



GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793
Nr. ZFI-BM-201x-xx

Bachelor's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Implementation of a Peer-Reviewing Platform on the Blockchain

Fabiola Buschendorf

Institute of Computer Science

Bachelor's Thesis
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

25. June 2018

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌐 www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Dieter Hogrefe
Second Supervisor: Dr. Mayutan Arumathurai
Professional Advisor: Benjamin Leiding

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 25. June 2018

Abstract

The blockchain technology offers an infrastructure for decentralized applications, such as cryptocurrencies. The data structure is shared in a peer-to-peer network, in which untrusted members can distribute and process information transparently and verifiable. Modern blockchain platforms allow Turing-complete computation with so-called smart contracts and data storage for application development. Further, desired behavior is rewarded with cryptographic tokens, that can be traded for further services. This thesis takes advantage of those properties and proposes a prototype of an academic peer-reviewing platform on the blockchain. Peer-reviewing benefits from decentralization, as large scientific publishing houses possess a monopoly on renowned journals, thus they control access to and prices of published scientific articles. Further reports reveal that anonymity and lacking incentivization facilitate fraud within the process and cause poor-quality reviews. In this work, a requirements engineering process identifies associated roles and functionalities of an academic peer-reviewing system. An analysis compares existing blockchain platforms and selects a well suited environment for the system. The architecture of a decentralized web application is designed. Finally, an open-source prototype is implemented using recent tools and frameworks and code examples are explained.

Contents

1	Introduction	1
1.1	Contribution and Research Question	1
1.2	Related Work	2
1.3	Thesis Structure	3
2	Presuppositions	5
2.1	Blockchain Technology	5
2.1.1	The data structure	6
2.2	Ethereum Blockchain	7
2.2.1	Accounts	7
2.2.2	Transactions	8
2.2.3	The Block	8
2.2.4	Ethereum Virtual Machine	10
2.3	Peer-Reviewing Process	11
2.3.1	Criticism	13
3	Analysis	17
3.1	Requirements Engineering	17
3.2	Blockchain Comparison	20
3.3	Blockchain Platforms	21
3.3.1	Bitcoin	22
3.3.2	Ethereum	24
3.3.3	Steem	25
3.3.4	IOTA	26
3.4	Decision Process	27
3.4.1	Conclusion	28
4	Architecture Design	31
4.1	General Architecture	31
4.2	Smart Contracts	33

4.3	Access Control	33
4.4	File Storage	35
5	Implementation	37
5.1	Solidity Contracts	37
5.2	Defining Input Forms with JSON Schema	39
5.3	Accessing Contracts with web3	41
5.4	Adding Files to IPFS	42
5.5	Challenges	43
6	Conclusion and Future Work	47
6.1	Answering the Research Questions	47
6.2	Future Work	48
	Bibliography	55

List of Abbreviations

P2P	Peer-to-Peer	1
RQ	Research Question	2
PoW	Proof-of-Work	6
PoS	Proof-of-Stake	7
AOM	Agent Oriented Modeling	17
PC	Program Committee	12
DPoS	Delegated Proof-of-Stake	2
MCMC	Markov Chain Monte Carlo	27
ASICs	Application-Specific Integrated Circuits	23
TWh	Terawatt hour	23
GH/s	Giga Hashes per second	22
GHOST	Greedy Heaviest-Observed Sub-Tree	24
EVM	Ethereum Virtual Machine	7
DAG	Directed Acyclic Graph	26
IPFS	Inter-Planetary File System	27
RBAC	Role Based Access Control	33
EVM	Ethereum Virtual Machine	7
RPC	Remote Procedure Calls	31
PKI	Public Key Infrastructure	35
DHT	Distributed Hash Table	35
JSON	JavaScript Object Notation	37
ABI	Application Binary Interface	41

Chapter 1

Introduction

Peer-Reviewing is an essential part of science. Experts review scientific papers, spot mistakes and suggest improvements to their peers - colleagues and researchers from the same field. This is important to ensure correctness, novelty and substantialness of the subject [1].

The results are published in journals or presented at conferences. The more prestigious the publishing medium, the better the reputation for the corresponding author and reviewer and the higher the number of citations of the researchers' work, the more importance is associated with it. This process is not flawless: badly-paid or overworked referees do not spend sufficient time on reviewing, thus nonsense articles are being published [2], an anonymous reviewer can take advantage of her position and force authors to cite her work [3] and powerful publishing houses lock scientific results behind a paywall [4], causing high costs for readers.

To facilitate open access, incentivize reviewers for high-quality work and prevent misuse, M. Spearpoint [5] proposes a peer-reviewing platform on the blockchain data structure. A public blockchain is a transparent and shared log, maintained in a distributed Peer-to-Peer (P2P) network. Rewards in form of a digital token (i.e. *cryptocurrency*) can incentivize researchers to contribute with high-quality reviews. In such a decentralized network, no central institution is in charge of the managed process, thus censorship is not possible. Pseudonymity or anonymity [6] can be adopted, if required.

1.1 Contribution and Research Question

A next step is to develop a prototype. To the best of our knowledge, there is no implementation of a decentralized peer-reviewing platform to date. Additionally, there is a need for a detailed analysis of such platforms requirements and design. The aim of this thesis is to explore a possible architecture and tools in order to develop a minimum viable product in form of a decentralized

web application.

Such a system contributes in several ways to today's academic publishing environment: Firstly, it abolishes the need of trust in publishing platforms, as every organizational step on the blockchain is cryptographically verifiable. Secondly, it takes the monopoly of academic publishing from big publishing houses, thus it promotes competition in the business. Thirdly, it demands accountability of reviewers for their decisions. Finally it is an implementation of a relevant use-case on the blockchain, which serves as a proof-of-concept.

A more structured approach divides the technical contribution of this thesis into a central Research Question (RQ), which consists of three sub-questions:

How to design a blockchain based peer-reviewing system?

