

The term peer production describes a model in which people work together in a co-authoritative manner. Peer production is an organization that develops in three different fields:

- Decentralized
- Non-monetary motivation
- Difference between property and organization

Peer productions are ”large-scale, collaborative and primarily voluntary models” [1]. These productions are defined as decentralized because the objectives of production are not solved individually but by groups with an organizational hierarchy but by an open collaborative group. Motivation of these groups are heterogeneous this means that every person of the peer production has different reasons to participate than they are: **human motivation**, **innovation**, **experimentation**, **knowledge**, **organization cost** and difference between **property and organization**. This differentiation is the most common reason why people choose peer production, peer production's organization models for assigning tasks are based on combinations of participatory, meritocratic and charismatic skills while the property is managed through a contract that has no restrictive restrictions.

Free and open-source software (FOSS) or Wikipedia are the most common example of peer production, production is decentralized among different people (each person adds some features to the production becoming collaborator), the collaborators are motivated by reasons not monetary and production is coordinated / managed by collaboration contracts; in some cases the contributions are covered by copyleft licenses.

Peer production in the last decade developed into two different flavours: Common-based peer production (CBPP), like FOSS or Wikipedia; Firm-hosted peer production, such as Yelp or TripAdvisor, which derived from CBPP but in that the property and organization are governed by the firm-hosted. For example, Yelp provides a service through which users provide reviews on local activities, but contributors are obligated to accept the terms of use before. Then yelp maintains property and usage on own database. The other forms close to peer production are crowdsourcing (that will be the topic of the next section), online labour markets and Open collaborative innovation. Crowdsourcing is distinct from peer production because the assignments are pre-determinate, in this case, motivation could be both monetary and non-monetary. Online labour markets have a different model of decentralization, more individual than collaborative because they are a market in which users demand a solution for their problems and the best solution will be the only remuneration, monetary motivation. Open collaborative innovations are practices, in production and innovations, among firms to make technological improvements.

In terms of organization, peer production marks a new social model where everyone can access to production with "symmetrical access privileges (with or without use rules) to the resource without transaction"[2]. In peer production every people has not any limitation to access to a given a resource or project, every collaborator could suggest any improvements and could self-assignment to a part of the project.

2.1.1 ORGANIZATIONAL MODELS

According to the organizational models (Figure 1) , we can divide each form of peer production into three different functions in: **Resource space**, **Capital requirements** and **Project space**. Resource space

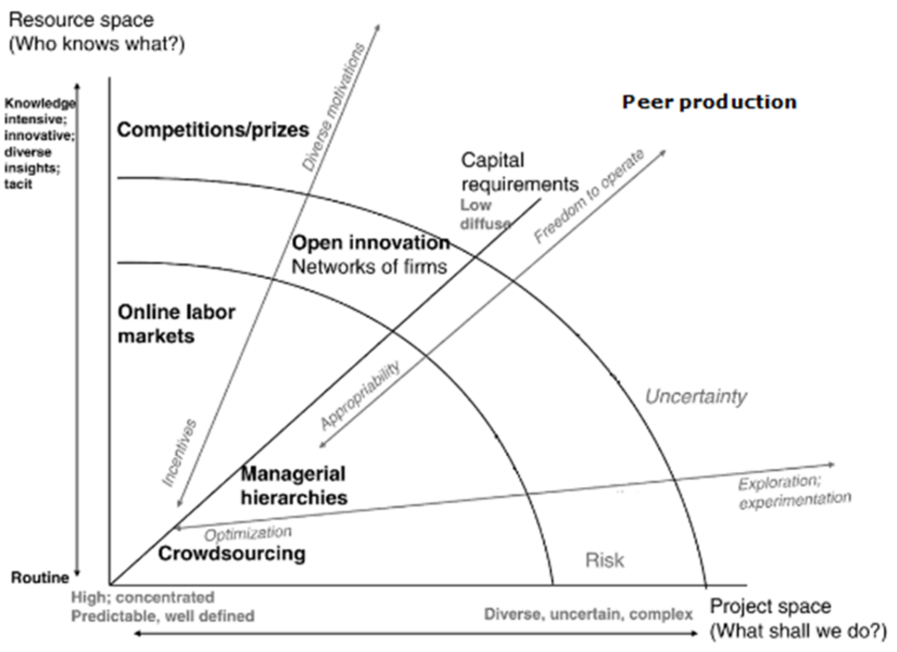

Figure 1: organization model

represents who are contributors which have the potential to contribute to production. This function is based on knowledge of people which contributes to projects. In the upper part of this space, we find organizational models that concern those forms of production that are more decentralized, where it is not well known who will contribute to production (complexity and uncertainty) as peer or competitions/prizes productions. While in the lower part of this space there are all those less decentralized forms of production where one knows better what kind of person will contribute to the production (routine) as in crowdsourcing. Project Space is the space that indicates how much the final object is defined. Where organizational model, in project space, is more decentralized there are more risks to the production (object is not well defined), then hierarchical organizations do not incentive to invest because the risks are more than of the monetary return. On the other hand, organizational models like crowdsourcing are models with low risks because the objects of production are well-defined first. The last one is the capital requirement is the function that indicates the ownership of the production object. Peer production can emerge if the physical capital requirements of a project are very low, i.e. the capital is distributed among the different contributors. While in Managerial hierarchies the capital requirement is concentrate between few actors' managerial hierarchies. After that, we can distinguish two extremities in Figure 1 on the one hand crowdsourcing and on the other peer production. Crowdsourcing in terms of motivations it needs of incentives while in peer production there are diverse motivations (these motivations will be discus following chapter). While in crowdsourcing managerial hierarchies are the owners of the production (appropriability) in common production the contributor is free to operate. The last difference between peer production and crowdsourcing is: peer production has better exploration and experimentation for the own structure, in the crowdsourcing optimization is well-managed because is a certain structure (the roles are defined first).

2.1.2 MOTIVATIONS

According to Ghosh (1998) and the statements by Lerner and Tirole (2002) contributors in peer production are moved by different motivations: "(1) the use-value of the project to the contributing, (2) hedonic pleasure from of participation, (3) enhanced employment prospects from reputation gains or human capital accumulated, and (4) social status gains within the community of peers"[3] [4]. But in general, the motivations that drive individuals to participate in peer productions are often heterogeneous therefore designing a well-functioning system becomes more complex. According to sociologists the relation between social-motivation and old forms of standard economic incentives (like salary) are not separable, then the old economic system reward (punishment) is not compatible with the peer production system. According to Alexy and Leitner:

"positive (negative) monetary rewards (punishments) can (1) drive the total sum of motivations for any given individual in the opposite direction, and (2) lead to substitution in the population of agents from individuals driven by prosocial motivations to individuals driven by monetary motivations."[5]

On the other hand, it is more complicated to define if integrating rewards to peer production there are advantages to the production, because web sites, such as Weblogs Inc., which tried to use social recognition for production have failed. However, it has been noted that the tendency to cooperate is a determining factor in the design of the best organizational structure of peer production. According to Benkler, to design a best organizational structure for a peer production there are six different "levels of prosocial contributions"[6]:

- Communication: a system with unstructured exchanges is better than a system with a canned message. A peer production system is a system based on a peer structure, i.e. open and continuous communications

- Normative framing and norm-setting: one of the most important motivation is like peer production is regulated. It is important that the rules are clear and documented, and it is important not to change these rules to help users from developing relationships with each other. "Moreover, in the case of FOSS, normative framing has been described as permitting a mixture of monetary and non-monetary rewards, as long as monetary rewards are separated from the governance of the project" [5].

- Reciprocity, reputation, transparency: called also intrinsic incentive, studies shown that different levels of reciprocity in peer productions can improve levels of contribution. Moreover, the mechanism that gives transparency to identity can be used to generate more relationship cooperation between contributors

- Fairness: it is important that within of peer production there is not any suspect of hierarchical structure and that there is continuous negotiation between participants.

- Empathy and solidarity: "Cooperative systems perform better when they emphasize other-regarding motivational vectors" [6]. Cooperation works better if the system gives the possibility to shared own point of view of the project.

2.1.3 PEER GOVERNANCE AND PEER PROPERTY

Peer governance is the way in which peer production is managed. Peer governance is not the power in the hands of a dominant figure or group within peer projects but following the principles of peer production is the possibility of decentralized power among all participants in the project. According to Vasilis Kostakis: "Peer governance is a new model of governance and bottom-up mode of participative decision-making that is being experimented in peer projects, such as Wikipedia and FLOSS" [7] [8] "Thus peer governance is the way that peer production, the process in which common value is produced, is managed."[9]. In participative decision-making is essential freedom to operate/contribute. The most advantage of peer governance is freedom to operate/contribute because the lack of need to seek permission before the contribution is the essential motivation that makes peer governance able to avoid the loss of information and diversity that are common in hierarchical systems. We must not think peer governance as an anarchist system, in fact, against low-quality material and to safeguard the project's integrity, everyone is free to make a contribution but the community can refuse it. Wikipedia is the most common example based on this system. In the peer projects, there are "benevolent dictators" or "Alpha Artist", as we will see later in the crowdsourced art, which are the reference figures of that project. According to Axel Bruns, the benevolent dictators are: "as ones of several heterarchical leaders of the community, who have risen to their positions through consistent constructive contribution and stand and fall with the quality of their further performance" [10]. Benevolent dictator figures must be coordinator not controller in a peer production system, he, through a process of acquisition of trust by the other contributors, pass to the centre of the peer network acquiring a role of support and help to peer contributors. On the other hand, there is the problem of the authorship of the project. In the beginning, the peer production project is guaranteed by the property default regime. This right of property is exercised by the benevolent dictator. Then the property is given to license to the community in two different ways:

- FOSS licenses
- Creative Commons

both of two licenses are non-profit. While the first is used especially for free software, the second one is used in different fields. Within creative commons, the most important division is between those that the re-use of contents for any propose and those that seek to limit this. Sometimes peer production's projects are under "copyleft" license, it is a special license that which allows the re-use of the materials only on the same terms as they themselves received access. The problem of authorship will be dealt with in detail in the crowdsourcing art part because the authorship represents a fundamental part in crowdsourcing productions.

2.1.4 WIKIPEDIA AND PEER PRODUCTION DESIGN

As mentioned earlier, the main important elements to design a better solution to peer production platform are transparency, norms, organization, rough consensus and non-determinative voting. Most of these elements are rules by the Engineering Task Force (IETF) and World Wide Web Consortium (W3C). These two collaborative institutions have the purpose to rules the Internet protocols (e.g., TCP/IP) and Web formats (e.g., HTML). The persons who participate in these institutions work in mailing lists, teleconferences and face to face meeting. Now we will see how Wikipedia is designed. Wikipedia, the most famous peer production project, is called like a “multilingual, web-based, free encyclopaedia based on a model of openly editable and viewable content”. Everyone is free to edit contributions through wiki system even without creating an account. Through a wiki account, you can access some additional features such as ”watchlist”, which allows you to follow content by receiving notifications when this is changed. Under the point of the transparency, the Wiki system for each content of the site creates a register visible to all with all the changes made, a contributor can then decide to restore the previous content of the last change. The Watchlist and Revision pages are the most important web page of Wikipedia because through these the contributors can understand who has the same interests and generate discussions among them on that topic. Then discussion page is fundamental in the production process of Wikipedia's system, these pages are used to communicate, to make help request and to coordinate work. Most of the contributions on Wikipedia are made by a few people, in fact, according to Yann Algan, Yochai Benkler, Mayo Fuster Morell, Jérôme Hergueux in 2013, ”only 4 per cent of the participant from 100,000 to 1,000,000 contributions, while the rest of the users contributed only 10,000 entries”[11]. It is important to note that the norms within Wikipedia were born with time through the progressive construction of consensus among contributors. Some of these norms are a neutral point of view (every point of view must be considered) and take good faith (in general, users do not have malicious purposes). Wikipedia's organization is linked with the role of benevolent dictators that is held by “Wikipedia administrators” who have special powers to block malicious editor and protecting vandalized pages according to community rules. Wikipedia administrators are users who have contributed to many voices and have gained a lot of consent from the community. But in peer production, the most difficult challenge is a rough consensus. In Wikipedia the **challenge of consensus** born with the disambiguation. According to Wikipedia definition disambiguation is:

"Disambiguation in Wikipedia is the process of resolving conflicts in article titles that occur when a single title could be associated with more than one article. In other words, disambiguations are paths leading to different articles which could, in principle, have the same title"[12].

In these situations, the norms indicate that using verifiable sources and assuming good faith the community must reach an agreement, but what happens if there is no agreement or consensus? In Wikipedia general definitions for consensus speak of "general agreement," "without active opposition to the proposed course of action". The most difficult challenge for consensus is not to reach a decision but rather to reach the best possible solution. Always citing the Wikipedia definition of consensus: "Achieving consensus requires serious treatment of every group member's considered opinion.... In the ideal case, those who wish to take up some action want to hear those who oppose it, because they count on the fact that the ensuing debate will improve the consensus", then it is important remember that Wikipedia is based on good reasons system, not a non-determinative voting system that is often used to start a discussion, not the end it. So, the problem is not how to reach consensus but who participates in the discussion, in this context Wikipedia reveals some shortcomings. In Wikipedia, the fact of not having a mailing list and its openness make it difficult to define who they participate in the discussion. This situation makes easy trolling phenomenon that transforms argumentation in endless discussions and therefore the group of discussion collapse in frustration. To resolve this issue the IETF and W3C have deliberated some rules:

- the discussions must be built to focus on the topic, to avoid endless discussions

- the working group chair is essential for summarizing and recording discussion, to avoid discussions on some topics. To reject discussion a Working Group Chair can use different criteria: old (already resolved), minor (topic minor for working group), timing (not yet opened for discussion), or scope (topic outside of working group's discussion).

On the other hand, there is the problem of silence. In a discussion group, silence after a request is not a good thing. It is difficult to interpret **silence**, because as Wikipedia states it could mean both assent, dissent or a lack of interest. At the IETF and W3C, there is a process called "last call" where the participants before starting the practical test must give their point of view on the discussion.

In Wikipedia, the role of benevolent dictators is taken by the facilitator. This figure has many tasks, the most important is once the discussion is over, they summarize the decisions taken, and if there are no objections, indicate this as a new role for the community. In Wikipedia facilitator are known as "clerks" because as Sheeran states it has roles clerk responsibilities like scheduling discussion, starting question of discussion, roles and helping the discussion (i.e., they indicate what is important and rule out obstructionists or spammers), and indicating the final point of discussion (i.e., when consensus was reached in the discussion). In some discussions, it is very difficult to get the consensus. As happened in the discussion on the television series Buffy the Vampire Slayer, there was a long dispute if there is disambiguation between the title of an episode and the title of all TV series, 26 people have chosen to support only the disambiguation if necessary and seven have chosen to oppose. The problem was that there was no consensus.

Consensus system generated the most important problem in Wikipedia. According to Christian Stegbauer, the Wikipedia community has turned into a structural cultural production as a company, where there is no peer community but a hierarchical structure. The first ideology of Wikipedia was everyone can participate in the creation of contents, then ‘global knowledge’. But the privileges given to the facilitator to mediate the discussions generated a positional Wikipedia system that created a ”lack of social integration or negotiating options” [13] between facilitator and new users. Always according to Stegbaue the control, therefore, is managed by an elitist group that makes all the most important decisions.

Another way to make a decision in a peer community is voting. While in a small group the best way is discussion and consensus, when there is a large group of opinions in a community voting often is better than discussion. In general, Wikipedians consider ”Voting Is Evil” [14] but it has been shown that the probability of contrasts between two people increases geometrically with the increase in group size. As seen previously consensus generated few persons make decisions for the whole community, making it difficult to express their opinion. So voting result more neutral than consensus, then why do Wikipedians consider voting as evil? According to Wikipedians, the vote discourages consent because it avoids the discussion and it generates groupthink. Many times, the vote simplifies the complex question (i.e. yes / no vote) generate false dichotomies and confusion. Then voting in Wikipedia is used in few cases, for example for deletion of voices, in fact, according to Wikipedia: ”Wikipedia is not a democracy”[15].

COLLABORATORIUM Therefore, the design of peer production is very complicated. According to Luca Iandoli, Mark Klein, and Giuseppe Zollo a way to manage the complexity in peer production:

”is to move from the issue of designing a platform to the more general problem of designing the community” [16].

In their study “Can We Exploit Collective Intelligence for Collaborative Deliberation? The Case of the Climate Change Collaboratorium”, they developed a collaborative platform named Collaboratorium. Now we will see the main feature of **Collaboratorium** and the most important difference compared to Wikipedia.

Like Wikipedia's community, in Collaboratorium there are different topics with one or more goals coherent whit topic, a large group of participants, rules and facilitators who manage the community, but in Collaboratorium the innovate thing is deliberation. While in Wikipedia and forums and blog users adapt to the consolidated (not interactive, static) decision-making process, Collaboratorium deliberation is: "**a set of processes through which members explore the solution space and converge to a decision**". Collaboratorium uses proxy approach, i.e., intermediation/delegation for the deliberation process because according to the study without control there are not adequately knowledgeable and high-quality arguments.

The deliberation process, that in Wikipedia is named consensus process, is a set of two steps: convergence and exploration. Deliberation process is a system that links the quality of the topic to the author's reputation. Both the user supports the author's argumentation and does not support the author's argumentation, users must score the idea and to find reliable sources or propose new argumentation to demonstrate the reasons for their choice. It is also possible that the user can read arguments contrary to the author's idea and modify his opinion. This system allows "knowledge structure and scoring system work together to encourage a collectively-vetted, logic- and evidence-based approach to decision-making " [16]. Now we will analyse, according to the peer production motivations listed above, the main differences between Wikipedia and Collaboratorium:

- While in Wikipedia users are free to express their opinion, in Collaboratium there is a formal structure which **every collaborator** must use to express his opinion. This framework is essential to ensure the whole community participate in the discussion by providing valid arguments. This process makes through process use of "claims, rebuttals, pros and cons, explicit opinions, facts and figures" [16].

- Collaboratium incentives the "mass-conversation". To achieve this goal, Collaboratium's users, before to participate in the discussion, **forced to read other contributions**.

- Studies shown that participation level decrease with an increase of argumentation rules. Then Collaboratium will have fewer users than Wikipedia. But in general, when there are fewer rules, the **quality of discussion** is less than when there is more, i.e. limit of redundant information, off-topic and digression.

- According to the authors Collaboratium will produce **less polarization** than Wikipedia, the deliberation process is more restrictive than consensus, but reduces the possibility that decisions are made by a small decision-making group. People are more interests to participate when they are more involved in the discussion.

Collaboratium platform has three figures: moderators (filter, reject off-topics and mediate contents of posts), authors and readers/voters. Moderator's rules "involved classifying and sometimes editing posts, offering suggestions to authors, aggregating similar arguments, and occasionally re-organizing the overall argument map so that related topics are grouped into the same branch" [16]. Authors initially post or modify a topic; this topic will be pending status. After that the moderator certificate the validate of the post, all users can rate the post with an anonymous identity. Then users can deliver a suggestion to the authors to improve argument through a discussion forum. The topic contains different elements pro/con to the argumentations, suggestion ideas and documents to validate the content of the post. Nobody can re-post an argument except moderators. Moreover, mediators have a key role in Collaboratium because do comments and suggestions to the posts, through the management of the map, and they help users to link their knowledge with the map of the topic. According to the study, at least a percentage between 5% and 10% must be moderators.