RQ1 : What are its functional requirements?

RQ2 : Which existing blockchain platform is suitable?

RQ3 : What is the system architecture?

1.2 Related Work

There are related projects involving content-rating, academic publishing and open access journal systems which we list and discuss here.

There are centralized open-access systems, namely *Open Journal Systems*¹, an open-source software founded by the Public Knowledge Project and *EasyChair*², a free platform for conference and review management. Both of these projects offer similar solutions: opening peer-review and taking control from big publishing houses. But these systems are hosted on centralized servers, thus internal processes are not transparent. The systems require each user to trust in the honesty of its managers, in which we see a general pitfall.

Since the blockchain technology emerged, decentralized systems had become possible. Steem, an incentivized, public content platform using the blockchain technology aims to offer a social media platform which rewards popular content [7]. Users can post content for free and receive digital tokens for comments and likes on their graphics and texts. The digital token is tradeable and exchangeable into an asset with which the users can either vote for verifiers, or take part in the verifying process themselves. Verifying is rewarded again. This principle is called Delegated Proof-of-Stake (DPoS). In 2016 there has been published a Whitepaper about PEvO (Decentralized Open Access and Evaluation) [8], which outlines a publishing and reviewing platform on the

¹<https://pkp.sfu.ca/ojs/>

²easychair.org

Steem blockchain. This project does not provide public code by now. Further, the Steem blockchain has some properties which we consider impractical, see Chapter 3.

Finally, as already mentioned in Section 1, M. Spearpoint published an article in 2017 about a proposed currency system for academic peer review payments using the blockchain technology [5]. The paper focuses on a token-incentive system in order to elevate the quality of current peer-reviewing. The work in this thesis takes a step further: It actually implements a conference management platform in which reviews are uploaded and organized decentral and it provides a proposal of an extensible architecture. It does not provide a currency system, yet, due to time constraints.

1.3 Thesis Structure

This work first explains the technical details behind a blockchain and briefly illustrates common peer-reviewing processes and their weaknesses in Section 2. Secondly, in Chapter 3 it elaborates on a requirements engineering process in which the systems goals and functionalities are determined in form of an Agent Oriented Goal Model (AOM), thus answering **RQ1**. It thirdly answers **RQ2** by analyzing different blockchain platforms in order to find a suitable underlying system for the development of a peer-reviewing system. The analysis and decision process is based on a taxonomy of blockchain platforms developed in 2017 by Xu et. al. [9]. Xu's work provides a comprehensible decision model and is, different than other taxonomies, widely accepted in the research field. Finally this work provides a first architecture design in Section 4 plus code examples of a bare-bones prototype implementation using the most common tools for decentralized web application development in Section 5, thus answering **RQ3**. In Chapter 6.1, future directions are suggested.

Chapter 2

Presuppositions

In this chapter, basic principles of the blockchain data structure (2.1.1). Additionally Section 2.2 elaborates on mechanisms of the Ethereum blockchain in particular. Common processes in peer-reviewing are explained in Section 2.3. Further, Section 2.3.1 underlines the criticism on current peer-reviewing models and publishers.

2.1 Blockchain Technology

Blockchain applications are popular since the cryptocurrency *Bitcoin*, developed by Satoshi Nakamoto, went public in 2009 [10]. The coins are exchangeable in a decentralized network, thus independent from central authorities. The blockchain technique provides a nearly immutable data structure for coin transactions. By appending new blocks to the current chain and propagating this information over the network, every participant has an overview of current balances.

Alternative currencies, named *altcoins*, started spreading with different application purposes either building on top of the Bitcoin protocol or developing an own system of the distributed transfer book. The idea of using the blockchain technology for various purposes was accelerated by *Ethereum* in 2013, initiated by Vitalik Buterin and developed by the Ethereum Team [11]. It established a network capable of executing small programs, called *smart contracts*. Converting the digital currency *Ether* to the internal pricing *Gas* these programs can be executed and verified by each participant, diminishing the need of a trusted third party. Though decentralization can contribute to the transparency and security of transactions and contracts, a blockchain is not a universal remedy. Current block sizes are limited and block creation can be time and energy consuming. Not all kinds of data should be accessible and transparent to the public. Further, the Bitcoin blockchain reached a size of 140 GB by May 23th 2018 ¹, impeding the participation in the network.

¹<https://blockchain.info>

2.1.1 The data structure

In detail, a blockchain is a data structure referencing its ancestor, similar to a back-linked list, see Figure 2.1. Each block contains information about several transactions and a header. In Bitcoin an average block carries 500 transactions, resulting in a block size of 1 MB [12]. The chain is kept redundantly on each participants computer, referred to as *node*, in a P2P network. The task of these nodes is to verify the correctness of each transaction and to bundle pending transactions, so that a new block is built. A *consensus* is reached if every node accepts a newly created block and updates its chain.



Figure 2.1: A blockchain containing hashes as references to transactions and the block's parent.

Each block is identified by a hash on its header. The block header contains a field for the previous blocks' hash, the protocol version, a timestamp, a reference to the block's transactions and information about the mining process. *Mining* refers to the act of adding a block to the chain and verifying transactions. This process is commonly designed as a difficult puzzle, limiting the ability of any one party to control the consensus process. In Bitcoin, this process involves finding a hash with sufficient trailing zeros, i.e. the miner needs to perform a hashing algorithm multiple times, which scales best on machines with a lot computing power, or *hash power*. This algorithm is time and energy consuming, so the *incentive* for a participant to mine a block is a reward in form of the networks currency (Bitcoin, Ether, Steem...) or token. This mining principle is called *Proof-of-Work (PoW)*, but alternative solutions exists. Once a block is mined and verified, the nodes start building on top of it. In the case two nodes solved the puzzle simultaneously, the consensus mechanism in Bitcoin is to accept the longest new chain, so each node has to build another block on top of the newly created one in order to convince the network to accept its mined blocks and to receive the reward [13].