SLASHDOT The figure of the moderator in relation to Slashdot is analyzed by Audun Jøsang, Slashdot[17] is a forum where users can post any type of article. When an article has been published everyone can comment on this article. There are two types of users: Logged In User and Anonymous User. Mediators are chosen among Logged In Users. They will are a mediator for a specif time frame, then later Slashdot will select another group of mediators. When a user is selected as a mediator she/he has a given number of moderation points to spend to evaluate the comment. A comment can be evaluated in a range between [-1, 5], each evaluation can raise or lower the evaluation of the comment by one point. So, each Logged In Users has a reputation linked with his/her sum of comments ratings, the reputation is called Karma and it can be: Terrible, Bad, Neutral, Positive, Good and Excellent. Users with high Karma start with a rating of comments of +2, while users with low Karma start at -1. The Karma system is used in Slashdot to give the opportunity for users to set thresholds for filtering of the levels of comments. Moreover, Slashdot has a second level of moderation, called meta moderation, to avoid unfair moderations problem. Only longstanding users can become Metamoderator, in this case, a user asks to become meta-moderator and then meta-moderator can evaluate 10 different comment evaluation of moderators. Metamoderator can express three different judgments: fair, unfair, or neither.

2.2 CROWDSOURCING AND ART

Thanks to digital innovations and interconnections between people through the web, concepts such as creativity, authorship and individual expression have changed in many fields. In the beginning, only companies understood the enormous potential of crowdsourcing and took advantage of Internet communities to get profits and develop their projects. Crowdsourcing and the possibility to reach a large number of individuals, through the web, has been the reason for the success of companies like iStockphoto and YouTube. was coined by Jeff Howe in an article in the Wired magazine of 2006:

"crowdsourcing represents the act of a company or institution that assumes a function once performed by employees and outsources it to an indefinite (and generally broad) network of people in the form of an open call." [18]

Howe's definition is a practical definition that identifies crowdsourcing as a platform where companies or institution delegate indefinite network of people to realize something, with or without financial incentives. The strength of crowdsourcing lies in the potential of those who contribute (collective). There are many points in common with peer production, including motivation, governance and design, but it depends on how the platform is structured. In the art field, collective intelligence can achieve efficient results only if communities use collective intelligence to improve artistic process[19]. As in many Roy Ascott's works[20][21][22] where the audience participates to the artistic process, today thanks to the Web 2.0 the possibility is extended to a global community. Another example is the "exquisite corpse" a technique invented by the surrealists during their shows, the technique was a game in which several participants had to draw something on a sheet and passed the sheet to the next player adding a rule to the game, then crowdsourcing is a foreseeable consequence of the past artistic trends. Compared to the past, the revolutionary reach of crowdsourcing is the audience does not work to take part in the artistic production but to become a part of artistic production. The practice of online crowdsourcing art is used to develop artwork in different fields, such as visual art, music and creative writing.

According with critics: "the art object is no longer necessarily the primary focus of the encounter with art on the web". Participatory artistic practise in the last few decades have changed boundaries between artist and their audience, these changes reflect new social practices late capitalistic. Then, artists practice participatory art because: “activation (by stimulating empowerment as well as individual and collective agency), authorship (or the cessation of authorship to make the artistic process more democratic and egalitarian), and community (via the restoration of lost or weakened social bonds)” [18].

On the other hand, many of these theories do not consider the medium by which crowdsourcing art is developed. According to Literat a fundamental aspect often overlooked is that art through crowdsourcing is done online, so the whole artistic process carried out in the network. This leads to generating a ”lack of social gatherings and face to face communication”[18], it is, therefore, important to know how to better manage the artistic process in the crowdsourcing community.

2.2.1 DEGREES OF PARTICIPATION

Crowdsourcing community many time is associated with commercial crowdsourcing, but crowdsourcing community in the artistic process is something different. Problems of artists who work in the related art have overvalued this practice, with a revolutionary intent but there are different degrees of participation within crowdsourcing.[23] According to Literat there are three different degrees of participation that are divided in turn into a subcategory.

The first one is receptive participation, this degree of participation is referred to a traditional artistic mode like painting, opera, etc. The audience receives the finished artistic product but through active interpretation and creative consumption, the audience becomes participator of the artistic process. In this case, there is not active participation in the artistic process.

Executory participation in the second degree of participation, in this type of participation, there is a default artistic structure where participator can operate on the artistic process only in specific terms and conditions. In this context, participator does not have the possibility to improve or modify the initial conditions. There are three different levels:

- Tokenistic projects are organized in micro-level and often they have non-logical structure. According to most critics, these projects just improve the social reputation of the artist without real social contribution, for example during Aaron Koblin's TED conference [24][25] explained his project 'The Sheep Market'. The project consisted to draw a sheep on Amazon's Mechanical Turk every person that drawn a sheep received a 0.02$ like a contribute, just one person used drawing the interface of the project to ask to Koblin "Why? Why are you doing this???". Through this project, Koblin wanted to demonstrate how people could be used to improve the social reputation of the artist.

- Engaged is more transparent than the tokenistic projects but it has a project structure defined a priori. In this case, the participator know what the final goal of the project will be, but they cannot intervene to improve the project

- In Creative participation, the project is opened to a discussion between the participators and the parameters to participate are less strict than other forms of Crowdsourcing participation. In this case, there is an alpha artist who owns the work of artwork and acts as a curator of his own project by regulating the contributions of the participants.

The last degree of participation is structural, there two types of structural participation codesign e coauthorship. In other words, in these cases, contributors participate to realize the idea of the artwork. Codesign projects are a shared platform within which every user can freely suggest improvements and/or changes to existing artwork or add a new one. For our research, the coauthorship is the degree of participation most important but it is the most complicated because there are not any cases studies. The problem is to give a definition of an author in coauthorship's cases, this will be discussed in the next chapter. For example, in the case of “The Sheep Market” Koblin's sheep sold in groups of 10,000 for $20 each. After that, someone opened a discussion on Mechanical Turk forum titled “They're selling our sheep!!!” where discussed ethical, legal, and aesthetic aspects of crowdsourced art, most of these discussions were of the possibility to reacquired rights on sheep.

2.3 ALPHA ARTISTS AND ADMINISTRATORS

So, the artistic process is an individual expression, artists express their sensibility and their owns ideas through artwork. In this context, crowdsourcing in the art field is a new way to a new way to intend art and the artistic process. Crowdsourcing redefines the role of the artist that becomes the initiator of the artistic project. Howard S. Becker defines that initiator as “alpha artist”[26], according to the types of crowdsourcing project, listed above, the role of the alpha artist is different. Essentially, alpha artists succeed in conceiving the artistic project, but in some cases, they can also participate in the artistic process of helping users in their choices. This is a fundamental role in the exegesis of the community. The role of the alpha artist can be comparable to Wikipedia's mediator or benevolent dictators in peer production or moderator in collaboratium. From receptive to structural participation, degrees of collaboration between artist and crowd is much different, however, for the realization of the artwork, they depend on each other. So, there is an indissoluble interdependence between two roles that, as for all types of collaborative platforms, develops through the discussion.

The figure of the artist, even if it remains central in the artistic process, is re-discussed and it assumes characteristics close to that of a curator. According to Literat, a curator is who “judges, selects, and arranges the participant-submitted contributions into a final product” [18]. On the other hand, as indicated above, in Collaboratium project the mediator is who mediates between the content of production and users. So, the two figures have many points in common, curator preserves the artistic quality of artwork while mediator preserves argumentation quality. The figure of the alpha artist is not new in the art world, for example, the conductor in the act of execution acts as a glue between the various parts of the orchestra.

On the other hand, it is important to note that, unlike the mediator, the role of the alpha artist is different, i.e. while the mediator is a figure that preserves the impartiality of the discussion the alpha artist has a greater involvement in the discussion for its role as an initiator. One of the most important challenges for collaborative art platform is that: it must guarantee the greatest possible impartiality to the community through the role of the alpha artist.

An example of a possible solution is described by Audun Jøsang in his work ”A Survey of Trust and Reputation Systems for Online Service Provision[17], where Jøsang states that there are guidelines to be followed to maintain impartiality in judgment within a community:

1. Have a scoring system and distinguish the users just entered in the community from those of long standing.

2. Recognize users with many poor performances.

3. Recognize performance in the latest.

4. Hard to manipulate from the peers.

5. Individual assessments must not affect the total average.

3 BLOCKCHAIN

Software architecture the discipline that studies the structure of high-level software divides the software systems into two different categories:

- **Centralized architecture**: there is a central node and every component of the network is connected only with it, reception and sending of messages on the network is managed by the central node. Each component must wait the central node to be able to speak on the network.
- **Distributed architecture**: every nodes are connected with one another without central control. All nodes are connected together at least indirectly

The most important advantages of distributed architecture are linked to the possibility of reducing costs and amount of computing by distributing the system on all the nodes of the network. Another important incentive is the possibility that the network will continue to operate even when a node stops. On the other hand, distributed architecture has some disadvantages related to the fact that the nodes must manage the coordination themselves, other disadvantages including higher complexity of the program because the program must manage the coordination, communication, utilizing and security of the network; dependencies on networks of nodes

A particular type of distributed software system is p2p (peer-to-peer). This system is formed by a series of nodes, where every node performs a computational operation for himself (storage, distributed, and processing of information) and at the same time the operation is available to other nodes in the network (for example p2p file sharing system). According to the definition of Daniel Drescher, p2p system works "transforming users computers into nodes that make up the entire distributed system" [27], in turn, all nodes work to achieve and maintain system integrity.

Two of most important challenge for p2p platforms are **trust** and **integrity**. A system is integrated if it satisfies the properties of safety, consistence, completeness and correctness. While a system is reliable if users contribute and will contribute, i.e. on an ongoing basis, the idea of system integrity is reinforced. So, when the trust of the system is failed there is a lack of integrity, then integrity and trust are interdependent. To maintain the trust and integrity of the p2p system is important to know the number of peers (sometimes peers that connect with the network fail or produce the wrong result) and their reliability. The reliability of peers is the challenge most important for p2p system, dishonest and malicious peers make the system unreliable and then other peers "turn away and stop contribution computational resource to the system" [27].

3.1 BLOCKCHAIN. HOW DOES IT WORK

3.1.1 GENERAL DEFINITION

According with its creator Satoshi Nakamoto, blockchain is:

"a purely distributed p2p system of ledgers that utilizes a software unit that consist of an algorithm, which negotiates the informational content of ordered and connected blocks of data together with cryptographic and security technologies in order to achieve and maintain its integrity" [28].

So, blockchain starts from the idea to solve the problem of reliability of unknown peers, achieving and maintaining in a purely distributed p2p system a high level of trust. To implement this in the blockchain has been introduced a process (transaction) that verifies, through some witnesses, your **identity** and **integrity** of the system, the process will be more reliable if there will be more witnesses. In the end, the process stores transactions in a map, calling it ledger or register. **Ledger** is stored in the nodes of the p2p system, it has two most important function:

preserves any historical transaction data of blockchain and **documents** each transfer of ownership. If the ledger is open to anyone is easier providing proof of ownership, like a public testimony in a court. But in some cases, it is important to preserve the **privacy** of transferring ownership, then in these cases just who have trustful identity can write in the ledger. The most important issue related to the ledger is "**double spending**"

3.1.2 TRANSACTION

Blockchain is suitable for different fields but, to understand how it works internally, we will deal only with ownership cases. The use of the blockchain for ownership allows the exchange of goods such as a virtual currency. As mentioned previously, the process behind blockchain is called transaction, this process can be compared to bank money transfer. In the ledger, every transaction is recorded in a **transaction data**. Data in transaction data are like a bank account statement that contains a summary of all the operations performed and the total amount of the balance. Transactions data are used to make a transaction and they provide different information:

- peer's identity who starts the transfer of ownership;
- peer's identity who receive the transfer of ownership;
- the goods value of transaction;
- when transaction is done;
- transaction fees (the amount of which can be chosen by the person sending the transaction and it are paid to the block's generator node or miner);
- the proof that owner agree with transaction;

At the end of the transaction, the transaction is added to the ledger and it becomes part of the transaction history. Keeping the transaction history is useful to have an owner/ownership map and thus avoid **double spending** problems. It is also important to note that in transaction data there is an information “when transaction is done”, it is important because we have to keep track of the **transaction order** (chain) to know “how much is total the amount of peer”, and therefore avoiding that a peer spends more than its amount. To achieve and maintain trust in a blockchain, transaction history must have different levels of integrity, it depends on the final goal of the blockchain project. **Form** and **content** of transactions must provide incorrect format and also, according to Drescher, is appropriate if transactions: ”ensuring that an account does not hand off more than it currently owns, only the owner of account can execute a transaction, preventing double-spending, limiting the amount of items that can be transferred in a single transaction, limiting the number of transactions per user, limiting the total amount of items spent in a given time period, enforcing that an account keeps an item for a minimum time period before it can be transferred further” [27]. This way to manage data structure defines the validation rules for a transaction that will be treated later.

3.1.3 SECURITY CHALLENGES, HASHING AND CRYPTOGRAPHIC

The way in which a blockchain system maintains a high level of trust is represented by encrypting transaction data and protecting users account. The process of encrypting transaction data is called **cryptographic hash value** [29]. A hash function is a non-invertible function that maps an arbitrary length string to a string of fixed length, it accepts one input data at a time. The result of the hash function is a hexadecimal number. The most famous group of hash functions are the cryptographic hash functions, these functions have different properties: function is executed quickly to avoid errors; function returns the same result, for the same input, at different times; every input value has different output result; it is impossible to calculate inverse function. The most difficult challenge to calculate hash value is which hash functions accept just one input data at a time. To solve the challenge of different input in one time, now will list five different patterns:

- independent hashing processes one input at a time;
- repeated hashing works like independent, but after that it processes the result value of the previous process calculating a new hash value
- combined hashing aggregates every input in just one input and process it (with a letter space the same result of repeated);
- sequential hashing divides the initial data into a series of inputs, then processes the previous input, aggregates the result with next one and processing it;

These patterns are used to solve some secure issue: result hash is used to compare inputs value among each other (comparing the hash is less expensive than the inputs); hash is processed a first time and is compared the next time to see if there are any differences, so if the input data has not altered; calculated hash is the same at different times for a references[1],i.e. taken a given reference if the hash of that input changes over time, it means that the data in that reference has been altered. These are some general uses in computer science of hash functions for security, but in the specific blockchain the hash functions are used for: **chain**, **Merkle** tree and **puzzle**. Let's see what they are:

[1]references are data types that refer to an object elsewhere in memory and are used to construct a wide variety of data structures, such as linked lists. Generally, a reference is a value that enables a program to directly access the particular data item.

- a chain is used to ensure the integrity of all blocks; chain is a list that links each block with the next through its hash reference value, i.e. taken n values: data 1, data 2, ..., data n and n references: R1, R2, ..., Rn-1. We connect data 1 with a block formed by data2 and r1 (reference of the previous block), then we continue connecting all data with the reference of the previous one. The last data, i.e. data n, has no reference and is called "head of the chain", data n is the data through which the chain can be accessed. Hence, if the chain has broken somewhere, it is proof that some data have changed after was created;

- Merkle tree is a tree structure that takes its name from its inventor Merkle. This structure can be seen as an overturned tree. At the bottom, we have the transactions (for example two for each branch). At the next level, we have the references to the transactions are grouped together. While at the top level we have a hash reference of the pair grouped together to another branch. This procedure is repeated until a single hash is reached;

- hash puzzle is a mechanism that has been created "to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers" [30]. Hash puzzle takes advantage from the fact that hash functions are not invertible functions, so it asks the user who wants to solve the puzzle to look for a certain hash value (restriction) between a fixed input value (data) plus a list of a variable number (once). When the output hash value coincides with the solution of the puzzle, the user has solved the challenge. Solving the puzzle requires a computational cost, since the hash function is non-invertible function, so the only way to solve it is by trial and error.

The process to protect users account is called **asymmetric cryptography**[31]. The general idea of asymmetric cryptography is to ensure the user account identity and access of his or her property, in a distributed architecture. The goal of identity protection is achieved through public-private-key encryption, everyone can transfer ownership through public key while just who possesses the private key is can access ownership. For example, as shown in Figure 2 we can imagine the public-private-key encryption system as the postal system, anyone can send a letter if he/she knows destination address (public key) but only who has the key of that mailbox (private key) can access to the letter. Now, we start to describe the main important steps of cryptography. Cryptography is composed by encryption (lock the identity) and decryption (unlock the identity), often the encrypted data is called cypher text. The first step is to produce, through a cryptographic key, the cypher text starting from data which sender wants to send, then sender sends data to the recipient and finally, recipient encrypts the message with the key.

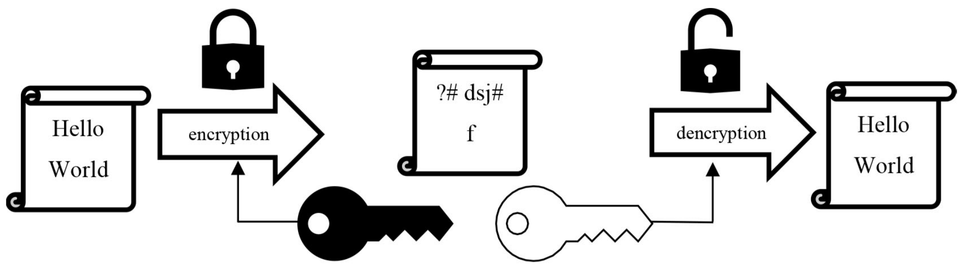

Figure 2: Cryptography

There are two types of Cryptography symmetric and asymmetric. Symmetric cryptography uses the same key for encryption and decryption. But the main used method in cryptography system is that asymmetric, in this method there are a pair of complementary keys for every cypher text. As shown in the Figure 2 the first key is used to encrypt the message and to create cypher text, while the second key is used by the receiver to decrypt the cypher text and to read the message. Based on their roles, in the asymmetric method the keys are called respectively **public** and **private** key. Both encrypt data and decrypt cypher text can be private or public. While public key allows access to anyone and it is used for trustworthiness, private key (which is owned only by the owner) does not allow access to anyone except the owner and is used to keep safe and private data. As previously mentioned, in the blockchain technology there are **public-private-key** and **private-public-key** encryption these systems are based on the asymmetric method in public to private flows and private to public flow. In the blockchain system asymmetric cryptography is used for:

- **Identifying accounts** and **transfer ownership** use public to private key. Every node in the blockchain system has a public key. So, the sender encrypts her/his transfer with the public key is used to identify the receiver user account. A public key is a number which everyone can use to identify accounts to which the transfer of ownership should be sent.

- **Authorizing transactions** uses private to the public key of asymmetric cryptography. When an owner wants to transfer her/him property a blockchain platform must ensure this transfer, to achieve this goal a digital signature is created. So, a digital signature is the private key that secures the integrity of data by the receiver (for example goods value of the transaction); by applying owner's private key on the hash value and insert this in transaction data. The most important property of asymmetric private-to-public key is that digital signature (or cypher text) can be encrypted and decrypted with one and only one pair of keys, then this method is used as proof of identity of the owner.

These two methods work together in the transaction to verify transaction data. The transaction encrypt/decrypt process it is a three-step process: identifying accounts, authorizing transactions and transfer ownership. The sender of the transaction identifies receiver identity with receiver public key (Identifying), at the same time, he/she applies the digital signature to encrypt transaction data (Authorizing). While the receiver verifies the identity of the sender using the public key that corresponds to the digital signature (Authorizing), and he/she uses his private key to receive ownership (Identifying). Thus, if the two processes generate is completed, the transaction is verified while in the opposite case fraud is identified. Once the sender has sent his/her ownership trough the transaction, the transaction will send in broadcast on the network and it will take place only when a certain number of nodes will have verified and confirmed it, based on the specific blockchain network rules. The verification process is done by nodes by verifying the semantic correctness of the transaction data while the confirmation process is the addition of the transaction to the blockchain-data-structure. We will see later how transactions become part of blockchain-data-structure.

3.1.4 BLOCK

Now we show the main important feature on the data blocks is used in a blockchain platform

Starting from the metaphor used by Drescher, we can compare **blockchain-data-structure** as a book structure. The first challenge in blockchain-data-structure is how ordering the block each other? So, taking how example the book, the page number of a book indicates the sequence of pages, sometimes it is possible that page numbering follows an order different from the natural decimal system, for this reason, each block contains a reference to the block that precedes it. The next step is to store and identify a reference for the content of the block (text). So, we store the text of the page in a different place than the page of the book. For example, we can take that page and transfers it “in box or on a shelf”. Text can be compared to the transaction data in blockchain-data-structure. In the real blockchain application, there are not transactions data in a block, they are stored in an external database and are represented by hash numbers in the block. The last step is to create a unique reference number to link blocks together. Through cryptographic hash values are calculated the reference value of the blocks, which is a combination between reference hash number of transaction and the reference hash number of precedent block (combined hashing), it is called header.

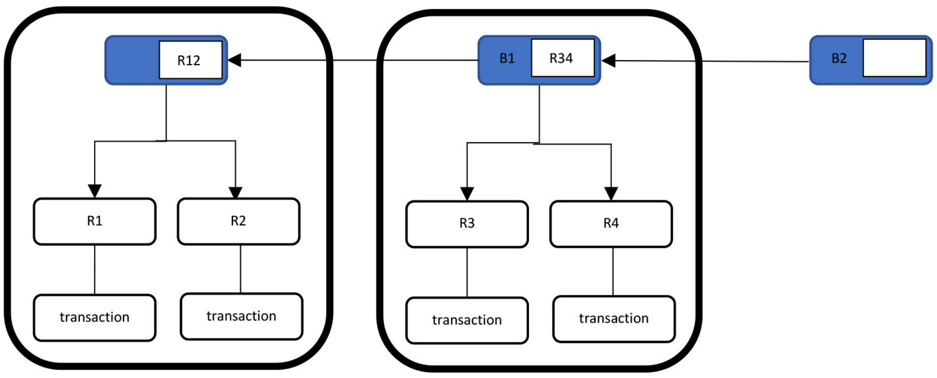

Figure 3: Blockchain-data-structure

Now, we will explain how a block is composed. As shown in Figure 3 The block structure is a Merkle tree[32] structure on three different levels:

- **Block Header** contains header of the block if it is the first block the hash reference number is just the reference of transaction. The headers linked together to form blockchain data structure
- **Transaction reference** contains the reference of combined hashing between two different transactions (Merkle tree).
- **Transaction** contains two different transactions data.

It is important to remember that the blockchain data structure consists in a series of linked "header" but there is only one head, the last block.

3.1.5 HEADER AND MINING

The main purpose of the blockchain platform is to preserve the blockchain-data-structure to prevent any malicious intent. This goal is achieved through the model with which blocks are made. Block model makes the whole system immutable[2] and high computational costs, these are the requirements to make a system able to prevent malicious intent. The block's model can be explained through creating a block and adding it in blockchain-data-structure. As we have previously seen the most important part in a block is header, each header contains mandatory data. Hash transactions reference number and hash reference of the previous block, already mentioned above, make the system immutable, preserving the risk of changes to the structure, we will discuss this in the next paragraph. In addition to these two elements, in the header, there are three other elements, mixed together in one hash value (mixed hash): the difficulty level of hash puzzle, the start time of solving the hash puzzle and the nonce of test value. But why there are these values in the header? **Hash puzzle** is that function which needs computational resources to solve the puzzle before to add a new block a node has to solve a relative hash puzzle. So, the hash puzzle is unique for each block and is used to increase computational costs of blocks creation. So, the hash puzzle increasing computational costs makes blockchain-data-structure alterations more difficult. To ensure that nobody tries to change computational cost the difficulty level is part of the header and then of block's hash value (as we will see later edit blockchain-data-structure requires a very high computational cost). **Creation of a new block** is called mining and the nodes that run this process are called miners. This process has different steps (we will follow the Merkle tree structure like in figure):

[2]In imperative programming, values held in program variables whose content never changes are known as constants to differentiate them from variables that could be altered during execution. Examples include conversion factors from meters to feet or the value of pi to several decimal places. Read-only fields may be calculated when the program runs (unlike constants, which are known beforehand), but never change after they are initialized.

1. Miners listen for transaction broadcasted and they select validate ones (generally miners choose transactions with a high transaction fee)
2. transactions are added to a new Merkel tree model (transaction3, transaction4);
3. generate the hash reference from the header of predecessor block (B1);
4. take the root of the Merkle tree and generate the hash reference of transactions (R34);
5. obtain difficulty level of the hash puzzle;
6. get current time;
7. generate a preliminary block with the information obtained previously;
8. solve the hash puzzle;
9. add the solution of the hash puzzle to the preliminary block.
10. add the components to the header and generate a mixed hash

So, the hash reference of block header will be the combination of hash reference of the value of that process; the new block will be added to the chain and it will become head of the chain. To create a new block there are some **validation rules** that a node must be respected if these rules are not respected the blockchain-data-structure detects invalid request and it refuses the block adding request.

- The **first validation rule** ensures that there are no changes within block structure, and then compared to the other blocks of the blockchain-data-structure. The changes could invalidate the whole blockchain-data-structure, following the Mark tree structure (represent in figure 11): if change or replace the content of transaction (transaction1 or transaction2) the reference is broken (R1 or R2) and hence, also, the reference of the Merkle tree(R12); this alteration could causes later the invalidation of the whole block and of then blockchain-data-structure because of whole the reference of the next block is linked with previous one (moreover, even alteration of only a hash reference of one element of the structure causes invalidation of the whole blockchain-data-structure).

- The **other validation rules** are linked with the last three elements and they require: that the difficulty level is correct; timestamp is after the timestamp of the preceding block; the block contains nonce.

According to Drescher, if someone tries to manipulate a piece of transaction data in blockchain-data-structure even just with 20 blocks to avoid creating inconsistencies in the structure must rewrite every block (first rule) and resolve 20 hash puzzles (second rule)[33]. Generally, to solve a hash puzzle takes 10 minutes, so it will take 210 minutes for the entire structure. The hash puzzle is the most fundamental element to make blockchain-data-structure immutable.

3.1.6 COMPETITION

The main important purpose of a peer-to-peer platform is that every node has the same history or transaction history. The most challenge is linked to the internet. In distributed peer-to-peer system internet is used as a communication medium, the internet gives the opportunity of unique address, text messages, connection any times. But at the same time, messages can not arrive or arrive in a different order. So, the peer-to-peer system must guarantee communication between nodes in the system, to achieve this result the distributed peer-to-peer system distinguishes three different cases:

- when the connection between peers is already existing each node checks, sending a small message called ping, that other nodes are reachable. Then the receiving node confirms the request by sending another message, called pong. Nodes that do not a response to the request are removed from the nodes list. This method is called ping-pong;

- when a new node is added to the peer-to-peer system, the system takes care of creating a new connection. The new node sends a request to other nodes in the network, so who receives the request activates the ping-pong method and add the new node to the list;

- when the information needs to distribute among nodes in the network, it is the most important communication challenge for the system. The blockchain is a distributed system, without central control, so there is no entity that can handle communication between different nodes. Nodes communicate with each other through transactions contained within the blocks. As we have seen before, these blocks are chain together, so the question is: how are managed the communications in the blockchain system (I.e. adding and validation transaction) without a central unit? We will see how the blockchain system achieves this goal.

The process of sharing information between nodes of a blockchain system concerns adding and validation transaction/block, then validation rules (discussed above). But, in the previous paragraph, we omitted what happens when block fulfils (or not) the requirements of validation rules and the way which **nodes compete to add a block to blockchain-data-structure**. Since there is no central unit that manages the system, blockchain-algorithm makes nodes supervisors that the validation rules are respected. But without the central unit, there is a problem of **coordination among nodes**[34]. According to Drescher, the core of blockchain-algorithm is **"working-rhythm"** that it allows coordination in a blockchain system in two different rhythmic steps.

- When a node receives a transaction, immediately the node processes a new block (first step).
- So, block follows the steps of Creation of a new block, described above, and it sends the new block to all other nodes (second step).

After that, the recipients of new block control that it is valid or not. To maintain the integrity of a blockchain system is necessary that the blocks are generated through high-quality work. So, to achieve this goal, blockchain-algorithm makes the creation of block a competitive system with **rewards** or **punishments**. The main idea of blockchain-algorithm is to create one block at a time and to reward the node that generated the best block, but reward requires computational resources. To prevent loss of computational resources there is a competition. Competition is a combination of **speed and quality**. Following the working-rhythm process, nodes that participate in a competition to generate the block have to solve the same hash puzzle (first step). So, the node that first solves the hash puzzle (speed) sends the block, with proof of work, to the other nodes (second step), these nodes add the block to blockchain-data-structure and validate the new block-based on transaction data and block header (quality). If the block is valid the creator's block receives the reward and a new competition is open. While the block is invalid the node has withdrawn the reward (punishment), the block is removed from blockchain-data-structure and the speed competition is reopened.

3.1.7 TRANSACTION HISTORY

As mentioned previously, the message in the blockchain system can be lost or messages can arrive with a different order. Moreover, previously we saw that every node, after the competition, keeps the block in its own blockchain-data-structure[35]. So, we can deduce that **not** all nodes have the **same** blockchain-data-**structure** and then the same **transaction history**, i.e. hierarchy of transactions. To handle this problem blockchain systems use **distributed consensus** (a topic addressed in the chapter on community peer), i.e. all nodes use same agreement to select the transaction history. Even if the competition method is the best criterion for selecting blocks to insert in the transaction history, there may be other conflicts due to errors in the exchange of messages between nodes. To solve this problem all nodes must agree to the same version of the transaction history. There are two criteria:

- Longest-chain
- Heaviest-chain

The longest chain is the criterion that **considers the chain where most of the blocks aggregated** between them. We consider two blocks joined together with a certain hierarchy, branch of the tree, and two nodes. One of these nodes has the complete transaction history, complete branch, while the other one has only the first block, only on part of the branch. All of two nodes won a competition and we would like to add a new block to their branch. So, while the first node will add own block to the second block, the second node will add his block to the first one. The longest chain rule indicates that the blockchain-data-structure is that of the first node, as it is the longest chain or longest branch. The rejected block will become an orphan block. Two important elements of this criterion are: if a node has two or more same length branch in his transaction history, he is free to decide which branch extend; the criterion develops a blockchain-data-structure of the most shared blocks, i.e. more nodes have a given block in the transaction history more likely the block is part of the blockchain-data-structure.

The Heaviest chain is the criterion that uses **difficulty level to choose blocks** for the transaction history. The difficulty level of transaction is determined dynamically, every block has in the header a different difficulty level for the hash puzzle. This criterion works like longest chain criterion but determines the next block of the blockchain-data-structure based on difficulty level, greater the difficulty level of the puzzle hash, so greater the probability that the block will become part of the transaction history. If two blocks have the same difficulty the block will be chosen with the longest chain criterion.

What happens to the excluded blocks? Blocks that do not become part of history are called orphan blocks. As we have seen previously in quality competition, the orphan block is removed from the blockchain-data-structure and the reward withdrawn from the node. When a block is discarded it disappears from the transaction history, so orphan blocks are treated as non-existent blocks and another opportunity is given to the transaction data contained in the block. Indeed, orphan blocks' transaction data will become a part of new block creation and therefore a new hash puzzle competition.

Blockchain-data-structure develops a tree-shaped where the branches that do not fulfil the criteria are cut. Development of the tree is governed by nodes that wins the competition and then can choose which branch to extend (random process). This authoritative path of choice the branch added to the immutability of the blockchain-data-structure makes any attempt to manipulate the decision-making process very difficult. Indeed, these two elements are linked since a computational effort (hash puzzle) must be made to enter the decision-making process (to add a block), then the whole blockchain-data-structure is the sum of the majority of decision-making processes (or sum of the majority of computational efforts). The majority because only blocks not discarded became part of computational power. For example, one of the possible manipulations to the structure could be the exchange of the authoritative chain with the orphan blocks and generating a new authoritative branch, this could undermine the integrity of the system (51 per cent attack). The

51 per cent attack hardly manages to manipulate the structure because the decision-making process is collective and, above all, because this process is preserved through the hash puzzle, that it requires 51 per cent of the computational power used until now to change blockchain-data-structure.

4 SMART CONTRACT AND ETHEREUM

Ethereum was born from Vitalik Buterin, according to him the blockchain technology, in particular, bitcoin is a limited technology because blockchain platforms have a very limited script and do not allow any modification. Ethereum, according to the idea of Buterin, must be a blockchain platform that gave the possibility to develop its own script "allowing unlimited freedom in creating a feature set, but also for the cost of development time, bootstrap effort and security. Using scripts is easy to implement and standardizing while being easy, suffers from scalability errors" (Vitalik Buterin, 2019). These scripts are called smart contract.

4.1 SMART CONTRACT

According to Nick Szabo, the first who theorized the concept of smart contract, a smart contract is:

"**A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to common contractual conditions (such as payment terms, liens, confidentiality, and enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.**"[36]

In general, smart contracts are computer programs that have different features, the main important are:

- **Enforceable** is that feature that allows executing the script contained in the smart contract without any mediation smart-contract without any mediation
- **Secure** and **unstoppable**, these programs must be written in such a way that avoid exceptions and are quickly compiled

- **Automatically executable**, smart contracts must be managed in a smart way, i.e. they have to be completely automatic and follow the manual inputs. Often smart contracts follow the state machine model

- **Semantically rules**, making a smart contract semantically understandable preserves in legal situations and allows understanding to those unfamiliar with programming languages

According to Imar Bashir "smart contracts are deterministic in nature", i.e., like for computational efforts in blockchain systems, every node executes a smart contract in the same way and achieves to the same result. There may be programming languages, which in different contexts can produce different results for the same variables, so it is advisable to produce a high-level code. One of the first smart contracts was Ricardian contracts. According to Ian Grigg, the main idea of Ricardian's smart contract must be" easily readable by people (like a contract on paper), readable by programs (parsable like a database)". [37]. This contract is drawn up through a document, signed by the issuer, which contains both the legal and digital aspects (code). Issuers sign the contract with a private key and the contract is encrypted via a hash function. Each transaction is linked with identifier hash to preserve against malicious intents. We can distinguish three different parts in Ricardian contract: a semantic part that indicates the legal contents and some parts of machine code; then the contract is linked with the hash value, it is used to have unique and secure contract identifier; in the last part there is a genesis transaction that uses hash value to for the first time and after it is used by each transaction to execute the contract. The main difference between the Ricardian contract and smart contract are related to semantic issues. While smart contracts are mainly related to the execution of the programming code and do not have an agreed document, Ricardian contract is more focused on the semantic question of the agreement (prose, parameters and code).

4.2 ETHEREUM

Ethereum achieves the goal of creating a completely customizable blockchain platform through smart contracts, then new conceptual applications called DAO (Decentralized Application Oriented). To do that, the Ethereum system manages the interaction between different components: p2p network, Ethereum client (that works on the nodes) and web3.js library (used to link with user interface). Ethereum has a cryptocurrency to reward the miners, this currency is called ether. According to the definition of Ethereum in the yellow paper written by Gavin Wood:

"is a project which attempts to build the generalised technology; technology on which all transaction-based state machine[3] concepts may be built"[33].

4.2.1 STATE ACCOUNT

Each node of the Ethereum system is represented in the database by the account state. In Ethereum there are two different types of accounts Externally Owned Account (EOA) which it is normal blockchain account that manages own private key, it can generate transaction, and can trigger smart contract; Contract Account (CA) has associated code, which is a smart contract, it can be executed by the contract account through a transaction or message, and each mining node executed its code. In turn, CAs can trigger other contracts. Each **account** has different **parameters**:

- Nonce is a number which in EOA is the counter of the transaction,

[3]A blockchain can be viewed as a state transition machine whereby a state is modified from its initial form to the next and eventually to final form as a result of transaction execution and validation process by nodes. So, the initial state represents the state before the transaction execution and the final state is what the morphed state looks like.

while in CA it represents the number of times when the contract was activated (how many times the accounts created the contract)

- Balance represents the amount of cryptocurrency (ether) of account, EOA can send/receive ether while in CA accounts it represents the amount of gas payed
- **Storageroot** an encoding 256-bit number that represents the root of a node in the Merkle Patricia tree[4]
- Codehash in CA accounts it represent hash of smart contract code while in EOA accounts it represents an empty field.

4.2.2 MESSAGE AND TRANSACTION

In Ethereum there are two types of fundamental unit: message and transaction. All of two types of data have a number of common fields, these **fields** are:

- GasLimit is a fee, it indicates the maximum amount of Ethereum currency (called gas[5] submultiple of the Ether, 1 gas = 0.00001 Ether) that you are willing to pay to execute the transaction, or the maximum amount of gas that smart contract execution triggered by the message can incur. Moreover, GasLimit can be seen as the maximum number of steps which sender want to execute in the transaction
- To is the address of the receiver

[4]A PATRICIA trie is a special variant of Merkle tree, in which rather than store the hash key, the nodes store only the position of the first bit which differentiates two sub-trees.

[5]as we shall see later, to execute a contract the node that validates the transaction (and its smart contract) makes some computational efforts. The gas is a unit of measure necessary to avoid that there are infinite loops in the code due to errors or malicious peer.

- Value is the amount of Ethereum currency that in message or transaction are transferred, while in a contract transaction it is maintained
- Init or Data, init, used just in the transaction, is a byte array of unlimited length that is used to store the smart contract, while data, instead of init, is used to store the input data in message or transaction.