As the verification of hundreds of single transactions can be tedious, another useful data structure is used: A binary tree called *Merkle tree*. In a Merkle tree each data point, e.g. a transaction, is hashed and grouped into pairs. The hash of each of these pairs is taken and stored in a parent node. The parent nodes are grouped in pairs again, hashed and stored one level up the tree, see Figure 2.2. A single hash results and is stored in the block header as the Merkle tree *root*. Verification of existence of a single transaction is in $\mathcal{O}(\log(n))$, if n transactions are included [12].

This data structure is living in a public or private P2P network. Each participant holds a copy of the blockchain and has equal power. Bitcoin for example, is a public network in which everyone can take part by downloading a client software and start trading or mining.

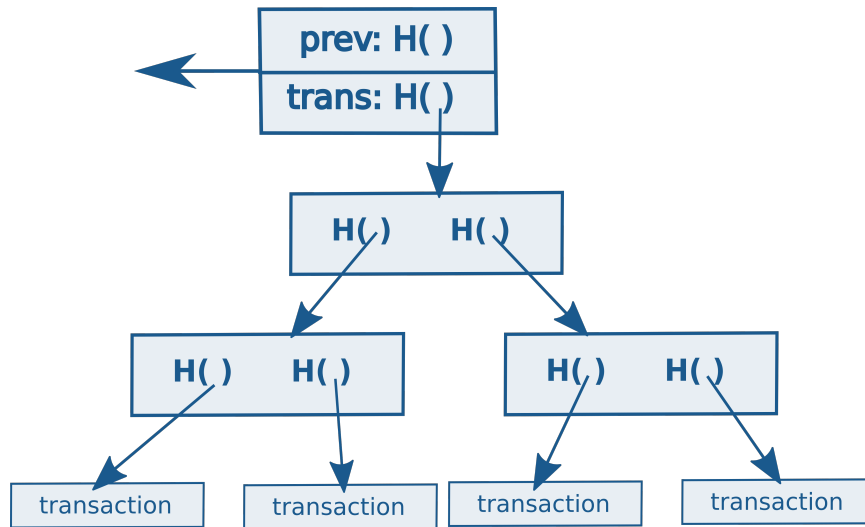


Figure 2.2: A Merkle-tree referencing the blocks transactions

2.2 Ethereum Blockchain

In Section 2.1, the functionality and components of a blockchain are explained in general. This section refers to some specific features of the Ethereum blockchain, which are necessary to follow the work done in this thesis. Ethereum is a public blockchain which features the cryptocurrency *Ether* and enables turing-complete application development. Written in a scripting language, such applications, called *Smart Contracts* can be executed by a virtual machine on each miner's computer. Ethereum currently uses PoW, but a switch to Proof-of-Stake (PoS), a more energy-efficient algorithm, is planned. Its Whitepaper was published in 2013 by Vitalik Buterin [14] and in 2014 a technical description was given in the Yellow paper by Gavin Wood [11]. Both are being further developed until now. The following sections explain the Ethereum-specific structure and function of accounts, transactions and the Ethereum Virtual Machine (EVM).

2.2.1 Accounts

Ethereum declares both subjects and contracts as *accounts*. An account with associated code represents a contract which can receive *message calls* to execute the accounts code. An account is identified by a 160-bit address and is mapped to an *account state*. A state is a serialized data structure in form of binary data with arbitrary length (byte-array). This information is then stored in a database, where the account's address is kept as a key and the byte-array represents its value. This key-value pair is identified by a 32-byte hash, which is stored in a modified Merkle Tree. As explained in Section 2.1, this structure allows to obtain a root-hash which represents the current

world state in a cryptographically secure manner. Further it allows to proof existence of a particular key-value pair within a Merkle tree of a given root hash. The account state consists of following fields:

- **nonce**: A value representing the number of transactions of that account, or if it is a contract, the number of contract-creations made.
- **balance**: The number of *Wei* (smallest unit of Ether) owned by that account.
- **storageRoot**: Root of the modified Merkle tree which encodes the storage contents of that account. This data is stored permanently.
- **codeHash**: The hash of the immutable EVM code that is executed when the account receives a message call. Assuming the account represents a non-contract account, this field is the hash of an empty string.

2.2.2 Transactions

Transactions are used to (i) call functions of a contract via *message calls* or (ii) to create a new contract. They are cryptographically signed by the sender and specify a number of common fields:

- **nonce**: The number of transactions send by the sender.
- **gasPrice**: The amount of Wei to be paid per unit of gas. This price varies with the load of the network and needs to be adjusted by the sender.
- **gasLimit**: The maximum amount of gas that should be used in this transaction.
- **to**: In (i) the address of the receiving contract or in case (ii) an empty field.
- **value**: A number of Wei that gets (i) transferred to the recipient or that (ii) is used to fund the newly created contract
- **init**: (Only in (ii)) Is a piece of EVM code that gets executed only once and returns the code body which can be executed via message calls.
- **data**: (Only in (i)) A byte-array: The input data of the message call.

A signed transaction is applied to the current state, as depicted in Figure 2.3.