In Ethereum network EOAs play a fundamental rule, they can forward transactions and messages. The substantial difference between messages and transactions is:

- Transaction can be of three different types: fund transfer between EOA, smart contract creation and change the status of an existing contract. The first type of transaction is a normal transfer of funds between EOA and does not provide for the execution of any smart contract. While, in the other two types EOA acts, through a transaction, creating or altering the status of a smart contract and then of CA. In smart contract creation, EOA sends a contract creation transaction that is deployed by miners node with EVM (a virtual machine to deploy contract), then if the transaction is validated a CA is created in the Ethereum network. The last type of transaction is used when an EOA or CA wants to execute a specific function of another CA. CA can activate this type of transaction only indirectly by sending a message to the recipient CA. The transaction has some additional fields compared to those mentioned above: signature through public-to-private key encrypt the transaction, Nonce a unique number in the entire system that is incremented each time a user sends a transaction, and GasPrice the amount of gas that you willing to pay to execute for each computational step of the transaction. Moreover, a contract creation transaction has a byte array of arbitrary length and EVM initialization code

- Message can be sent by contract to other contracts or uses to transfer information from one OEA to another. Messages are similar to the contract the main difference is that can be produced by contract and it does not involve the use of gas. Messages are the serial object of data and value. The message is transmitted only between the two accounts involved in the communication, it does not broadcast in the network and it does not become a part of blockchain-data-structure. In addition to the fields mentioned above the message has the field of who sends the message because unlike the transactions the communication is univocal between two nodes of the network. When a node wants to account a message uses the call function. For example, given a SimpleStorage contract (contract with a function that stores a number) when an OEA (sender) wants to change the state of SimpleStorage contract needed to know what is the state of store variable in the contract. OEA could ask the variable state through a transaction, but it would involve the use of a quantity of gas. So, OEA uses the call function, then a message, to ask the state of store variable to the SimpleContract. After not having found any exceptions, OEA acts changing the value of the variable sending a transaction. Finally, the transaction executes the SimpleContract's function.

4.2.3 WORD STATE AND EVM

Word statein Ethereum is a map that indicates the "state" in which the system is situated. "The world state (state), is a mapping between addresses (160-bit identifiers) and account state" [33]. As we mentioned previously Ethereum is a state machine that mapping its state with a recursive algorithm RLP (recursive length prefix). RPL encodes data of account state, from string or list of items, into binary data that can be stored in Patricia tree, state database. So, following the whole Patricia tree data concerning a user can be stored or transmitted. his structure, although simplified compared to the Merkle tree, has the same immutability benefits.

To process the states, Ethereum has an Ethereum virtual machine (**EVM**), it is used as a stack-based machine to bring the system in another state. EVM is a virtual machine that allows the execution of complex codes (smart contracts). EVM runs on the nodes physical infrastructure that **storage** and **processing** smart contracts. Each node of the network has its own EVM and performs, in particular, the same set of instructions as the other nodes. To preserve the integrity nodes' physical infrastructure EVM does not have access to external resources, like a file system. The smart contracts are developed in high-level code, with solidity languages. Solidity is an object-oriented programming language to write a smart contract based on JavaScript. EVM works with bytecode. Smart contract before being executed, it is compiled by EVM compiler, named **solc**, and it produces the bytecode (with maximum stack size is 1024 elements) which can then be interpreted by the EVM. Moreover, EVM manages the exception to avoid not enough gas or invalid instruction. EVM executes the bytecode one byte at a time, once the single byte is processed it updates its counter called **program counter** (PC), the process is performed with an infinite cycle. The internal virtual structure of the machine is composed: **storage**, **memory** and **stack**. The stack is a container in which (temporary) byte of bytecode are executed, the bytes can be entered (PUSH) or picked up (POP). Memory has a byte of arrays where code is saved before to be executed, the memory is volatile once the program has been executed the data are deleted. In the "storage" the bytecodes are saved permanently, in form of key-value. The main function of the EVM consists of three steps:

1. through the CODECOPY function the program code is copied

from storage to memory

2. EVM reads the program from memory, then EVM runs the code byte by byte in the stack
3. The EVM counter (PC) is updated with the current step of the code

Each state of the transaction is updated after every execution of the code. In order to execute the code EVM needs of some key parameters: address of the code owner, address of transaction sender, gas price, address of the code initializer, transaction value, code as bytes array, block header, number of executing transactions. Ethereum EVM can be seen as Turing state machine[6]. EVM in the iterates its function on every element contained in the stack(infinite loop), the **iterator function** follows the following steps:

1. It takes the instruction of code from byte array in the memory
2. It adds/removes (PUSH/POP) a message to the stack
3. It reduces the gas amount
4. It increments the program counter (PC) and it goes to the next instruction

[6]The machine operates on an infinite memory tape divided into discrete "cells". The machine positions its "head" over a cell and "reads" or "scans" the symbol there. Then, as per the symbol and its present place in a "finite table" of user-specified instructions, the machine (i) writes a symbol (e.g., a digit or a letter from a finite alphabet) in the cell (some models allowing symbol erasure or no writing), then (ii) either moves the tape one cell left or right (some models allow no motion, some models move the head), then (iii) (as determined by the observed symbol and the machine's place in the table) either proceeds to a subsequent instruction or halts the computation

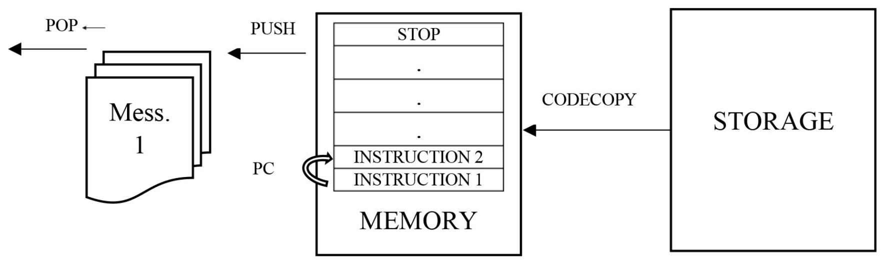

Figure 4: EVM iteretion function

4.2.4 ETHEREUM STATE TRANSACTION

So far, we saw the virtual machine acts by changing the transaction status to low-level. At a high level operates the process called Ethereum State Transition Function, this function is used to validate the transaction. So, the most important role performed by accounts is the textbf-validation of transactions, this process involves the use of the EVM in a transaction where there is a contract.

GAS Before to start to describe the process of transaction validate it is important to make a record because the gas price and gas limit are used. Gas is different from the normal fees used in the blockchain application. As previously mentioned, fees in a general blockchain platform are paid to miners node for its computational effort to generate a block, while the Gas is a new Ethereum concept. The Gas is paid to the miner for each resource that transaction sender "consume, including computation, bandwidth and storage executed"[38] with her/his transaction. Gas was introduced to avoid accidental or hostile infinite loops code and discouraging the use of too many computational resources in the smart contract. The transaction gas cost is given by the gasUsed*gasPrice, where the gasUsed is the amount of gas supposed that it will be used for the transaction. A transaction sender can decide what will be the commission of gas to pay for the transaction, however, miners could not include the transaction in the blocks because the value of invested gas is too low. The transaction, in addition to the possibility of being rejected by the miners, when it exceeds the amount of calculated gas can generate an out-of-gas (more operation cost than previously calculated) exception. So, it is advisable before to broadcast a transaction compare the cost of the operations with the table of planned operations[**?**] offered by Ethereum, and then decide which will be the amount of gas spent.

TRANSACTION VALIDATION Once the sender's account has completed a transition, it transmits the transaction over the network. All of the OEA **execute** and **VALIDATE** the **transaction** through of the Ethereum state transaction function, now we will see which are the main steps:

1. the first step consists in verifying the semantic correctness of the transaction, i.e. valid signature and nonce equal to the sender's nonce

2. fee is calculated as gasLimit*gasPrice, then the fee is subtracted from the sender by determining the address of the signature.

3. EVM calculates and subtracts from the paid fee the cost of gasPrice

after these first three steps the transaction is validated and then executed:

1. The transaction value is transfer from the sender to the receiver if the transaction is a transfer transaction. Otherwise, in contract creation transaction a CA is created, and the balance CA is set to the value passed with the transaction; while if the destination is a contract, the contract code is executed until there is gas in the transaction

Once the transaction has taken place, there are two cases:

1. Transaction maybe fails for insufficient sender balance or because the transaction not have enough gas to complete contract code. In these case, all states are revert except miners fees.

2. If the transaction succeeded the amount of advanced gas is sent back to the sender and miners fee are paid.

TRANSACTION RECEIPTS At this point the state return to resulting state. According to Imran Bashir, "**Transaction receipts** are used as a mechanism to store the state after a transaction has been executed"[39]. All receipts are stored in a Merkle Patricia tree and the root of this tree is a component of block header. Transaction receipts is described by 4 different components:

- A byte array, named post-transaction state that keep the transaction state after its execution.
- A number that registers the amount of gas used. It is stored after the execution of transaction.
- A log of the transaction a series of information of log code
- A filter that contains logger's address, series of log topic, log data decoded in the form of a hash (boom filter).

4.2.5 BLOCK

The **blocks** in Ethereum are fundamental unit, as discussed above for the blockchain platforms. In Ethereum the blocks header contains a list of transaction, block header and a list of orphan blocks. It is important to keep a list of orphan blocks because, as we will described later, given their importance in the transaction history, even those the nodes who mine them receive a reward. The block header is composed of a series of hash fields. In addition to the components mentioned above in the block creation, i.e. hash value of: list of transaction, previous block, puzzle difficulty, current time, address of creator and once, in Ethereum block header has other component linked to the structure of this platform. The other **components** represent by hash number are: list of orphan blocks, gasLimit is the limit of gas set by miner that the block can consume, gasUsed by transactions included in the block, list of all transaction receipts associated with transactions of the block, logs boom is a list of every recipct log of the transactions, nonce a prove of computational effort, and mixhash a combined hash between once and nonce.

MINING So, also the block creation (or **mining**) process in Ethereum respect the rules described in the introductory part of the blockchain structure. But there are some few differences due to the intrinsic characteristics of the Ethereum platform. These factors are orphan blocks and state of the transaction, in general, we can describe the mining process as a series of steps taken by the miners:

- listen to the broadcast transaction on the network and they chose which transaction will have to include according to the "gas criterion", i.e. to pick up a transaction with the high level of gas, to receive more reward and to avoid "out of gas" exception;
- perform the steps listed above in the "Creating a new block" from step 2 up to 10, using the fields of the Ethereum block add a list of orphan block linked of their block.

VALIDATION With the creation of the block, the first working-rhythm step is concluded. Now, then we pass to the **validation of the block** which is the second working-rhythm step. Also here, the structure is similar to the one described above, although with some differences. Once the block is broadcast in the network all the nodes receive it and perform checks on the validity of the information contained in the block, i.e. the second working-rhythm step, through the following steps:

- Check if all branches with the orphan block are satisfied, then the orphan blocks is valid
- Check if hash reference of previous block (parent) is correct
- Check if the timestamp of the block is between 0 excluded and 15 minutes of the parent block timestamp
- Check if the information of the block are correct, i.e. transaction root, gas limit, the difficulty of the puzzle, proof of work (nonce is correct), and hash number
- Check if the amount of gas spend by the transaction in the transactions list is less than the gasLimit of the block
- Check if the state of the last element of the root of Merkle tree is equal to the state of the block header

Competition in Ethereum is based on proof of work (or how to solve the puzzle) method called **Ethash** and it has the same characteristics set forth in the section competition in the chapter of the introduction of the blockchain technology. it was done on an algorithm that makes it difficult to use powerful tools to mining called DAG (Directed Acyclic Graph). DAG requires the node choosing subsets of a fixed its resource, dependent on the nonce and block header. DAG is a 2 GB memory, roughly 3000 blocks, which is updated every 5.2 days and must be preloaded by miners.

In the introduction chapter of blockchain technology we described the choice of **difficulty level**for hash puzzle as a random process, while Ethereum uses an algorithm to achieve this goal. Let the interval, T, an interval between the creation of two blocks, Ethereum algorithm decreases the difficulty of the puzzle hash of next block in a proportional way with the increase of time T. If the time T is less 10 seconds, the difficulty goes up in a proportional way. If the time T is between 10 to 19 seconds, the difficulty remains the same. If the time T is more 19 seconds, the difficulty level decrease in a proportional way. Moreover, the algorithm exponentially increases the difficulty every 10,000 created.

The reward when a block became part of transaction history is now 5 ether. In addition, miners receive the amount of gas consumed by the transaction within the block.

4.2.6 GHOST CRITERION

We saw that in the block a list of orphan blocks is saved and the need, in the block validation process, to verify that the orphan blocks linked to the relative block are valid. According to the role played in Ethereum, you now describe how orphan blocks become part of the process of reaching consensus. According to Zohar and Sompolinsky, in their paper **Secure High-Rate Transaction Processing** in Bitcoin, have shown that long or heaviest chain criterions are vulnerable to the 51 per cent attack and therefore to the double-spending problem. As demonstrated by Zohar and Sompolinsky, when there is a great deal of block creation, probably when an attack is in progress, it is certain that if the growth rate of the main network chain is less than the speed at which attackers create the blocks, the main chain will be determined by the malicious nodes. For example, following the longest-chain criterion, given a blockchain system, which quickly generates blocks, and given the longest branch, the malicious node could insert a chain block, previously generated, longer than the main one and therefore alter the transaction history. The protocol proposed by Zohar and Sompolinsky is called GHOST (**Greedy Heaviest Observed Subtree**). Ethereum to achieve the question of distributed consensus uses GHOST criterion. GHOST criterion considers orphan blocks as an important resource. GHOST "algorithm follows a path from the root of the tree (the genesis block) and chooses at each fork the block leading to the heaviest subtree" [40]. Heaviest subtree means that given two blocks that are part of two different branches with the same length (or branches with the same difficulty) the block that has **multiple chains with subblocks will be selected**. This criterion prevents malicious nodes from entering its pre-processed blocks chain.

Moreover, with this criterion, there is a higher level of security than other criteria because the network's capacity for growth of the main chain is limited (**delayed block propagation**). A network with high confirmation rate worsens security levels because there are more orphan blocks. The vulnerability is connected whit propagation rate of blocks in the network. As demonstrated in the Ethereum white-paper, the blocks need some time to propagate between nodes, then if a node has a lower computing power level than another node it is more likely to produce orphan blocks "and will not contribute to network security" [38]. Delayed block propagation has the effect of expanding the time to confirm a block in the transaction history and therefore to agree with the same history for all nodes in the network. Another problem with long or heavier criteria, always due to the difference in computational power between nodes, is ability of the node with greater power to carry out more mining, so this could transform the blockchain into a centralized network. To solve this issue GHOST provides to reward the orphan blocks, in Ethereum these blocks are called Ommers or Uncles. Rewarding the uncle even if they do not become part of the transaction history is a way to encourage the block mining by small nodes, i.e. those nodes that do not have much computational power to spend in the network.

For block validation and reward, Ethereum develops a **reduced version of ghost criteria**. Initially, uncles must be validated. All uncles are considered valid up to the seventh level with respect to the block to be validated. So, to be valid an uncle must have the following properties:

- belong to the root transaction history through a link with the block, this link may extend up to 7 sub-levels compared to the valid block

- the only requirement of an uncle is to have a valid header
- an uncle must be unique, uncle must be different both from others connected to the block and from the uncles of the previous blocks.

Ethereum establishes an extra of 1/32 corresponds to each uncle to the generator of the block. While for each uncle generated Ethereum corresponds 7/8, of the current value of a block, to the generator of the uncle.

4.3 DEPLOY SMART CONTRACT

In order to achieve the development of the contract there are several tools that through writing, testing, verification and implementation make the development of a smart contract accessible to all. These tools offer the opportunity to develop code for the smart contract, using high-level programming languages, understandable for EVM. So, the main feature of these tools is to translate the high-level language into bytecodes. In this section, we will analyze the most used programming language for the development of a smart contract, Solidity. I will also try to understand what are the steps to follow to develop a contract and what are the main security issues.

4.3.1 SOLIDITY

According to the definition, Solidity is "an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state." [41] Solidity takes the main features from other languages such as JavaScript, C and Python. A smart contract written in solidity is executable by any user on ethereum, it is compiled in bytecode through the EVM present on every node of the network. As previously specified, each contract on the Ethereum network is represented by an account (contract account) and it has a unique address through which it can be invoked. Solidity is also defined as a statically-typed programming language, "which means that the type of each variable (state and local) needs to be specified"[41], then in solidity "undefined" or "null" values generates an exception.

When writing the code, you must declare: the compiler version must be declared outside the contract code and any libraries that you want to import, while all of the other elements must be written within the contract:

Listing 1: layout of source code

```
//compiler version
pragma soliddity '0.4.22
// import, for example other contracts
import ''module-name''
//contract name
contract exampleContract{
    //some logic
}
```

So, solidity is a set of data types (that define the account state) and function types (that define transaction state). Data types are of two types: value (the value is actual value) and reference(the value is a reference to another value). Value types represent Boolean variables, integers, addresses, and emuns. Moreover, it may be in the form of array or literals (fixed value).

VALUE TYPES Integers can be signed (be both negative and positive) and unsigned (don't allow negative numbers). Like in C every integer is followed by the allocated space, from 8 to 256 bits, to better optimize memory, for example: `uint256 x` (represent an unsigned integer that allocates 256 bit of memory); `uint y` (represent unsigned integer that allocates 256 bit of memory); `uint8 z` (represent signed integer that allocates 8 bit of memory). The size allocated space varies by steps of 8 bits, i.e. 8, 16, 24 etc. With ”constant” after uint no need to add memory to allocate because no storage slot will be reserved. Address data-type holds a 160-bit. This value represents the address to reach a network account (EOA). The address has different queries to interact with its:

Listing 2: address example

```
pragma solidity ^0.4.22;
contract send{
   //indicates the owner of the contract
   address owner = this;
    //indicates the ether balance of the owner
   uint256 balance = owner.balance
   // address of the receiver
   address receiver = 0x1254ff07j456gbf6th5f2d0t9h550vf6j5676yj;
   if( balance>=20)
       //this query transfer 20 Eth from owner to receiver, if
           the balance of the sender is greater or equal than 20
           Eth
      receiver.transfer(20);
}
```

Transfer query can be used to interact with other contracts (the address receiver can be an address of CA). There are other queries which must be used with caution because they act at a low level, they are: send, call, callcode and delegatecode. Send is a low-level function of transfer, but when a transfer is called and the sender does not have sufficient funds or the code throws an exception the value is refunded; while in send, if there is an error, the value is sent and after the function returns false without any refund. The other three functions are functions to interface with contracts without Application BInary interface (ABI). As specified in the solidity documentation "All three functions call, delegatecall and callcode are very low-level functions and should only be used as a last resort as they break the type-safety of Solidity. The use of these functions will be removed in the future."[41]. Other **value types** are string, byte and enums. The string is a specific set of elements contains in double or single quotes, for example, `String s = "Hello, World!"`; byte is fixed-size arrays and they are declared using the keyword byteX, the X being any number from 1 to 32; for example `byte32 s = "Hello"`. So string and byte are similar but as demonstrated by Cryptopusco in his article, "bytes and strings in Solidity"[42], it is preferable to use byte to preserve the quantity of gas used, even if they do not support escape characters, such as \n, \xNN and \uNNNN; while enums are types that it defines by user, most of the time they are used to keep track of the status of the contract code.

REFERENCE TYPE Let's see which are the reference types. **Arrays** in solidity are a set of elements of the same size and type. Arrays can be static `uint[3] arrays` or dynamic uint[] arrays. The array has two queries length and push. To take the length of the array you need to use the query length `arrays.lenght`. The dynamic arrays saved in storage can be resized changed the length parameter, while arrays cannot be resized for if they are saved in the memory or if they have already been created. While push query, `arrays.push()`, is used to add elements to the dynamic array. In addition to the aforementioned arrays of uint, there are two other types of arrays: **strings** and **bytes**. The first one was mentioned at in the value types, but in the reference type, it has a dynamic dimension and has no query to interact with, the same for bytes are dynamic bite without query to interact with it. Push query can is used only with arrays and bytes, not with string. Solidity provides a reference type to group different data types into one logical types, this type is called Struct. Structs can contain any type including arrays and maps. The most important feature of Structs is that they can have multiple properties.

Listing 3: struct example

```
  //for example users can have different proprieties:
   struct User {
      //coin of user
      uint coin;
      //address of user
      address add;
   }
```

CONTROL STRUCTURE In solidity, there are different control structures which are: Functions, Events, Inheritance, Abstract Contracts and Libraries.

Based on how they behave mutually, in solidity we can distinguish the functions in two large groups: internal or external. The internal function is accessible within the contract itself and it is called recursively (via jump), while the externals one are out of the contract and can be accessed by other contracts via message call. Moreover, functions can be private or public, the first is visible only within the contract while the other one can be called internally or via message (functions are public by default). Functions allow two types of parameters: input or output (return functions).

Listing 4: functions example

```
//Anyone can an call this function
function example() {
  //some logic
}
//Can call only by the other functions in the contract
function example1() private {
  //some logic
}
//To return a value from a function
function example3() returns(uint) {
  return 3;
}
//it has the same characteristic of public functions, but it
    can call only by functions outside of this contract.
function example4() external {
  //some logic
}
//it has the same characteristic of private functions, but the
    contracts that inherit this contract can call its.
function example5() internal {
  //some logic
}
```

The internal call functions are very efficient as they do not delete the values present in the memory (see data location 4.3.1), while calls to external functions must be handled with extreme caution because the contract called could be harmful to the entire system and generate an exception if the contract called does not exist. Note the same variable cannot be declared twice within a function. All functions are fundamental elements to interact with a contract, we saw they have a different function. So, based on their function we distinguish View, Modifier, Pure, Fallback and Overloading functions. **View**, or constant, functions cannot change the contract state[7], it is a practical implementation of call concept discussed above in general introduction of Ethereum message;

[7]According to Solidity documentation are considered modifying the state if: Writing to state variables, Emitting events, Creating other contracts, Using self-destruct, Sending Ether via calls, Calling any function not marked view or pure, Using low-level calls, Using inline assembly that contains certain opcodes.[41]

Modifier functions are that functions that are executed only when the condition is satisfied, for example:

Listing 5: Modifier example

```
pragma solidity ^0.4.22;
contract main {
    //A mapping to store a user's coin:
    mapping (address => uint) public coin;
    // Modifier that requires this user to be richer than a
        certain amount
    modifier graterThan(uint _amount, address _userId) {
        require(coin[_userId] >= _amount);
        _;
    }
    // Must be greater than 16 to spent coin, we can call the
        'graterThan
    function spentCoin(address _userId) public graterThan(16,
        _userId) {
        // Some function logic
    }
}
```

Pure functions are equals to view function but they promise to do not read [8] or modify contract state; **Fallback** functions not have argument and name, they are called when the other functions do not match with the identifier, often this function contain payable marked it is those a marked that contain a mechanism to "collect/receive funds in ethers"[43], then the ethers are stored by the contract; **Overloading** functions are that functions which they have the same name but different arguments.

Events are structures that activate the storage of some states of the

[8]According to Solidity documentation are considered read the state if: Reading from state variables, Accessing this.balance or ¡address¿.balance, Accessing any of the members of the block, tx, msg (with the exception of msg.sig and msg.data), Calling any function not marked pure, Using inline assembly that contains certain opcodes.[41]

EVM logs; they are important because they allow you to manage the contract from external interfaces, such as a javascript dapp (decentralized application) that can call events through a callback. When an event is called, the result of the call to the EVM is saved in the transaction log. The log cannot be accessed by the contract, but events in the contract must be used to notify changes in the EVM log.

Solidity supports the **Inhertiance**[9] between contract. Solidity contracts accept more inheritance, i.e. inheritance from one or more contracts, this does not involve the presence of more contracts in the blockchain. In fact, the different contracts father will be copied in the created contract and in the blockchain-data structure will appear only the final contract. To indicate a hierarchy we use the construct `is`:

Listing 6: Inhertiance example

```
pragma solidity ^0.4.22;
//this is the contract father
contract father {
   function example1() returns (string) {
        return "This is first contract";
    }
//"is" means if you compile and deploy son it will have access
    to both example1() and example2()
contract son is father {
    function example2() returns (string, string) {
        return ("This is second contract and ", example1());
    }
}
```

As indicated in the documentation solidity uses "**c3 linearization**",

[9] "Each class has a superclass from which it inherits operations and internal structure"... "Class inheritance also provides a way to organize and classify classes, since classes with the same superclass are usually closely related", " leaving the original code intact".(1991, Ralph E. Johnson)

a new version of Method Resolution Order (MRO). The linearization indicates the order in which the hierarchy of classes (or contracts) must be managed. So, linearization[10] is a process that follows three rules in the composition of the hierarchical graph of classes:

- the hierarchy graph determines the structure of class positioning in the code;
- the daughter classes must inherit all the father classes, the scheme must be applied first locally than in general;
- monotonicity, the class hierarchy cannot be reversed once established.

Abstract Contracts are contracts which implement abstract function used to perform the Template method, a pattern for OOP. They are similar to abstract classes used in java language. These types of contract cannot be compiled and they must have one or more abstract functions (functions without implementations). See here an example:

Listing 7: Abstract Contracts example

```
pragma solidity ^0.4.22;
//abstract contract
contract Member {
    //abstract function, without any implementation
    function name() public returns (bytes32);
}
//Inhertiance
contract employer is Member {
    //call to abstract function
```

[10]take the head of the first list, i.e L[B1][0]; if this head is not in the tail of any of the other lists, then add it to the linearization of C and remove it from the lists in the merge, otherwise look at the head of the next list and take it, if it is a good head. Then repeat the operation until all the class are removed or it is impossible to find good heads.

```
    function name() public returns (bytes32) { return "Mike"; }
}
```

A **Library** looks like a contract but they are stored at a specific address. The most important feature of libraries is that are executed in the context of the contract that calls them and then a library can only access to the state variables of the contract that call it. This feature of the libraries makes them as reusable for contract storage, e.g. a different way to implement data structure. The libraries work like contracts but they do not appear in the function hierarchy, internal functions of the library are seen by the calling contract. There is a special method to call libraries the command `DELEGATECALL`[11]. As internal functions variables are passed like memory types and they are not stored.

DATA LOCATION As mentioned above, in EVM there are two different places where to save information memory and storage, while in the stack the pieces of information are executed by EVM. In high-level programming languages for smart contracts, as solidity, these places are represented:

- in storage is saved every state variables. This persistent memory is very expensive both computationally and gas. Struct, array or mapping are saved here by default;
- in memory are saved temporary values and it is cheaper than storage to use. Its internal value "is deleted between function calls (external)"[42]

[11]a special variant of a message call, named delegatecall which is identical to a message call apart from the fact that the code at the target address is executed in the context of the calling contract and msg.sender and msg.value do not change their values. This means that a contract can dynamically load code from a different address at runtime. Storage, current address and balance still refer to the calling contract, only the code is taken from the called address.[41]

- stack has very low or zero costs but is very limited, for example, it contains local variables(except variables saved in the memory).

By default reference and value types are saved in the storage, it is advisable to save these values in the memory, through the command `memory`, if they are used only inside the function.

Listing 8: library example

```
pragma solidity ^0.4.22;
library Sender{
   struct senderSt{
      bytes[] sender;
      ...
   }
   function add(address sender){
      sender.push(sender);
   }
}
contract send{

   uint amount = 0;
   enum Status(Initialized, Executed, Collected);
   Status private st;
   st=Status.Initialized;

}
contract addAmount{
   using Sender for Sender.sender;
   function payme() payable{
      using Sender for Sender.senderSt;
      //can be used to attach library functions (from the
         library Sender) to any type (Sender.senderSt in this
         case)
      Sender.add(msg.sender)
```

```
        amount += msg.value;
        }
contract distribution{

}
```

4.4 UPGRADABLE SMART CONTRACTS

According to the general definition, by Jack Tanner, upgradeable contracts are: "Creating a smart contract that can be completely replaced with new logic is possible just by using more smart contract infrastructure. There are two main streams of strategies: proxies and the separation of logic and data into different contracts."[44].
Once smart contracts are distributed in a blockchain platform they are immutable. These patterns consist of the division of contracts into two different levels of data and logic. So, the most important advantages of these patterns are that they make contract upgradeable i.e. they give the opportunity to fix bugs or introduce new features in contracts without needed to develop a new one. As mentioned by Elena Nadolinski in her article "Proxy Patterns", Parity Wallet Hack, where 150,000 ETH was stolen, could have been avoided "if only there were been a way to update the source code after the smart contract has been deployed"[45] The upgradable approaches, to write code for smart contracts, present high risks, as they may be vulnerable to malicious peers. The vulnerabilities that a peer could exploit are linked to the following factors:

- the amount of gas spent to executed different contracts may be high due to the amount of operation called by the different contracts. It is appropriate to limit the number of operations to be performed
- variables are compiled in the contract, then variables depend only by the contract where they are compiled. So, when contracts called

on each other that could can a ripple effect (inconsistency).

The main difference between separation and proxies pattern is that in the proxies record data contract (the proxy) uses `delegatecall` to call the logic contract. Above all, when the call between two contracts is made with delegatecall, there must be no inconsistency between the two contracts.

4.4.1 DATA SEPARATION PATTERN

In this type of pattern, users can interact with the logical part. In this type of patter is important to know which are the difference between the function used to store data (data contract) and the function to manage data (logical contract). In data, contracts are present getters and setters, only the owner of the contract can call setters. it is possible to modify persistent variables with a supplementary data contract but every data contract added has an additional cost, in term of gas. So, in this pattern is most important to understand a priori what the state variables will be. This pattern has several approaches to upgrade new logical contract:

- transfer ownership of the data to a new logical contract, and disable the previous one.
- add the new version of the logical contract to the old one, like a chain. When the user wants to interact with the logical contract, with the old or a new one, he will be redirected to the latest version of the contract and only the last version of the logical contract will be executed. This approach maintains every contract and therefore adds more complexity each time a new logical contract is added.
- add an additional entry point for users called proxy contract, then user calls proxy contract which in turn calls logical contract.

The risks associated with this type of pattern are mainly due to the complex data, then to develop with this pattern is needed to know the complex data structures of solidity.

4.4.2 PROXY PATTERN AND ZEPPELINOS

The proxy pattern works like data separation pattern but the main difference is that "proxy contract calls the logic contract with `delegatecall`" [46]. The proxy contract, the contract through which users can interact with the system, holds data and contains the address of the logical contract, so this pattern works in reverse order with respect to the previous. In this pattern the proxy contract redirects users to the latest logical contract deployed, then when a logical contract is upgraded, it is also updated the reference to the new contract address. The proxy pattern has three different approaches:

- Inherited Storage
- Eternal Storage
- Unstructured Storage

The three different approaches are used to handle the allocation of different contracts. The main problem is related to due that the proxy contract and the storage contract could overwrite each other, using a slot already used.

INHERITED In this approach the proxy and logic contract shared the same storage structure for the state variables, UpgradeabilityStorage, or Registry. In this way, both contracts accept the same proxy state variables, even when a logical contract is updated. As shown in Fig 5 Upgradeable contains the history of logic contracts, then to upgrade the logic contract need to store the new logic contract in the registry and update the address of the logic in the UpgradeabilityStorage. UpgradeabilityProxy contains the reference to the proxy contract and the address of the logic contract. The step to record a new logic contract are:

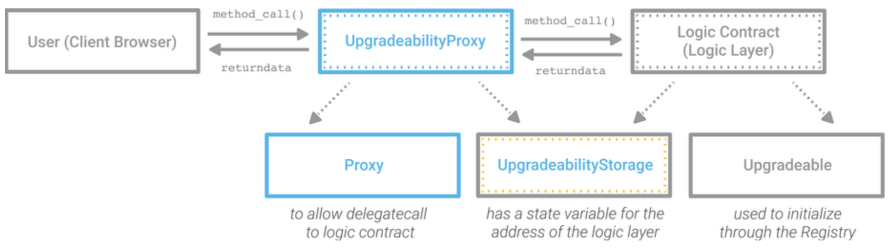
User (Client Browser)
method_call()
returndata
UpgradeabilityProxy
method_call()
returndata
Logic Contract (Logic Layer)
Proxy
to allow delegatecall to logic contract
UpgradeabilityStorage
has a state variable for the address of the logic layer
Upgradeable
used to initialize through the Registry

Figure 5: Inherited-storage-approach [45]

1. Develop the first version of the logic contract that it must be an Upgradeable contract
2. Develop a registry and add the address of the logical contract to it
3. Call the function in the registry to instance an UpgradeabilityProxy and to generate a new proxy contract

To update the logic all the steps must be followed with a new logic contract that it keeps the storage structure of both the proxy and the previous logical contract.

ETERNAL In this approach the storage holds all of the state variables of the logic contract, these variables are shared with the proxy contract. The future version of the logic contract will not have new state variables. So, the proxy contract, in a new version, could insert new variables without overwriting the logical variables. In this approach, only the proxy owner can upgrade the address to a new logic contract in the proxy. As shown in Fig6, we will see the step to record a logic contract are:

1. Develop the first version of EternalStorageProxy and logic contract
2. Update the address calling an instance of EternalStorageProxy (UpgradeabilityStorage)
3. Need to call upgradeToAndCall function in EternalStorageProxy to redo the setup of some function of logic contract. This hap-

pens because any state initialized in the logic constructor is not recognized by the proxy, due to the delegatecall.

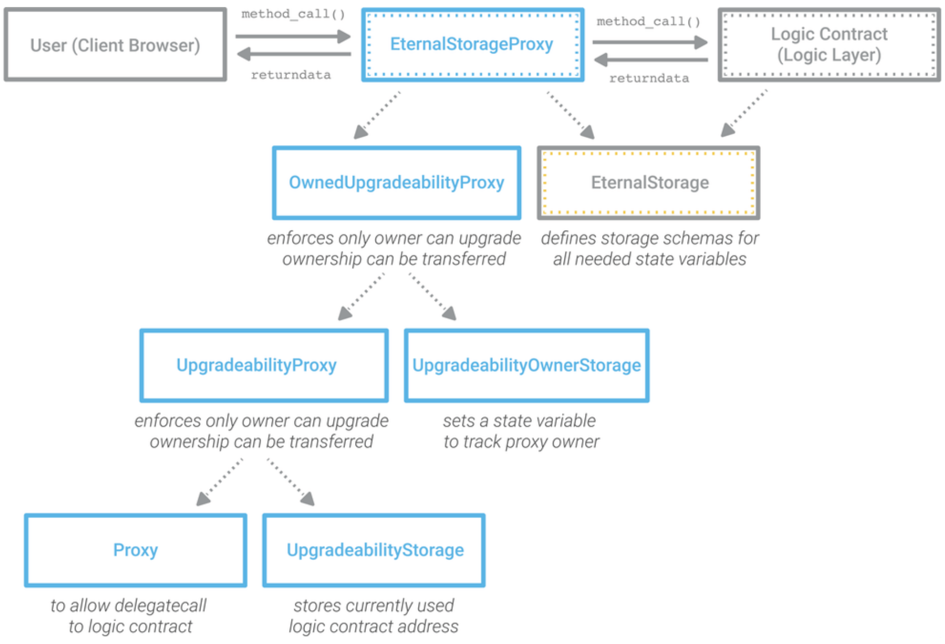
User (Client Browser)
method_call()
returndata
EternalStorageProxy
method_call()
returndata
Logic Contract (Logic Layer)
OwnedUpgradeabilityProxy
enforces only owner can upgrade ownership can be transferred
EternalStorage
defines storage schemas for all needed state variables
UpgradeabilityProxy
enforces only owner can upgrade ownership can be transferred
UpgradeabilityOwnerStorage
sets a state variable to track proxy owner
Proxy
to allow delegatecall to logic contract
UpgradeabilityStorage
stores currently used logic contract address

Figure 6: Eternal-storage-approach [45]

To update the logic needs to deploy a new version of logica contract and update EternalStorageProxy.

UNSTRUCTURED approach is similar to Inherited approach but the logic contract does not inherit the state variables of the update. As shown in fig 7, this approach used unstructured storage slot to save the data that requires an update. The most important improvement of this approach is that the state variables cannot be accidentally overwritten in a future upgrade. In this approach the proxy structure is completely separate from the logical contract, so the second does not know the inheritance of the storage variables. Only the address of proxy owner can upgrade the proxy contract to a new logic contract, and the only address that can transfer ownership. The main steps to upgrade a logic contract are:

1. Develop OwnedUpgradeabilityProxy and logic contract

2. Call an instance of OwnedUpgradeabilityProxy to update the address of the first version of the logic contract

3. As mentioned in the fourth step of Eternal approach also here there is a function, upgradeToAndCall, of EternalStorageProxy to redo the setup of some function of the logic contract.

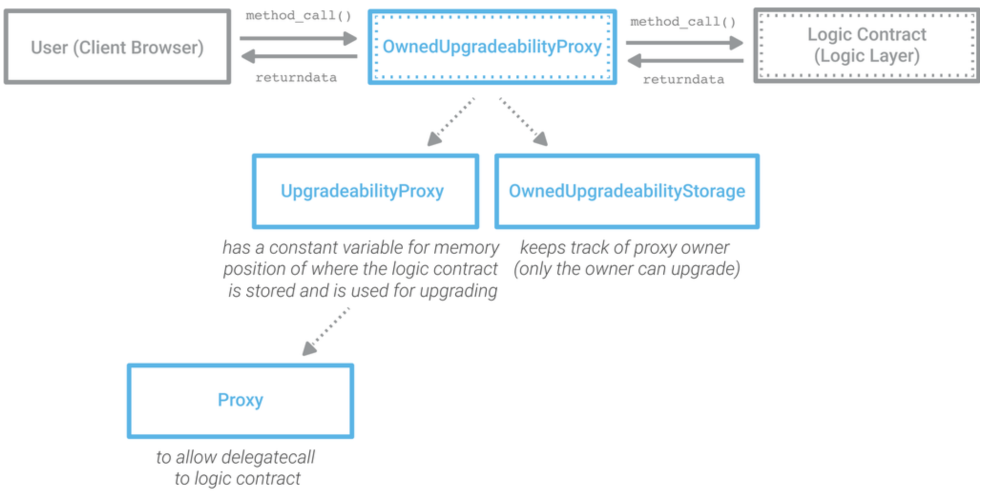
User (Client Browser)
method_call()
returndata
OwnedUpgradeabilityProxy
method_call()
returndata
Logic Contract (Logic Layer)
UpgradeabilityProxy
has a constant variable for memory position of where the logic contract is stored and is used for upgrading
OwnedUpgradeabilityStorage
keeps track of proxy owner (only the owner can upgrade)
Proxy
to allow delegatecall to logic contract

Figure 7: Unstructured-storage-approach [45]

5 SIMILAR PLATFORMS

Now we analyze two different similar platforms with the platform proposed in the dissertation

- CrowdBc
- BackFeed

5.1 CROWDBC

CrowdBC is a blockchain-based decentralization framework for crowdsourcing[47]. In this context, an applicant sends a task and a crowd of workers solves the problem. The main feature of this framework is the privacy of workers and transaction fees on every action. In CrowdBC there are four roles:

- Requester
- Workers
- Miners
- CrowdBc Client

5.1.1 ROLES

Requesters are the users who post a task that requires a solution. When a task is posted requester must pay in advance a deposit, a deposit is the sum of reward (the sum paid to the workers) and a penalty. The crowd is formed by **workers**, they have the skills and compete to solve the requester's task and achieve the reward, linked with the solution of the task. Workers are characterised by a series of tuples formed by reputation value and category (i.e. number of times which solved this category and number of high evaluation), we will see that it will be needed to choose a type of worker before a requester post a task. Also, the workers pay in advance a deposit to participate in a task that is equal to the penalty paid by requester. **CrowdBC Client** is like the EVM on Ethereum, then this client runs locally on every account and it is used to reach the agreement between workers and requester. **Miners** provide the block through mining process the blocks and they receive in return the reward and transaction fees. Both requester and workers can be miners. To keep track of the users' activities carried out in the platform there are three different keys private key, public key and address.