2.2.3 The Block

A Block consists of several transactions, relevant pieces of information represented in the block's *header* and a list of valid alternative parent headers, known as *uncles* or, the gender neutral term: *Ommers*. As Ethereum uses PoW, the content of the block's header does not differ a lot from Bitcoins header. The main difference is the presence of *ommers*: In Ethereum, the *heaviest* chain is

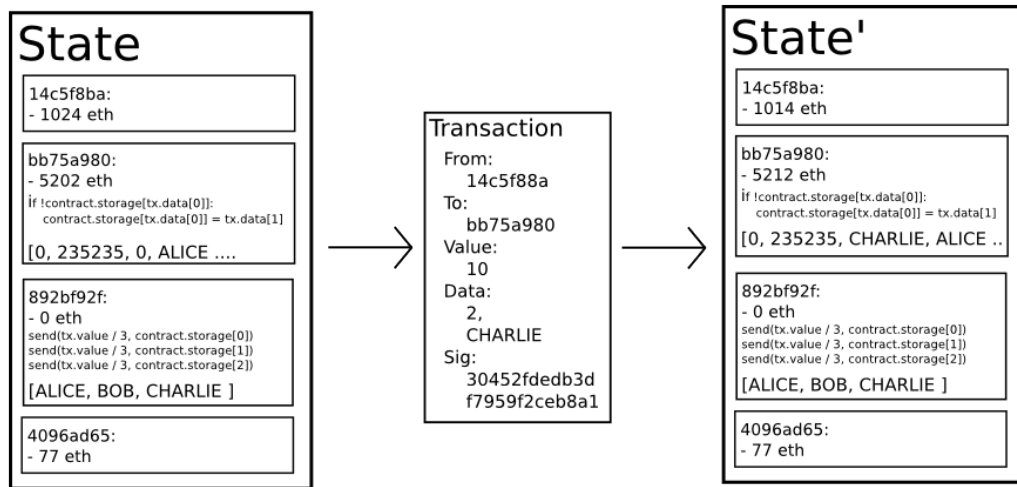


Figure 2.3: Ethereum state transition through a transaction. Source: [14]

chosen for consensus, meaning, a valid block referencing the most ommers is chosen by all nodes. Figure 2.4 depicts the structure of a modified Merkle tree referencing accounts state's including another tree which stores the accounts storage data.

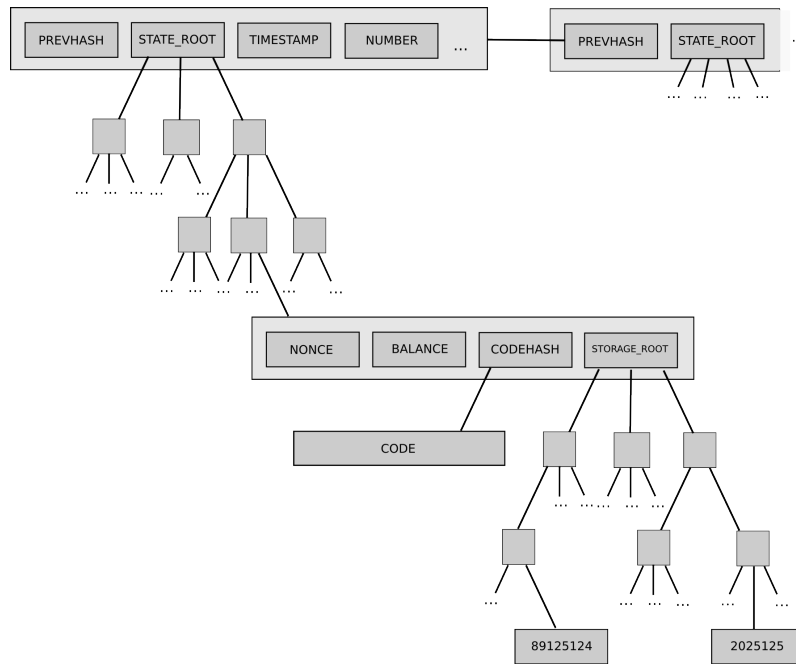


Figure 2.4: Ethereum block structure with modified Merkle tree of states and accounts. Source: [15]

2.2.4 Ethereum Virtual Machine

Each transaction alters the systems state and results in a new world state. Computations involved with message calls and contract creations invoked by transactions are executed in a virtual machine on each miner's computer. The EVM is a virtual state machine, which is quasi-Turing-complete, bounded by the parameter *gas*. The EVM is a 32-byte-word stack machine which executes simple instructions of a set of arithmetic, bitwise, logic, cryptographic operations. Each instruction has a fixed fee, the *gas price*. Initial memory usage requires to pay a higher fee than re-usage of already allocated storage. Additionally there is a reward for clearing an entry in the storage. For example, Table 2.1 lists a few gas prices for common operations and Table 2.2 lists gas prices for more complex operations, like creating new contracts or storing new pieces of data.

During an *execution cycle* the machine executes the code of one transaction and deduces the proper amount of gas. After it iterated over all transactions of that block, the result of all state transitions is determined and if a valid state results, the miner's computer can proceed with performing the PoW. Figure 2.5 depicts how a blocks transactions are applied subsequently by a miner's EVM reach a new world state.

Representation	Name	in	out	Costs (in gas)	Description
0x01	ADD	2	1	3	Addition operation.
0x02	MUL	2	1	5	Multiplication operation.
0x20	SHA3	2	1	30	Compute Keccak-256 hash.
0xf0	CREATE	3	1	32000	Create a new account with associated code.
0xf1	CALL	7	1	40	Message-call into an account
0x55	SSTORE	2	0	20000	Storage value is set to non-zero from zero.
0x55	SSTORE	2	0	5000	Storage value is set to or remains zero.
0xff	SELFDESTRUCT	1	0	5000	Halt execution and register account for later deletion
...

Table 2.1: Selected EVM operations, their 32-byte representation, name, number of input and output parameters and their costs. Source: [11]

Name	Gas	Description
Transaction	21000	Paid for every transaction.
New Account	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
Selfdestruct	24000	Refund given (added into refund counter) for self-destructing an account.
...