5.1.2 KEY FEATURES

The most important challenge for the designers of CrowdBC was to create a decentralized framework starting from a centralized system, i.e. crowdsourcing. To achieve this result CrowdBC is developed on three fundamental characteristics:

- Underlying Blockchain
- State Machine Construction.
- Three Layers Architecture

Underlying in CrowdBC is an arbitrary "program", i.e. smart contract. The "program" encapsulates the logic of a task respecting the values decided by the requester and it contains every information about the task (workers, requester, deposit, etc.). Every machine of the framework has a compiler to compile the "program". As Ethereum, CrowdBc has a **state machine** to keep track of the state of the task, the states are Pending, Unclaimed, Claimed, Evaluating, Canceled, Completed. Every time a state is updated, a transaction is generated and then the state is stored in the transaction. The state is finally updated on the blockchain when the transaction is included in a block. The **three layers** of CrowdBC are:

- Application Layer has three modules: **user manage** that it manages the user information; **program compiler** convert the “program” into the executable language of the blockchain layer; then, both request and workers can post a task through the **task manager**, the last module. The types of task are: posting, receiving, solution submission and reward assignment. When a task is submitted by a requester, it has associated with a ”program”.

- Blockchain Layer that deals with providing a consensus protocol for the ”program” and running the state machine. After the program is compiled it is included in a transaction, so it is checked by the miners and added to the blockchain. So the state machine takes an input of the application layer and then it triggers task state changes in the blockchain. only the metadata (owner, timestamp, pointer, data hash value, etc.) are saved on this layer. The hash value is saved on the blockchain layer to make blockchain-data-structure immutable and to prevent an attack to malicious nodes.

- Storage Layer is used to saved data of the task and the solution. Every data are signed with her/his private key by the owner. When a worker posts a solution he/she encrypt their data with requester public key, this means that only requester can decrypt the solution (avoiding malicious intent by other nodes). This layer is used to decrease the amount of data size stored on the blockchain layer, and then on every node. Each block in the blockchain layer has a query string that it is pointed on a data to the storage layer.

5.1.3 MAIN STEPS

CrowdBC Client works following six steps:

1. requester or worker registers on CrowdBC, CrowdBC Client generates a ”program” with user information and a public key pair;

2. CrowdBC Client update the "program", like a transaction the "program" is sent in broadcast and it is validated by the miners;

3. requester post a task and he/she pays the deposit to the blockchain. Furthermore, the requester sets the minimum qualification that a worker must have in order to participate in the task;

4. register workers receive the task from CrowdBC client after they choose to participate they must pay the deposit with coins and reputation value;

5. before the deadline, set by the requester, a worker can submit his solution. Solution address is encrypted with the public key by the worker and stored in the blockchain, while solution description is stored in the storage layer. Then the requester can decrypt the worker's solution with his/her private key.

6. In this step, the transaction is published in the blockchain and reward is paid to the worker. Rewards will be high as much as the evaluation of the solution.

5.1.4 CONTRACTS

As shown in fig8, CrowdBC is a sequence of programs, or smart contracts, that contain algorithms. CrowdBC has three types of smart contracts: User Register Contract (URC), User Summary Contract (USC), Requester Worker Relationship Contract (RWRC). Every time that a contract is updated or created a transaction is sent, and then a fee is paid to the miners.

URC is a contract that stores the initial information about a user and assigning an address and a pair of keys, public and private one. This contract does not store personal information of the user but in general

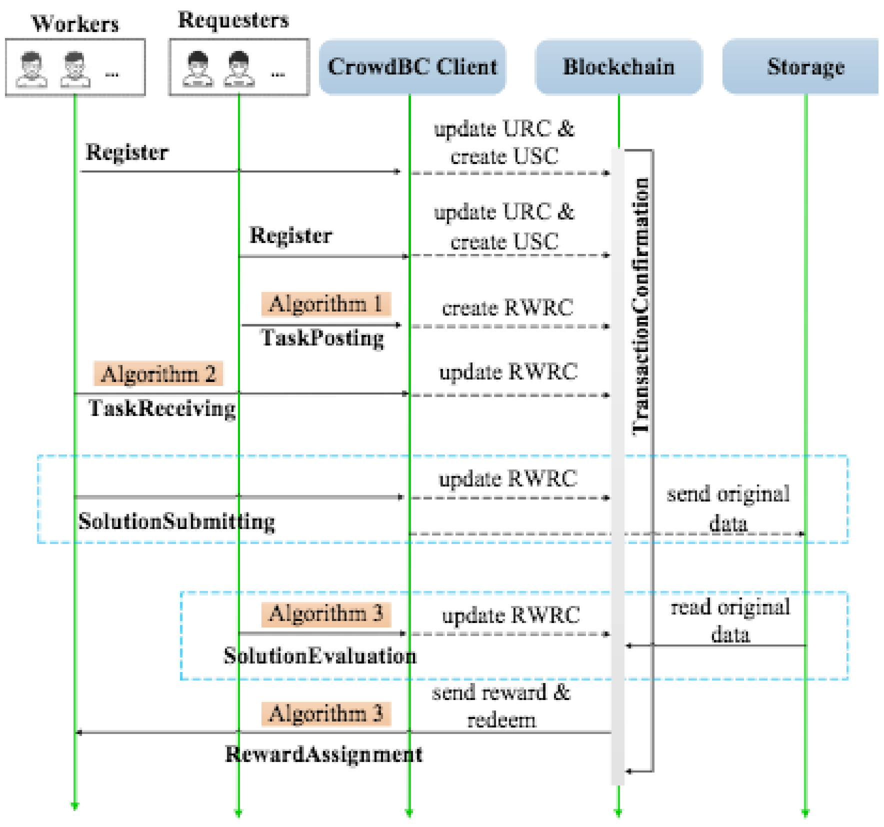

Figure 8: The process of crowdsourcing in CrowdBC and smart contract updating [47]

USC in order to have a greater chance of participating in a task, CrowdBC store personal static information and evaluation. The statics information are the profile, reputation, task general description and activity. **Profile** stores more personal information than URC, for example, skills, profession, etc. So, if a user registers with a true identity in URC, he can authenticate himself with his public key and keep his profile updated. **Reputation** is initialized when the profile is created and it is updated every time that a task is completed. Low level of reputation limits the opportunity to be chosen by requesters. **Task general description** is a general static on the tasks that have been completed and their evaluation. USC stores information about pending and bidding a number of **activities**. Activity is updated every time a task is completed. The task statuses corresponding to the USC can be Pending or Unclaimed.

RWRC is the main contract of CrowdBC because it manages the relationship between requester and workers. RWRC contains the main algorithms of the CrowdBC logic, which are: task posting, task receiving, solution evaluation, and reward assignment. As mentioned above in 5.1.3 in the third step, the requester post a task (T) with some information, which are: task description; public key of requester; solutionEvaluate(·) algorithm [12]; tuple of reputation and category for workers (chosen by requester); deposit; task deadline; task confirmation time. So, workers receive the task through checkWorkerQualification(·)[13] if they respect minimum reputation and reliability value. The address of these workers is stored in Wpool where the pool represents the number of required workers. Each worker can validate the request using the public key of the requester in the task, the requester encrypts the task with its private key. The deposit is used to avoid malicious peer, the deposit is managed by timed-locked deposit protocol for Crowdsourcing (RR Refund or reward). Requester deposit is list of elements: session-id, task id, task deadline, task confirmation time, public key requester, tuple of reputation and category for workers, pool, requester deposit and solutionEvaluate(·); while Workers deposit is a list of session-id, task id, task deadline, task confirmation time, public key requester, public key worker, coins deposit worker, reputation deposit worker and redeem(·)[14].

[12]"The reward assignment and reliability value updating rely on the output of solutionEvaluate(·), a trustful truth discovery algorithm. For simplicity, the output will be "H" (high level of effort) or "L" (low level of effort)" (low level of effort)".(2018, CrowdBC)

[13]validation function to "check if worker's reputation and reliability value satisfy the minimum limited."[47]

[14]redeem(·)= verify(·)(public key requester) and (verify(·)(public key worker) or solutionEvaluation(·))

Malicious requesters[15]or workers[16] will not have refund the deposit and the others participant it will be compensated with malicious's deposit.

The timed-locked deposit protocol follows the following steps: **Requester Deposit** is created and if it validates the task is sent to the workers, the requester can access to the deposit only after the task deadline; **Worker Deposit** is subtracted when a worker (with a reputation higher than minimum reputation) is selected for the task, the deposit is equal to the deposit of requester; **Claim**, worker follows the fifth step described in 5.1.3 and then she/he requires to redeem the reward through claims($\cdot$): session-id, transaction , public key worker , public key requester ,deposit coins requester, deposit reputation worker, Sign of private key requester ,Sign of private key worker. **Reward** can be required both workers and requester; they must control that the deposits are not deleted and that they get enough confirmations in blockchain for security. So, one of them sends in broadcast the transaction that redeems the deposit and if solutionEvaluate($\cdot$)= H then sending coins deposit of requester plus reputation workers, while if solutionEvaluate($\cdot$)=L

Reputation Management The whole CrowdBC system is based on reputation as workers are chosen based on their past reputation. Reputation in CrowdBC is represented by an integer, β, that it is the amount of evaluation for completed tasks, its range is from 0 to β_{max}. While h represents the average reputation of the whole workers. When a task is submitted and confirmed by miners the solutionEvaluate() applies the reputation β to the worker. Based on miners evaluation Crowdbc distinguishes two types of efforts:

[15]Malicious requesters make low ratings for each task response, in this way they do not lose the deposit and stole useful solutions

[16]Malicious workers are involved in free-riding attack, i.e. "create a forked chain if they receive low-level evaluation, not submit solutions on time and posting a task by themselves"

- L=low effort;
- H=High effort;

So, based on the work “Reputation-based incentive protocols in crowdsourcing applications” by Yu Zhang there are four types through which valuation can be expressed:

- if the effort is L and $\beta \geq h$, the reputation become equals to zero (β=0);
- if the effort is L and $\beta \geq h+1$, the reputation decrease $\beta - 1$
- if the effort is H and $\beta \geq h$, the reputation increase $\beta + 1$ or it remains β_{max} if it is the max value
- if $\beta < h+1$, the reputation increase $\beta + 1$

The workers who have a reputation under h can achieve only to a few simple tasks. This situation will continue until the worker's reputation is reached h.

5.2 BACKFEED

BackFeed is a platform of sharing economy[48] that linked the fields peer-production with the sharing economy, in the blockchain technology. The main purpose of Backfeed is to collect a different Dapp platforms in only one big community. The most important feature of BackFedd are:

- production of value;
- record of value;
- actualisation of value.

The result that Backfeed wants to the bringing of the society models from the industrial models to the information society. To achieve this result backfeed using the blockchain technology. Blockchain platform, offer to backfeed, the opportunity to give a common space to share, evaluate and achieve contributions purpose of single Dapp online community in 'sharing economy' system. So, online communities, to avoid failure, often transform their governance in hierarchical structure and using a more market oriented system, Backfeed born to help these communities in a sharing economy system. A whole decentralized approach helps not only the community but every contributor of the system.

5.2.1 TECHNICAL ELEMENTS

The main technical elements used by Backfeed are:

- A token
- A reputation score
- A protocol to for distribution of reputation and token

So, Backfeed implement a **token protocol**, i.e. when a contribute to a project is considered positively by the whole community the decentralized collaboration system releases a token (the token is not released to the project but to the contributor). This token can be used as form of currency to exchange value on the market. The tokens can be used to boost other projects on the Backfeed platform. To evaluate when a contributor is positive Backfeed uses a **reputation system**, each contribution to a specific project can be evaluate with a different system of evaluation, then backfeed uses a evaluation set based on the considered Dapp can be used (from 1 to 5 or 0/1 system). The most important feauture of Backfeed is the **value function**, i.e. the protocol that set the distribution of token based on reputation system. The value function is a linear function that as the value of the distribution increases, the value of the tokens released increases. For example, gives a evaluation set E

$$\eta = 1, 2, 3,$$

the value function v is

$$v = \eta \longrightarrow R$$

where where v(1)= 0; v(2)= 20; v(3)= 50 tokens. Moreover, the amount of tokens issued to the contributor depends on the median value of all the assessments weighted in the community where the contribute was released, i.e. the tokens will be released only when 50% of the overall reputation of the considered community will have voted

5.2.2 SYSTEM MODEL

The token distribution is used to incentivize contributions, but the Backfeed protocol is, above all, a public reputation system that manages the rules on which the reputation is made and distributed within a given community. The reputation can be gained by a user and it does not transfer to other users. In Backfeed there are two different types of user:

- **Contributors**: Anyone can make a contribution that can be evaluated both positively and negatively within the community where it is released. When the median reaches a positive value, new reputation is issued and distributed to the corresponding contributor
- **Evaluators**: Anyone can makes evaluations of others' contributions within a community. Every evaluation has a cost in term of reputation.

DYNAMIC REPUTATION SYSTEM Backfeed implements a dynamic reputation system, it consists of two complementary element. The first one is related to reputation paid when a evaluation is made, greater is the number of evaluation issued, lower will be the cost of reputation paid for the next evaluation issued. The second one is related to the moment in which the evaluation is issued,

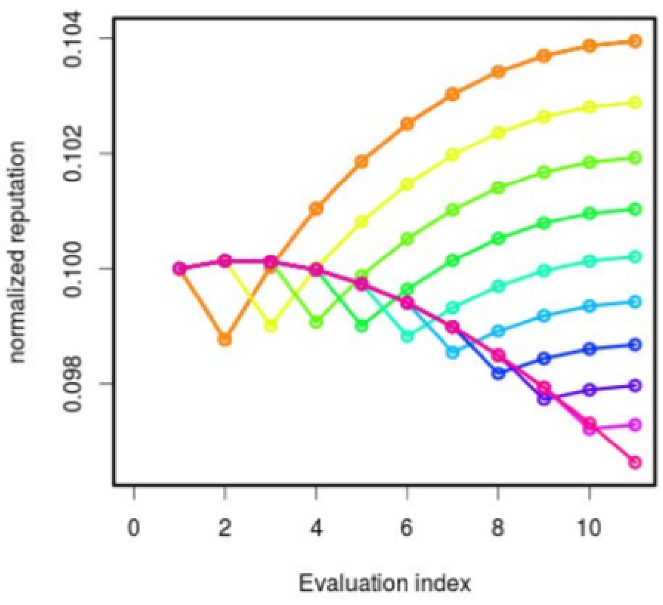

Figure 9: Illustration of the reputation flow for a simple scenario in which 10 users evaluate the same contribution consecutively by the same vote. [48]

for example takes ten evaluations with the same value of a contribution the initial contributor gained a percentage of each contribution that has the same value

6 BLOCKART

The platform Blockart is a peer production platform for the artist. In Blockart, there are three main rules: initiator, authors/voter/moderator and readers. The initiator posts a draft of own project and a pool of "select" authors help him/her to complete the project. Once the author has posted his solution, it will be voted by the voters and upon reaching a certain threshold of votes the author will become the owner of a part of the final project, mediators have the rule to manage the content of authors. The figure of the Blockart initiators is similar to the alpha artists, then initiator and alpha-artists are stackable. Up to this point, Blockart can be compared to a crowdsourcing platform because, like in CrowdBc, there is a requester who posts a task and there are workers who submit their solution to the task. So, while in crowdsourcing platforms the workers are executor in peer production workers (or authors) are the creator because they participate in the creative process not only as executor. As mentioned above, there is a form of crowdsourcing very close to the peer production, in fact in the crowdsourcing with structural participation workers are co-designer and co-authors, they become authors. These types of platforms leave more freedom for decision-making to authors and allow to better explore the resolution possibilities of the task but the most common problem of structural degree of participation in crowdsourcing is the management of authorship of the project. Blockart starts from this open issue and it looks for a way to solve the problem of authorship in these platforms. To solve this problem, we let to see which are the motivations that lead authors to participate:

- Crowdsourcing provides rewards with extrinsic motivation (i.e. monetary) when a task is completed.

- Whereas, as mentioned above, peer production platforms is based on intrinsic motivation, such as reciprocity, reputation and transparency.

Not forgetting that Blockart presents itself as a peer production platform, Blockart borrows the concept of prizes (extrinsic motivation) to distribute ownership of the final project, but at the same time to keep track of the reputation (intrinsic motivation) of the authors

6.1 SYSTEM MODEL

We will see how the main concept which characterizing Blockart system model

6.1.1 ROLES

In Blockart there are three different roles:

- Alpha Artist(AA) post a draft with a description of the main concept. So, the following step is the production. Therefore, before the start of the production, AA check which is the minimum reputation level to participate, the types of categories needed, a list of pre-assigned moderator and if the production is public or private. AA can suggest modifies during the production.
- Author/Moderator/Voter(AMV) is the fondamental figure in Blockart scheme. Every user who joins the production becomes AMV. AMV deals with proposing, voting and moderation the contents of a production. Thus, AMV roles are:
  - Author, AMV becomes author, in a peer production, when post a content in form of discussion or comment a content. So, authors have the fondamental role of post own idea to grow the project
  - Moderator, AMV becomes moderator only if the user reputation is above a certain threshold or if it is pre-assigned by AA. For each production there is a pool of moderator users.

    So, moderators have the rule of filter, reject off-topics and mediate contents of authors.

  - Voter, AMV becomes voter when he/she votes a content of discussion. Every AMV user has a set of limit number of vote for each production. So, voter have the rule of vote the contents of authors and then who will are the owner of the final project.

- The reader can read the production discussion if the production is public and it can participate in the production, while if the production is private can ask to participate in AA and then becomes an AVM. In both cases reader must have the minimum reputation level to participate in the production

- Miner in Blockart it has the role to mine blocks and adding valid transactions to the block.

Each user is characterized by a series of parameters: key pair (public and private), category, number of participation in productions, reputation, name of projects that he/she participated (with their respective percentages).

6.1.2 THREATS

All the Blockart figures could benefit from performing acts that threaten the ecosystem of the whole platform. Therefore, now we will define what are the potential threats deriving from harmful behaviour in Blockart:

MALICIOUS ALPHA ARTIST The malicious AA publishes a draft with a list of pre-assigned moderators, this can be dangerous because the moderators could be motivated by a malicious intent according to AA intent.

MALICIOUS AUTHORS They could try to get a part of the final project without making an efficient effort, moreover they could create a fork, from the main idea, if him/her idea does not insert in the final project. Another important malicious intent could post off-topics content by an author. As for reputation, they may deny that their idea is of low quality, otherwise could improve their reputation by posting a task and solving it by themselves.

MALICIOUS VOTER According to Audun Jøsang listed a number of items that could influence voters and then the motivation to became malicious:

- Low incentives to include a vote, a voter does not vote for comment for several reasons. A system that does not encourage the vote can trigger free-riding, that is to leave the task assigned to others to steal ideas. (deposit and incentives)
- Exchange vote, a voter could use the vote as a bargaining chip to receive a reward (anonymous vote)
- Unfair vote, it is difficult to judge a vote, especially when they are on a subjective basis (Endogenous Discounting and Exogenous Discounting of Unfair Ratings)
- Identities exchange, if a user at once a great loss of credibility, or reputation, could decide to create a new account with another identity. (agents used the resulting reputation score when negotiating future transactions).
- Change of value over time, the reputation with the time could change is important to maintain

MALICIOUS MODERATOR As with voters, moderators could also use the same motivations for malicious intent:

- Low incentives to include a moderation, a moderator must be encouraged in his moderation because avoiding moderation the system can be vulnerable to free-riding attack

- Exchange moderation, a moderator could use his/her moderation receive favours in exchange

- Unfair moderation, as for voting also moderation is difficult to judge, especially if it is a subjective opinion. Moreover, the role of moderator has a leading role in the management of the contents of other authors, therefore the malicious subjective choices influence the community more than the choice done with an unfair vote.

MALICIOUS READER can steal the production idea without to participate in peer production

6.2 BLOCKART: PEER PRODUCTION FRAMEWORK

Now, based on the analysis done so far we will formalize a peer production framework, in the specific case for artists, decentralized called Blockart.

6.2.1 LAYERS

According to Blockstack [49], Blockart framework is developed on three different layers:

- application layer: user and production manager, and decode the smart contract for blockchain layer

- blockchain layer: providing a consensus protocol, running the state machine and storing the blockchain data value (for example state variables)

- storage layer: store the contents of the production and sign them with the owner's private key

Our analysis will focus on the fundamental aspects of the application layer in Blockart. For the moment, Blockart relies on the possibility offered by Ethereum, i.e. to develop the application layer through a contract account. Thus, the blockchain level and the execution of the state machine (which links the application layer to the blockchain one) is entrusted to Ethereum and to the EVM. Therefore, the Blockchain layer and Storage layer could be implemented in the future.

BLOCKCHAIN LAYER The blockchain layer is the reason by which the different blockchain platforms are distinguished, two important features of blockchain layer are: the implementation of a consensus protocol and the management of the different states in which the system is located. Future Blockart improvements will involve the blockchain layer: firstly, As mentioned above, Ethereum uses GHOST as consensus protocol, then one of the most important future challenges for Blockart will be that to able to implement an own consensus protocol; secondly, every time a transaction is passed from the application level, the transaction is verified, added to the block and then the system status is updated. In Blockart, the change of the status is of the core of the production, i.e. the contents posting by authors. Therefore, as shown in figure 10 the status will be: pending, moderated, cancelled, voting, assigned, revoked.

STORAGE LAYER Blockart will use IPFS (InterPlanetary File System). This framework gives the opportunity to store content-addressable data, exploiting a peer-to-peer method of storing and sharing hypermedia in a distributed file system. So, for some data in the blockchain layer will be a hash reference pointing to the date content in the storage layer.

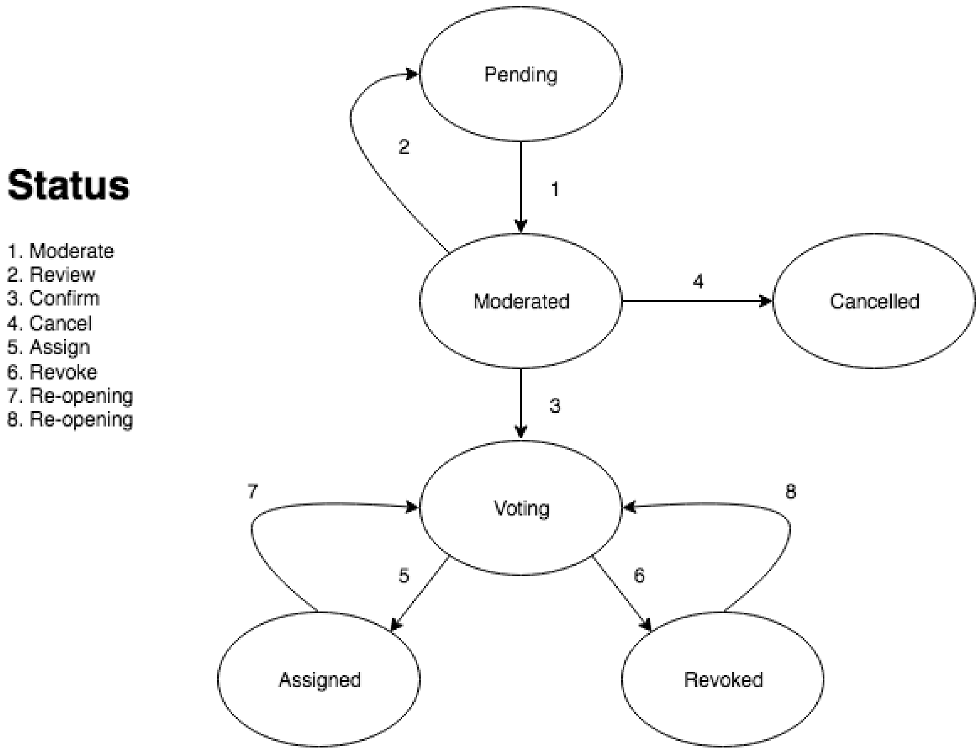

Figure 10: State machine model for a content

6.3 MAIN ALGORITHMS

We will now proceed to a step-by-step analysis of the main algorithms of the Blockart processes on the application layer. The storage layer will be used to store the content of the production to avoid to overload the blockchain layer with (since the blockchain layer must be store on all the nodes of the platform)

6.3.1 STEP BY STEP

So, a summarizing of the main steps in Bloackart is:

1. Users register on Blockart, they are added on as nodes on the platform and all of their fields are set to default values

2. AAs call a contract to open a production and, then, they set the initial requirements

3. Reader can decide, if she/he has a sufficient reputation, to partic-

ipate to the production. If the production is private participation is subject to the approval of the alpha artist

4. Reader became Author and Voter when he/she pays the "deposit", moreover a reader can became Moderator if meets the requirements.

5. An author post a content, then the "pool" of moderators confirm the quality of the content

6. In the follow step the peers who participate to the production vote the author's content, at this point the other authors can decide to start a new content starting from this they are evaluating

7. When a number of votes is reached, the deposit is refunded. While based on the quality of the votes it can increase/decrease the reputation of the authors and the assignment of a part of the property to the final project.

6.3.2 REPUTATION SYSTEM

The main purposes of the reputation system are to avoid malicious intent by voters and to maintain high-quality contents post by authors. Each user has two different parameters that are related to the reliability of the user on the platform:

- Trust Ratings
- Reputation

whereas trust in a node is achieved through by blockchain technology, Blockart develops an algorithm to track reputation over time. So, the system that we will be describing how to build and update and represent the reputation, here we will propose a Bayesian approach [17]. Assuming

[17]It is used to calculate the probability of a random variable given an observation: $P(A_i|B) = \frac{P(A|B_i)P(B_i)}{\sum_{i=1}^{n} P(A|B_i)P(B_i)}$

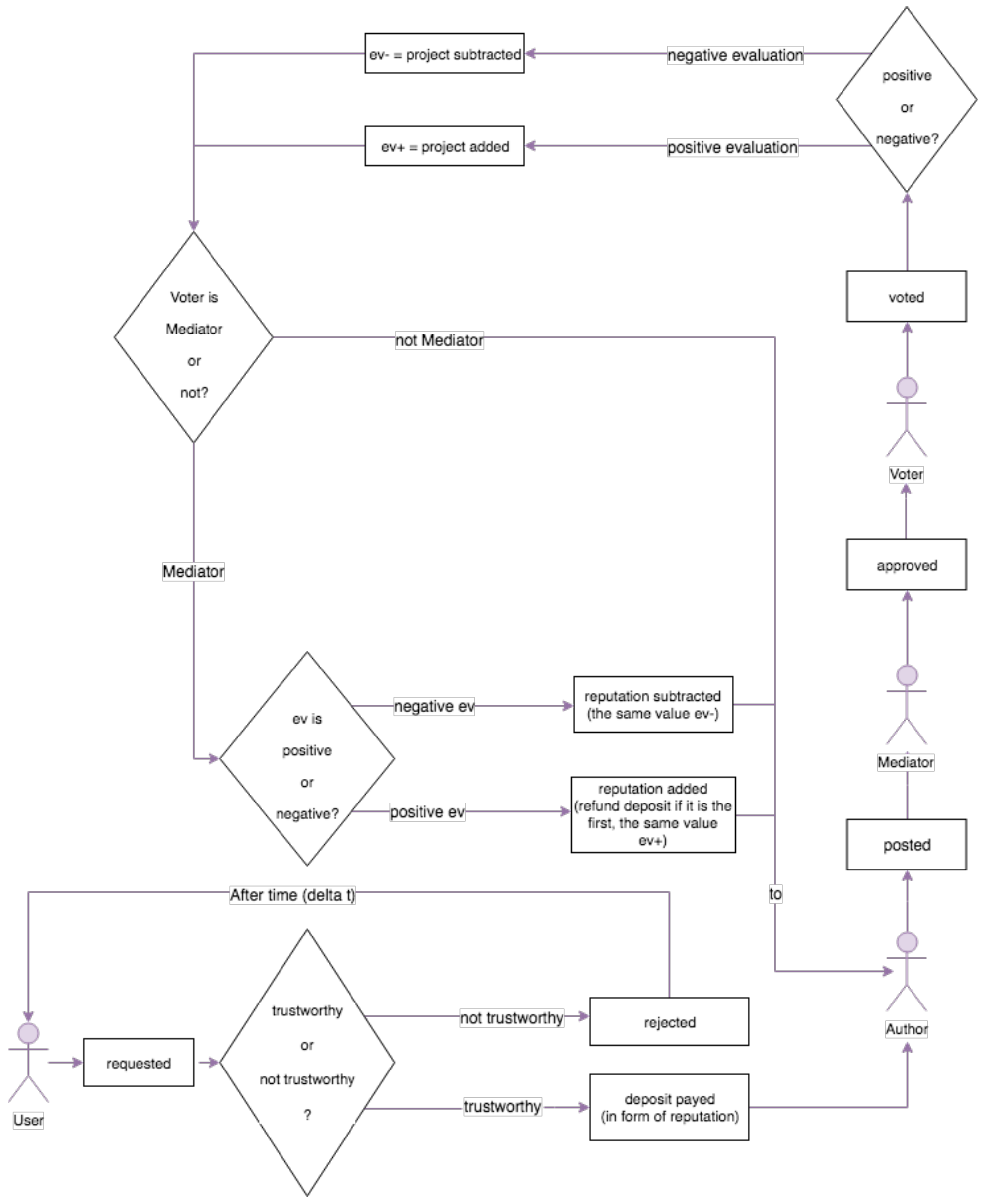

Figure 11: main algorithms working

that the malicious intent of a node is unknown, θ, given a series of observations (contents quality post by author) we will be able to predict the probability that a user may act in a malevolent manner, i.e. the probability that he/she could become a malicious voter. The initial distribution of the probability of θ is called "prior", then this value is updated each time another user makes an evaluation, whit a bad or good evaluation. So, the distribution is a binomial function called Beta(α, β)[18]. In the beginning, Beta, "prior", is chosen arbitrarily for all users and is equals to (1,1). When a new reputation is issued, the previous one is updated. So starting from the previous currency, Beta (α, β), there are two different cases:

$$Beta(\alpha', \beta') = \begin{cases} Beta(u * \alpha + 1, u * \beta) \text{ if } \text{ Negative evaluation} \\ Beta(u * \alpha, u * \beta + 1) \text{ if } \text{ Positive evaluation} \end{cases}$$

where $u = 1 - \frac{1}{m}$, m represents the number which indicates the number of evaluations that have a stationary behaviour, i.e. the number of evaluations beyond which the reputations remain the same even after endless evaluations. Moreover, after a period of inactivity, δt, the user's reputation became Beta$(u * \alpha, u * \beta)$. The expected value of Beta is easy to calculate:

$$E|Beta(\alpha, \beta)| = \frac{\alpha}{\alpha + \beta}$$

So, in Blockart a user can be classified as:

- trustworthy if $E|Beta(\alpha, \beta)| \leq t$
- not trustworthy if $E|Beta(\alpha, \beta)| > t$

where t is the minimum threshold for participating in a production, and in Blockart is equal to 0,75. Therefore, if a user exceeds the threshold, he will have to wait for a period of time equal to δt to participate in the

[18]The Beta function is the conjugate prior for binomial likelihood and thus the posterior density is also Beta

production. Moreover, the number of stationary behaviour, m, is equal to 10. If a user remains trustworthy for more than ten productions he receives a positive vote, this is done to encourage long-time users.

6.3.3 DEPOSIT

To join the production every AMV must pay a deposit, it corresponds a part of his/her reputation. The deposit consists of attribution of negative evaluation (6.3.2). So, is calculated the reputation $E|Beta(alpha, beta)|$ and if this value is above the threshold t, AVM will not participate in the production. The deposit will be refunded with a positive reputation vote to the AMV when she/he reaches a minimum number of votes after entering a content.

6.3.4 MODERATION RULES

At the beginning, AA post a draft and he/she invites a group of the initial moderator, maximum 4 (note all of the moderators must be only the authors which participate to the production and with a reputation above the threshold). For each moderator chosen by the AA, a random moderator must be chosen among the authors. Moreover, all of the authors which to the production are moderators. If there are not at least two moderators the production does not start. A moderator can moderate only the contents of other authors. The roles of the moderator are different, a moderator may:

- suggest editing of author's content
- suggest deleting an off-topic or low-quality content
- re-organizing the overall argument map of contents
- offering suggestions

Except the last two points, the points are applied only when 50% plus one (except the content's author receiving the suggestion) of the moderators have approved that suggestion, to avoid unfair moderation. All of the moderation is private. To incentive moderation, any moderation that reaches 80% approval among all moderators receive a reward in the form of reputation, this reward is calculated based on the numbers of the participant to the production. To avoid that mediators can agree on which contents to mediate, in turn, each mediator has a specific time frame to judge the contents posted by authors.

6.3.5 VOTING RULES

Each voter has a maximum number of vote. A vote can be negative or positive. Each vote is anonymous. Every time that a voter performs a vote must release a comment with the motivation of the vote (also this anonymous). So, other users can report the comment, for example as inappropriate, and then moderators can delete the vote and assign a negative evaluation (6.3.2) to the reputation of the voter. Instead, a mediator can assign a positive evaluation (6.3.2) to a worthy comment and suggest to the voter how that comment could become a content of the production.

DISTRIBUTION OF REPUTATION For the reputation system, are considered only the votes of mediators. For every positive evaluation of the content, by a mediator, the author receives a positive evaluation (6.3.2), while for every negative vote, by a mediator, the author receives a negative evaluation (6.3.2).

DISTRIBUTION OF THE FINAL PROJECT Taken an author of its percentage compared to the final project will be:

$$\begin{cases} 100\frac{\sum_{i=1}^{n} V^{+} - \sum_{i=1}^{n} V^{-}}{\sum_{i=1}^{n} n^{+}} & \text{if} \quad \sum_{i=1}^{n} V^{+} >= \sum_{i=1}^{n} V^{-} \\ 0 & \text{other case} \end{cases}$$

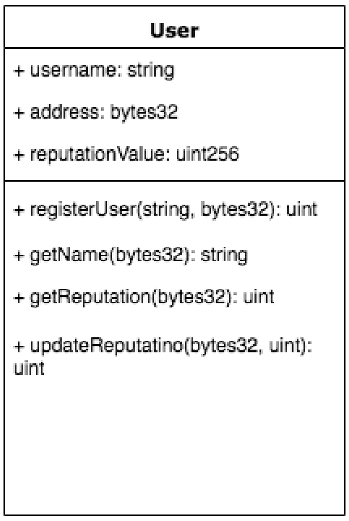
User
+ username: string
+ address: bytes32
+ reputationValue: uint256
+ registerUser(string, bytes32): uint
+ getName(bytes32): string
+ getReputation(bytes32): uint
+ updateReputatino(bytes32, uint): uint

Figure 12: user manager

where n is the total number of votes that authors received for the post of every content in the production, n^+ is the total number of positive vote that authors received for the every content post in the production, V^+ is number of positive evaluation received to the author and V^- is number of negative evaluation received to the author.

6.3.6 CONTRACTS

To better understand the flow that a user follows in Blockart, in this section, starting from fig.11 we will analyze how the contracts implement the main algorithm. Blockart implements four different contracts UserManager, DiscussionManager, ParticipantManager and Comment-Manager

USER MANAGER is the contract that deals with manage user information. The main functions are to register a new user and to restore the reputation and username of registered users As shown in the image fig.12 user is composed of three fields: username, address and reputationValue. The first function, registerUser, is used to register a new user to Bloackart and it requires the string username and address, of the registering user, in the form of bytes32. When the user is initialized the function registerUser set reputation to 0. getName and getReputation serve to retrieve information respectively about username and reputation of the caller user

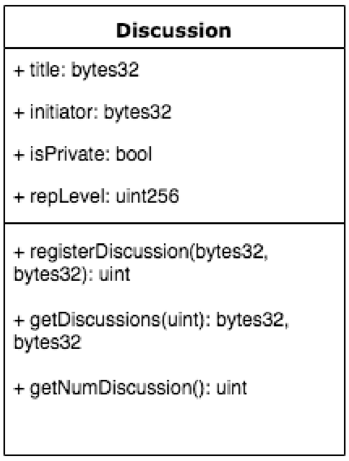
Discussion
+ title: bytes32
+ initiator: bytes32
+ isPrivate: bool
+ repLevel: uint256
+ registerDiscussion(bytes32, bytes32): uint
+ getDiscussions(uint): bytes32, bytes32
+ getNumDiscussion(): uint

Figure 13: discussion manager

DISCUSSION MANAGER is the contract that deals with manage discussion information. After that, a user is registered can start to post a new discussion through this contract. As shown in fig.13 discussion is composed by title, address of initiator if the conversation is private or public and the minimum reputation level for the user that will participate to the production to become the mediator of the discussion. When a user wants to open a new discussion he/she must call the class registerDiscussion, this class requires two different bytes32 one the title of discussion and one the initiator of the discussion (note for this beta version repLevel is set to 0 and isPrivate is set to false). The other two functions are getters, the first one getNumDiscussion return the total number of discussion stored in BlockArt and the other, getDiscussion , that return title and initiator of the discussion given the number of the corresponding key of the map of discussions

COMMENT MANAGER is the contract that deals with manage comment information, this contract is called after that the user has initialized a discussion. Comment is characterized by six different fields: author of the comment, title of the comment, number of positive evaluation received by the comment, number of negative evaluation received by the contract and the title of the discussion associated to this comment. The first function is registerComment that requires three different bytes32 (comment author, comment title and discussion title) and a string with the content of the discussion, this function can be called only when a discussion has been initialized and a user is interacting with the discussion front-end component. The function getComment gives an identifier number of the key of the map of all comments, it returns: comment author, comment title, string with content, number of positive evaluation received by the comment, number of negative evaluation received by the contract and the title of the discussion. While the other two functions are related to managing the number of the received vote of the comment. They are called, by front-end component, when a user a already post a comment and another user, voter, vote that comment, then when a user registers a vote he/she calls registerVote which requires comment author, comment title, discussion title, voter and a boolean to verify is the comment is positive or not. While getVote gives comment author, comment title and discussion title returns two number one with positive and the other one with negative evaluation.