Table 2.2: Selected abstract EVM operations. Source: [11]

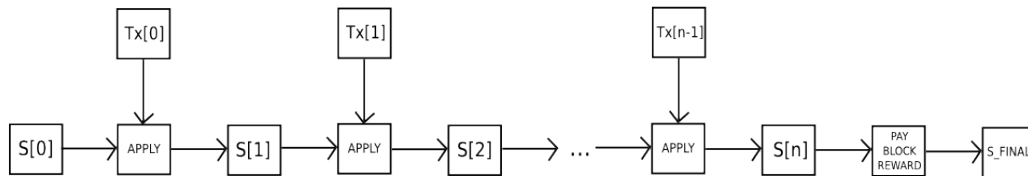


Figure 2.5: Applying a block's transactions to the previous block's state results in the new world state. Source: [14]

2.3 Peer-Reviewing Process

In academic research scientific articles are published by journals, at conferences or collected in proceedings. Peer reviewers help to validate and ensure the correctness and sufficient novelty of an articles findings [16]. Typically there are specialized publishing houses or mediums targeting one or more scientific fields. Some of the most well-known publisher's are Elsevier, IEEE, Springer or AAAS. For instance, AAAS publishes *Science* and Springer Nature publishes among others the

renowned weekly journal *Nature*.

A common metric for a journals influence is the *impact factor*. It is a yearly measure which reflects the number of citations to articles published in that journal [17]. To ensure high quality, submissions firstly underlie basic criteria concerning the articles topic, structure, grammatical correctness and length. Secondly, they undergo a peer-reviewing process that involves actors of different roles, which are described here by referencing the mechanisms used by the platform *EasyChair*². EasyChair is a free, web-based tool to organize academic conferences independently, create events, invite Program Committee (PC) members, announce call for papers and assign submitted papers to reviewers. EasyChair involves three main roles [18]:

1. **Reviewer / PC Member:** A PC member can either be a reviewer herself or delegate a review to another person.
2. **Author:** an author can make submissions to conferences, update these submissions and receive notifications from chairs.
3. **Chair:** A chair is a special member of the program committee and has more privileges than ordinary PC members. A chair can edit the conference configuration and change the PC.

These roles are flexible. There exist different peer-reviewing models: Either the names of the reviewers are hidden from the author (single blind review), the author and the reviewer are anonymous (double blind review) or both names are known to each other (open review) [16]. Accessing an article requires scientists to pay, if the author did not decide to publish her article as *open-access*, in which case she pays in advance [4].

In detail, the peer-reviewing process in case of a conference using *EasyChair* works as depicted in Figure 2.6: Firstly, a chair creates a conference and configures a submission form, provides information about deadlines, the required submission format and configures a reviewing model. She then invites PC members and creates a public call for papers, e.g., opens the submission form. Secondly, authors fill out the specified form and upload their work. Thirdly, a matching process assigns a submitted paper to a reviewer, or a PC member delegates the task to a sub-reviewer. Assigning papers is done using either a randomized algorithm which takes care of conflicts of interest, or PC members bid on papers they prefer to review. Finally the review is submitted. A final decision about which paper is accepted for the event is taken by the chair. Optionally a corrected version of that paper can be re-submitted [18].

²easychair.org

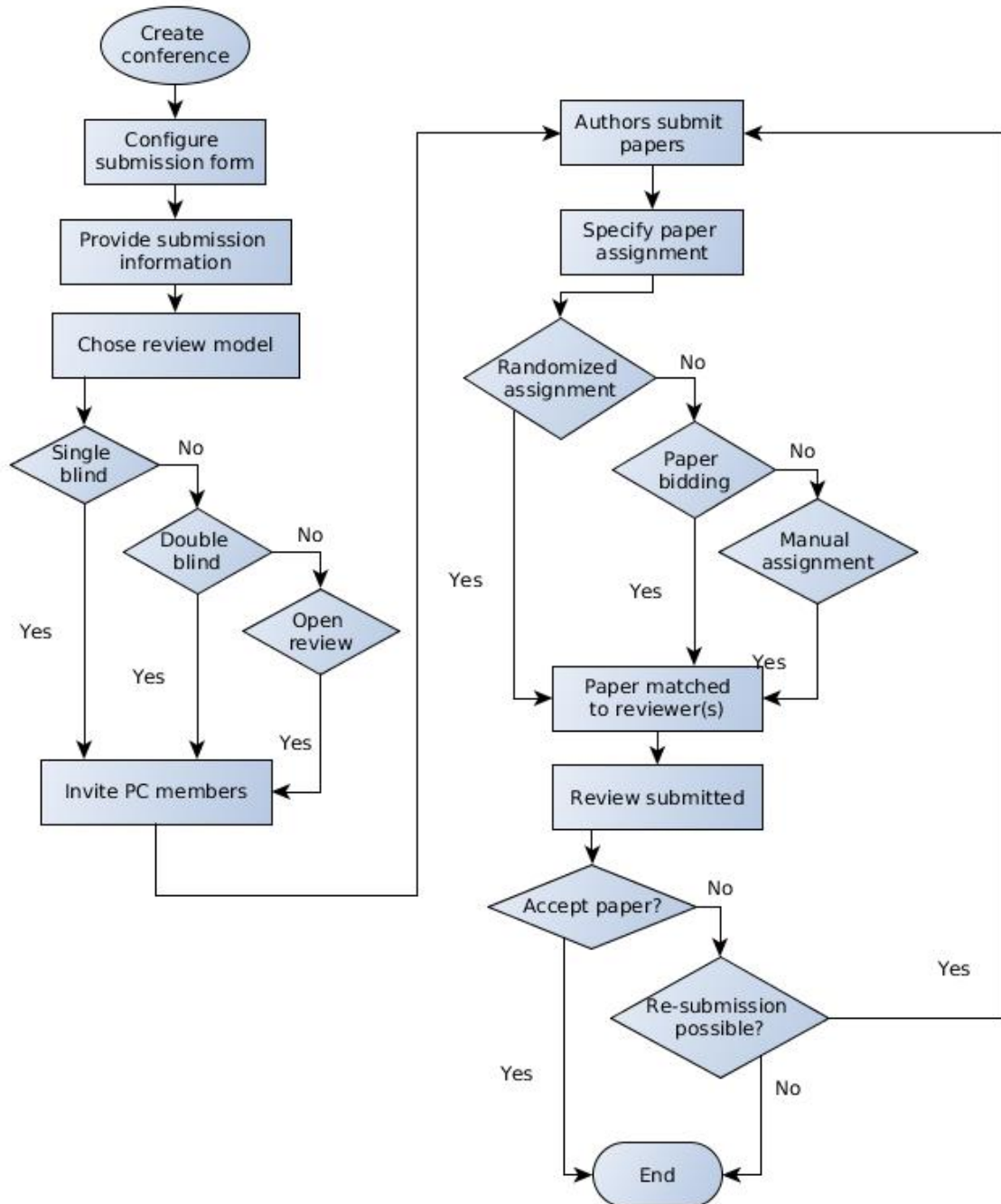


Figure 2.6: Conference management workflow of EasyChair.

2.3.1 Criticism

Criticism today may be divided into two categories: Flaws of the reviewing practices itself and the publishers role within the process.

In academia, doctoral students and professors are often underpaid or overworked. Consequently, referees do not spend sufficient time on reviewing, thus nonsense articles are being published [2]. In case of a blind or double-blind review, an anonymous reviewer can take advantage of her position and force authors to cite her work [3]. However, if anonymity is not given, peer-reviewing is prone to bias against authors from less prestigious institutions or female authors [19].

Further, powerful publishing houses lock scientific results behind a paywall [4], causing high costs for readers. Many publisher claim the right to distribute and charge for scientific articles, but authors argue that publicly founded science should be publicly available [20]. Recently, more than 200 German Universities and research institutions shut down their contracts with Elsevier, arguing a margin of more than 30 percent for a publishing business is too high and reviewers are badly-paid. Moreover, they criticize the monopoly of high impact-factor journals and Elseviers lacking openness towards open-access solutions as the publisher is charging around 2000 dollars per such published article [21].

Chapter 3

Analysis

In this chapter, the analysis of a set of different existing blockchain platforms is described. Section 3.1 elaborates shortly about the platforms' requirements, in Section 3.2 a set of blockchain-features is introduced on which basis Section 3.3 compares different existing blockchains, namely Bitcoin, Ethereum, Steem and IOTA. Finally, Section 3.4 presents a decision model which will conclude about which blockchain is preferred for this specific use case.

3.1 Requirements Engineering

To specify the functional requirements of a peer-reviewing system, the main actors in publishing and reviewing and their roles are depicted in Section 2.3. This process is used to specify goals of each actor, using an Agent Oriented Modeling (AOM) Goal Model. Sterling and Taveter summarize this modeling concept as follows:

"The main objective of goal models is to enable a customer, domain analyst, and system designer to discuss and agree on the goals of the system and the roles the system needs to fulfill in order to meet those goals." [22].

Figure 3.1 describes the notation used in Figure 3.2. Summarized, the required functionalities extracted from the model are:

- a) Create identifiable accounts
- b) Create a conference
- c) Send messages/ invitations out of the system
- d) Connect account with a conference
- e) Assign roles to an account
- f) Upload files
- g) Match paper to a reviewer

- h) Rate content
- i) Send messages within the system






Symbol	Meaning
	Goal
	Quality goal
	Role
	Relationship between goals
	Relationship between goals and quality goals

Figure 3.1: The notation for goal models [22]

Altogether, this section answers RQ1: "What are [the systems] functional requirements?".