PARTICIPANT MANAGER is the contract that deals with manage participant information. This contract is complementary to comment contract but while commentManager contract deals with managing the information of the comments of the discussion ParticipantManager contract deals with managing the participants of the discussion, the fields of this class are: participant address, discussion title, number of remaining vote, boolean indicates if the participant is mediator or not,

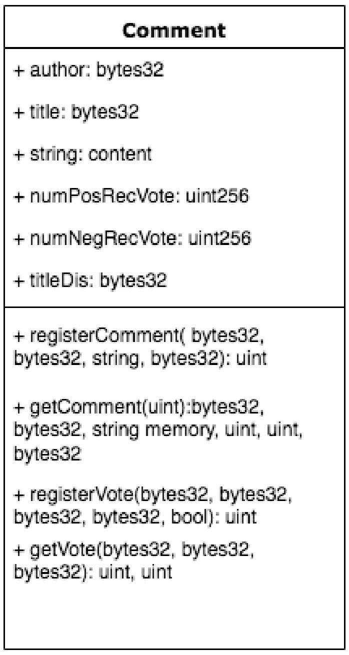
Comment
+ author: bytes32
+ title: bytes32
+ string: content
+ numPosRecVote: uint256
+ numNegRecVote: uint256
+ titleDis: bytes32
+ registerComment(bytes32, bytes32, string, bytes32): uint
+ getComment(uint):bytes32, bytes32, string memory, uint, uint, bytes32
+ registerVote(bytes32, bytes32, bytes32, bytes32, bool): uint
+ getVote(bytes32, bytes32, bytes32): uint, uint

Figure 14: comment manager

number of positive votes received, number of negative votes received and the owned percentage respect with the discussion. When a user comes in a discussion of BlockArt the interaction active the registerParticpant function, that function allows to register a user like a participant, then this function requires the title of discussions and the address of the user that ask to become a participant (in the final project this requirement is denied if the reputation is not enough to participate to the discussion). reigisterParticipant function set the value of remainder vote to five and the other fields to zero. The other two important functions of this class are givesVote, getParticpantPergentage and getReputation. givesVote function runs when a user gives a vote to the comment, then when a user gives a vote to comment on two different contracts are called this one and CommentManager contract. givesVote assigns the vote to the comment's owner and it requires the address of the author of the content, a boolean to understand if the vote is positive or negative, the address of the voter and the title of the discussion. While getParticpantPercantage contains the algorithm describe above in 6.3.5 paragraph and require the

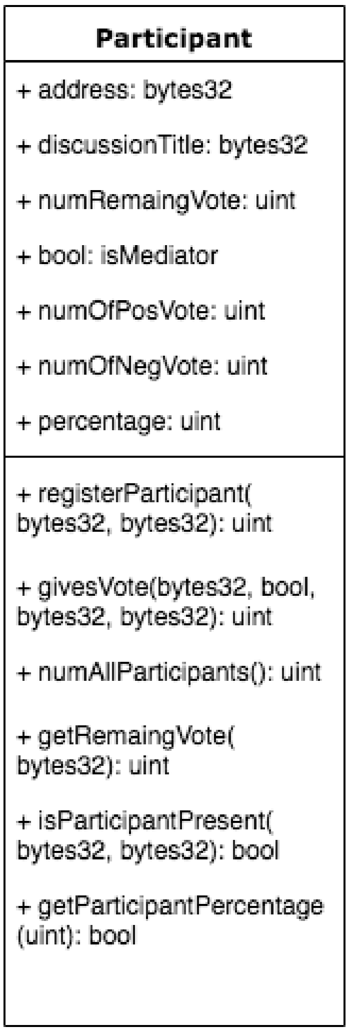
Participant
+ address: bytes32
+ discussionTitle: bytes32
+ numRemaingVote: uint
+ bool: isMediator
+ numOfPosVote: uint
+ numOfNegVote: uint
+ percentage: uint
+ registerParticipant(bytes32, bytes32): uint
+ givesVote(bytes32, bool, bytes32, bytes32): uint
+ numAllParticipants(): uint
+ getRemaingVote(bytes32): uint
+ isParticipantPresent(bytes32, bytes32): bool
+ getParticipantPercentage (uint): bool

Figure 15: participant manager

id key of the map with participant require address of user participant and discussion title, and return the number with percentage compared to the discussion. getReputaion contains the algorithm describe above in 6.3.5, it requires the address and returns the number of the reputation of the user.

6.4 EVALUATION RESULTS AND ANALYSIS

6.4.1 SYSTEM DESIGN

The development of BlockArt is focused on designing a secure and decentralized system. Thus, an application based on angular, web3.js and truffle of a BlockArt prototype was implemented, the prototype was tested on Ethereum's public network. BlockArt based on 4 different smart contracts: UserManager, DiscussionManger, CommentMnanager and ParticipantManager. Usermanager store the information about user reputation and authentication, DiscussionManager keeps track of the information on the discussion, CommentManager store the info of the comments and ParticipantManager deals with storing the info of each user's percentages on the given discussion. BlockArt has been implemented on official Ethereum public test network Truffle with a programming language that includes solidity, typescript, CSS and HTML with around 60,000 lines of code. BlockArt interacts with Ethereum based on web3js, a library for javascript applications on the Ethereum network. It is been deployed and migrated Blockart's contracts with Truffle, then the combined use of Ganache (a blockchain tool for Ethereum development that gives the opportunity to create a local ethereum network with ten fake accounts, each account with 100 Ether) and Metamask (a browser plugin to interact with dapp in a browser) it possible to interact with Blockart dapp.

6.4.2 RESULT EVALUATION

Migrations give the opportunity to develop the contract to the blockchain. Truffle migration develops from contract to javascript, then these files can interact with web3js. when migration is developed its returns the result, these results can be used to analyze the efficiency of the contract in term of gas costs. So, we will see how the result of migration, with truffle, of every contract of BlockArt:

Listing 9: user manager evaluation

```

 Replacing 'UserManager'
-----------------------
> transaction hash:
0xe61e13dcd04051d8cf71b7c48720a8843aa154fe77764fc6d5630af225a5ba25
> Blocks: 0          Seconds: 0
> contract address:
    0xB8f038948ec46903caAcBAe7fB18b592C4E5065A
> block number:      1765
> gas used:          1084428
> gas price:         20 gwei
> value sent:        0 ETH
> total cost:        0.02168856 ETH
-------------------------------------
> Total cost:        0.02168856 ETH

Replacing 'DiscussionManager'
-----------------------------
> transaction hash:
0x5649d75265f2bd9ae49a05a3e728c2445aec252d2d9d599a5b388cbda6a25954
> Blocks: 0          Seconds: 0
> contract address:
    0xfea06e25d0f0EFf52B24fdf693CcFcCC9a9E0968
```

```
> block number:      1767
> gas used:          468067
> gas price:         20 gwei
> value sent:        0 ETH
> total cost:        0.00936134 ETH
-------------------------------------
> Total cost:        0.00936134 ETH


Replacing 'ParticipantManager'
------------------------------
> transaction hash:
0x78f1312e28ccbd5a71bc7b52a70f259b0f9ece86f8899e33007c6b9fe3a77885
> Blocks: 0          Seconds: 0
> contract address:
    0x5842a55015FfA01A572aaa616c9b963853A229cC
> block number:      1769
> gas used:          2681788
> gas price:         20 gwei
> value sent:        0 ETH
> total cost:        0.05363576 ETH
-------------------------------------
> Total cost:        0.05363576 ETH



Replacing 'CommentManager'
--------------------------
> transaction hash:
0x076c70c600cb2c6d9d70b9dbca60ae1cfb16d8a6cd6bf46aa4f79a0deab38f12
> Blocks: 0          Seconds: 0
> contract address:
    0x57f02E609896Cf431e49DB911887981cCf785964
> block number:      1771
> gas used:          2047169
```

```
   > gas price:          20 gwei
   > value sent:         0 ETH
   > total cost:         0.04094338 ETH
   -------------------------------------
   > Total cost:         0.04094338 ETH


Summary
=======
> Total deployments: 5
> Final cost:        0.1308569 ETH
```

In Blockart, the average transaction fee to deploy a contract is 0.026 ETH. According to AMT reward policy, about 4,42$ is paid for each contract. It's worth noting that, as specified on ETH market price [50], 1 ETH price is about 170 $, the cost is not acceptable and unpractical in a real environment. Furthermore, we can see that the cost to develop all the smart contracts is around 0.13 ETH equal to 22.66$.

7 CONCLUSION

BlockArt: A Blockchain dapp for collaborative artistic production. The main propose of this work has been to define a tool able to present a completely decentralized application of support for artistic production communities. So, the main propose of this work, as described above, are:

- make artists more aware of the themes of participatory art;
- create a platform proposed as a hybrid platform between peer production and a crowdsourcing one;
- realizes a fully decentralized application.

For the first propose, from the analysis carried out, the following research has deduced that being the artistic process an individual expression of the sensitivity of the artists and of communication of their ideas. Participatory art is inserted as a new way of understanding art and the artistic process. In this context, BlockArt defines a new form of artist "alpha artist", no longer the one who is the exclusive holder of the artistic process but one who has the whole artistic project in mind and supports the contributors in the artistic process.

The second propose of the work was to create a hybrid platform between peer production and a crowdsourcing one. During the analyses has been two the platforms that came closer to the application proposed by blockart

- CrowdBC;
- BackFeed.

CrowdBc is a blockchain-based decentralization framework for crowdsourcing[47]. This platform is a pure crowdsourcing platform, while BackFeed is a platform of sharing economy [48] that offers a commonplace for different dapp. As shown in the table below, there are a

Table 1: Functionality

Functionality List			
Functions	Backfeed	CrowdBC	BlockArt
Crowdsourcing	Yes	Yes	Yes
Peer-Production	Yes	No	Yes
Reputation	Yes	Yes	Yes
Property	Yes	Yes	Yes
Sharing Ec.	Yes	No	No
Token Protocol	Yes	Yes	No
Evaluation	Yes	Yes	Yes
Moderation	No	No	Yes

lot a contact points between these two platforms and BlockArt.
So, the most important feature proposed by BlockArt is the moderation system. In particular, the moderation system is a system that linked to the BlockArt reputation system gives a Blockart a competitive advantage over its competitors. The reputation system is a system that takes its cue from collaboration platforms, as collaboratorium, which drives Blockart away from a pure peer production platform but makes the entire platform more secure against malicious intent.

The third propose has been achieved thanks to the development of the application. Blockart process has been shown with a practical example, which illustrates that the blockchain-based framework is feasible.

However, some of the limitations and difficulties encountered in the Blockart design and development process must be attributed to the lack of a testing process, without a testing process the major limitations are due to understanding of:

- the ability of target users (i.e. artists) to be able to conceive a different approach with respect to the artistic process

- the flaws in a system that is a hybrid between crowdsourcing and peer production
- the real maturity of blockchain technology in term of security and efficiency
- the possibility to develop platform cheaper than the one created

This study is conducted to evaluate the implementation of a blockchain platform of collaborative artistic production. The whole idea is applicable and implementable in the proposed area. Instead, in the future, the idea can be evaluated for different areas. For example, the same idea has been proposed to I3P (incubatore imprese innovative) of Politecnico di Torino for the startcup competition. Starting from the idea described for BlockArt, a startup has been devised that helps the communication between different productive sectors in companies with multisectoral production contexts. Following the winning of the first phase of startcup competition, the incubator helped us to develop a business plan. Blockart revolves around the of blockchain technology. Although we have seen that it is possible to implement a blockchain-based application, there have been problems with transition costs. Therefore, further work could be based on the possibility that a cheaper public blockchain is needed for BlockArt. Hyperledger as a well-known blockchain fabric could be a solution.

References

[1] Piotr Konieczny. Wikis and wikipedia as a teaching tool: Five years later. First Monday, 17, 09 2012.

[2] Robert A. Cropf. Benkler, y. (2006). the wealth of networks: How social production transforms markets and freedom. new haven and london: Yale university press. 528 pp. $40.00 (papercloth). Social Science Computer Review, 26(2):259–261, 2008.

[3] Rishab Ghosh. Cooking pot markets: an economic model for the trade in free goods and services on the internet. Brazilian Electronic Journal of Economics, 1(1), 1998.

[4] Josh Lerner and Jean Tirole. Some simple economics of open source. The Journal of Industrial Economics, 50(2):197–234, 2002.

[5] Oliver Alexy and Martin Leitner. A fistful of dollars: Are financial rewards a suitable management practice for distributed models of innovation? European Management Review, 8, 10 2011.

[6] Yochai Benkler. The Wealth of Networks. Yale University Press, New Haven, Connecticut, 2006.

[7] Michel Bauwens. The political economy of peer production. Post-Autistic Economics Review, 37, 01 2005.

[8] M. Bauwens. P2p and human evolution: Peer to peer as the premise of a new mode of civilization. Ensaio, rascunho, 1, 01 2005.

[9] Michel Bauwens. p2p foundation.

[10] Vasilis Kostakis. Peer governance and wikipedia (interview with bauwens bruns).

[11] Y. Algan, Y. Benkler, Mayo Morell, and J. Hergueux. Cooperation in a peer production economy experimental evidence from wikipedia. pages 1–31, 01 2013.

[12] Wikipedia:disambiguation. Wikipedia: The free Encyclopedia.

[13] Geert Lovink, Nathaniel Tkacz, Joseph Reagle, Dan O'Sullivan, Lawrence Liang, Alkim Akdag Salah, Cheng Gao, Krzystztof Suchecki, Andrea Scharnhorst, R.Stuart Geiger, Edgar Enyedy, Peter Kaufman, Johanna Niesyto, Hans Mathews, Scott Kildall, Nathaniel Stern, Nicholas Carr, Alan Shapiro, Florian Cramer, and Shun-Ling Chen. Critical point of view: A wikipedia reader. SSRN Electronic Journal, pages 342–350, 06 2012.

[14] Polls are evil. Wikipedia: The free Encyclopedia.

[15] Wikipedia:what wikipedia is not. Wikipedia: The free Encyclopedia.

[16] Luca Iandoli, Mark Klein, and Giuseppe Zollo. Can we exploit collective intelligence for collaborative deliberation? the case of the climate change collaboratorium. SSRN Electronic Journal, 12 2007.

[17] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. Decision Support Systems, 43:618–644, 11 2006.

[18] Ioana Literat. The work of art in the age of mediated participation: Crowdsourced art and collective creativity. International Journal of Communication, 6:2962–2984, 01 2012.

[19] Howard S. Becker. Art Worlds. Berkeley: University of California, 1982.

[20] Roy Ascott. Art Telematics: toward the Construction of New Aesthetics. NTT Publishing Co., 1998.

[21] Roy Ascott. Engineered Nature: art and consciousness in the post-biological era. Bristol: Intellect Books, 2006.

[22] Roy Ascott. The Future is Now: Art, Technology, and Consciousness. Beijing: Gold Wall Press.2012, 2012.

[23] Sarah Browne. Crowd theory lite 'the crowd' in participatory art and pop economics. Circa, pages 33–39, 01 2008.

[24] The sheep market: Two cents' worth. (master's thesis). Design and Media Arts Program, University of California, Los Angeles, 2006.

[25] Tedx talk, aaron koblin. TEDx Amazonia, Brazil, 2006.

[26] Howard S. Becker. Art as collective action. American Sociological Review, 39(6):767–776, 1974.

[27] Daniel Drescher. Blockchain Basics: A Non-Technical Introduction in 25 Steps. Apress, 2017.

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com, 03 2009.

[29] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. volume 3017, 02 2004.

[30] Adam Back. Hashcash - amortizable publicly auditable cost-functions. 12 2003.

[31] D. Zhang and Vivek Kanhangad. Encyclopedia on cryptography and security. Encyclopedia on Cryptography and Security, pages 529–531, 01 2011.

[32] Yi-Cheng Chen, Yueh-Peng Chou, and Yung-Chen Chou. An image authentication scheme using merkle tree mechanisms. Future Internet, 11:149, 07 2019.

[33] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. page 32, 2014.

[34] Elena Flor. Sdr and bitcoins: Competition or cooperation? The Federalist Debate, 31:22–23, 07 2018.

[35] Daniel Drescher. Choosing a Transaction History. 03 2017.

[36] Nick Szabo. Formalizing and securing relationships on public networks. First Monday, 2, 01 1997.

[37] Ian Grigg. The ricardian contract. pages 25 – 31, 08 2004.

[38] Chris Chinchilla. White paper. 6 2019.

[39] Imran Bashir. Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks. Packt Publishing, 2017.

[40] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. `https://eprint.iacr.org/2013/881`.

[41] Ethereum documentation.

[42] Cryptopusco. bytes and strings in solidity. 1 2018.

[43] Rangesh Sripathi. 6 payable functions in solidity — smartcontract — ethereum. 2 2018.

[44] Jack Tanner. Summary of ethereum upgradeable smart contract rd. 3 2018.

[45] Facu Spagnuolo Elena Nadolinski. Proxy patterns. 12 2018.

[46] Josselin Feist. Contract upgrade anti-patterns. 08 2018.

[47] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Liu Jia-Nan, Yang Xiang, and Robert Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. IEEE Transactions on Parallel and Distributed Systems, PP:1–1, 11 2018.

[48] Jelle Gerbrandy Elad Shtilerman Matan Field, Primavera De Filippi. Backfeed: Decentralized value distribution system for blockchain based applications. 2017.

[49] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. Blockstack: A global naming and storage system secured by blockchains. In 2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16), pages 181–194, 2016.

[50] Ethereum market, available: https://etherscan.io//, 2019.

[51] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger byzantium version e, 2019.

A Table of smart contract costs

Table of cost from solidty yellow paper [51]

GAS	FUNCTION
400	BALANCE
200	SLOAD
1	JUMPDEST
20000	SSTOREsetToNonZero
5000	SSTORE
5000	SELFDESTRUCT
32000	CREATE
68	codeTransaction
200	CREATEperByte
700	CALL
9000	CALLsetToNonZero
25000	CALLcreateAccount
10	EXP
3	addWord
4	addByte
32000	Contract-creating
375	LOGeachByte
21000	LOG
8	LOGbyte
375	LOGtopic
36	SHAinputData
30	SHA
33	COPY
20	BLOCKHASH
100	QuaadratictCoefficent