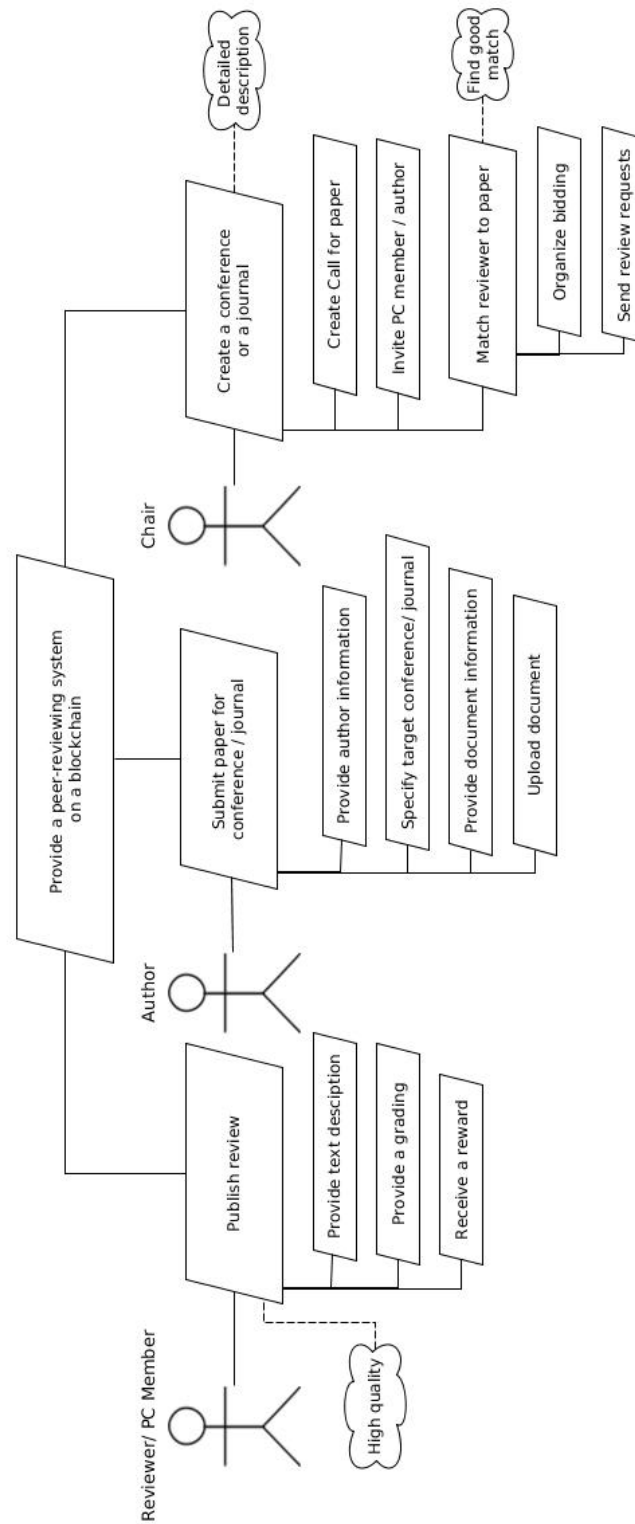


Figure 3.2: Functional requirements with AOM goal model

3.2 Blockchain Comparison

Our application is to be implemented on a public blockchain that is capable of running small programs. In [9] Xu et al. evaluate various blockchain properties with respect to cost efficiency, performance and flexibility, suggesting favorable characteristics. Using their findings, Section 3.3 analyzes characteristics of the different blockchains. In this section those selected criteria and their importance with respect to this project are introduced.

Adoption A well adopted blockchain offers a vivid development community, larger computing resources, easier participation and more eager improvement of its protocols. Less adopted systems are more likely to fail in the near future. This feature can be measured in several metrics: The number of full nodes, total hash-power in the network (for PoW) or the number of registered wallets/ users.

We aim to implement our prototype on a well adopted blockchain to facilitate its future development.

Usability High usability is important to facilitate access to an application to eventually attract end users. Some blockchains might require participation in mining, downloading the full chain (*full node*), or part of the chain (*light node*). Participation sometimes involves creation of a user account or a wallet, which can undergo a lengthy identification process.

We favor low requirements to technical know-how, thus lowering the barrier for participation for the end user.

Consensus Algorithm and Incentivation Consensus algorithms have an impact on the systems security, their degree of decentralization, power consumption, processing speed and incentivation [9] [12]. In this section we will compare architectures which implement variations of PoW, PoS and DPoS protocols, based on results of a survey in 2017 by Zheng et al. [23].

Due to its large scope, this criteria has a huge impact on the decision process. We aim to operate on a secure, energy-saving and highly decentralized blockchain.

Maturity The maturity of a system relates to its development stage and development duration. Software systems at a later stage of development have been extensively tested for bugs, include more product features, are performance increased and less likely to undergo essential code changes [24, p. 22].

We therefore favor blockchain platforms at late stages of development (beta over alpha) and with a longer development history in years.

Scalability The scalability concerns the transaction processing rate, e.g. the mining frequency and the block size. The bigger a blockchain network becomes, the more prone it is to transaction delays and the bigger the to be stored data structure gets. This drives small peers away to join the network, thus causes centralization [12, p. 52]. For efficiency reasons, smaller mining frequency and bigger block size are desirable but for scalability a larger mining frequency and a smaller block size is an advantage [9, p. 9].

Our application does not require precise on-time communication, but it might store relatively large amounts of data, depending on how much information we are willing to store off-chain.

Application Development This criteria rates *a)* the ability of the blockchain to run turing-complete pieces of code and *b)* the support for developers. The amount and quality of application development documentation, tutorials, good code examples, community support and the availability of test networks are taken into consideration.

These aspects will have a strong impact on the development duration, the extend and the quality of this project, as the development time is limited to three months.

Cost efficiency Blockchain transactions often require a fee to incentivize mining, which ultimately prevents spamming [12, p. 15, 123]. Complex computations and high amounts of data storage can increase the costs of an application. Sometimes a fee is substituted by other regulations, such as a maximum bandwidth [25].

This prototype aims to require no costs at all.

3.3 Blockchain Platforms

The public blockchains which we are going to analyze are: Bitcoin, Ethereum, Steem and IOTA. In April 2018, 1597¹ cryptocurrencies are listed online, each of those based on a distributed ledger technology. We chose those four examples because they meet certain basic criteria: The blockchain needs to support application development, it requires a fairly large number of active nodes to host our data and carry out computations reliably and it should be sustained by a large group of active developers. The findings are listed in Tables 3.1 and 3.2 and 3.3.

In [9] Xu et al. evaluate various blockchain properties with respect to cost efficiency, performance and flexibility, suggesting favorable characteristics. Using their findings, this section analyzes characteristics of the different blockchains.

¹coinmarketcap.com

Blockchain	Adoption			Consensus Algorithm
	# Wallets	# Full nodes	Hash Power	
Bitcoin	23 681 000	11 663	26 499 459 573 GH/s	PoW: Hashcash
Ethereum	29 095 934	16 686	270 000 000 GH/s	PoW: Ethash (switch to PoS)
Steem	870 414	< 81	-	DPoS
IOTA	~ 260 629	~ 157	-	Centralized (switch to MCMC)

Table 3.1: Feature comparison: Comparing (numerical) values from March 2018. The number of full nodes, wallets and the hash power indicates the degree of adoption of that blockchain. The different consensus algorithms have a different degree power consumption, e.g. PoW is more energy consuming than PoS.

3.3.1 Bitcoin

Bitcoin is the first blockchain-based cryptocurrency, released in 2009 by Satoshi Nakamoto [10]. Its main purpose is to substitute fiat currencies and a trusted central bank, whose history is according to Nakamoto "full of breaches of that trust" and further cause "massive overhead costs" [26].

Adoption Bitcoin is the most heavily minted cryptocurrency with a total power of 26 499 459 573 Giga Hashes per second (GH/s). On average, 11,663 full nodes are active over a period of 90 days and there are 23 681 000 wallet users registered [27] [28].

Usability As Bitcoin is the most well known blockchain, one finds many resources, such as the Bitcoinwiki². Usage of a lightweight node is possible, it does not require participation in mining. Discussions on Reddit and Stackoverflow are popular.

Consensus Algorithm and Incentivation Bitcoin uses a PoW algorithm, called Hashcash [29]. It requires strong CPU computation to find a SHA-256 hash that surpasses a certain difficulty in order to add a new block to the chain. The miner whose block was accepted receives 12.5 BTC, which is 86 219.88 US\$ at April 5th 2018, plus fees. In Bitcoin, the block mining reward halves every 210,000 blocks, so the reward will decrease to 6.25 in 2020.

²<https://en.bitcoin.it/wiki/>

Blockchain	Maturity		Scalability	
	Release	Age	Transaction Speed	Block size
Bitcoin	Release: 192	9 years	10 min.	<1 MB
Ethereum	Release: 127	4	14 sec.	~ 18 KB
Steem	Beta: 67	2	3 sec.	< 65 KB
IOTA	Early Beta: 48	2	2 min.	-

Table 3.2: Feature comparison: Comparing (numerical) values from March 2018. A higher release category and number indicates a further developed software. Older software might be more stable than recently developed software. Any new information on a blockchain is limited by its transaction speed. A blockchain might grow very fast if the block size limit is too high.

Due to the algorithms design it is possible to build specialized hardware, namely Application-Specific Integrated Circuits (ASICs). Further, mining only creates revenue if the latest hardware is used [12, pp. 142-144]. Nowadays, these can only be afforded by large institutions, which results ultimately in centralization. Cooling and running ASICs requires a huge amount of electricity and different estimations reported on 11th January 2018 that the network consumes between 18.40 and 42 Terawatt hour (TWh) per year, which could sustain between 1 685 208 and 3 846 671 U.S. households [30] [31].

Maturity The Bitcoin Core software is maintained by a community since its release in 2009. The Github repository is maintained by 24 members and 525 contributors. It is at version 0.16.0 on its 192nd release [32].

Scalability Bitcoins block size limit is 1 MB and it is mined once in 10 minutes on average [12, p. 98]. This configuration could not prevent transaction delays: At a peak time in November 2017, the median transaction time was about 20 minutes for transactions which included a fee. The blockchain size is 163 GB in April 2018 [27].

Application Development Bitcoins scripting language is not turing-complete [12, p. 79], so we can not develop complex applications on this platform.

Cost efficiency Daily average fees for one transaction are currently around US\$ 0.17, but have been at US\$ 36 in late December 2017 [33].

Blockchain	Development		Cost Efficiency
	Turing-Complete	Code Example	Transaction Fee
Bitcoin	No	-	0.17\$ - 36\$ per transaction
Ethereum	Yes	Yes	0.17\$ - 4\$ per transaction
Steem	Yes	Yes	No
IOTA	Not implemented	-	No

Table 3.3: Feature comparison: Comparing (numerical) values from March 2018. For easier development our application requires a turing-complete programming language, enabling the user to create Smart Contracts. Further we need a variety of code examples. Deploying this code on the blockchain usually costs a fee. We aim to pay no fee at all.

3.3.2 Ethereum

Ethereum was introduced in Section 2.2.

Adoption Ethereum is the most widespread cryptocurrency with a total power of approximately 270 000 000 GH/s [34]. On average, 16 686 full nodes are active [35] and there are 29 095 934 unique addresses registered [36].

Usability Like Bitcoin, Ethereum offers a wiki³, a documentation *Ethdocs.org* and many introductory tutorials on the organizations website⁴ for end users. Discussion on Reddit and Stackexchange are popular.

Consensus Algorithm and Incentivation Ethereum was launched with a PoW algorithm based on Ethash, which is a memory-hard problem. Modern consumer GPU's are already highly specialized for mining, thus mining is rather ASIC resistant. Additionally, the Greedy Heaviest-Observed Sub-Tree (GHOST) protocol allows referencing stale blocks (uncle blocks) to add weight to a chain and support its selection as the main chain. The miners of the uncle blocks are also rewarded, which combats centralization [37].

³<https://github.com/ethereum/wiki/wiki/>

⁴ethereum.org

Maturity The Ethereum Go Client software is maintained by a community since its release in 2014. The Github repository is maintained by 38 members and 246 contributors. It is at version 1.8.3 on its 127nd release [38]. There is a *development roadmap* which describes past stable versions of Ethereum and future developments [39].

Scalability On average, every 14 seconds a new block is added. Ethereum does not include a block size limit, but there is a *gas limit*, e.g. the computation and storage costs cannot exceed a certain value which is voted on by miners [37]. Currently, the average block size is 18 KB [36].

The GHOST protocol combats centralization and allows shorter inter-block times, thus increasing performance [40].

Application Development Ethereum is intended as a platform for decentralized application development and usage. The EVM can execute arbitrary code in different turing-complete languages [37]. For developers, there are testnets like *Rinkeby*⁵ on which application code can be executed for free. Open-Source projects like *Origin Protocols* host a demo on the testnet and serves as an example to learn from.

Cost efficiency Daily average fees for one transaction are currently around US\$ 0.17, but have been at US\$ 4 in mid January 2018 [41].

3.3.3 Steem

The Steem blockchain hosts a social media network *steemit.com* on which users can share content like texts, images or videos and earn bits of a cryptocurrency for high rated postings. It was introduced in 2016 by Ned Scott [25].

Adoption On Steem there are 347 witnesses (equivalent to miners) with more than one voter active in April 2018 [42], 870 414 accounts registered and out of all reachable full nodes in 10 April 2018, 24 were online and 57 were offline [43].

Usability Steem Inc. offers a FAQ and some information for developers⁶ and a *Discord* channel. Registering an account at Steemit requires providing an E-mail address and a telephone number. This information is validated, which takes several days. This way the developers want to combat spam. Participating and developing on Steem does not require being a witness nor hosting a full node.

⁵<https://www.rinkeby.io/>

⁶<https://developers.steem.io/>

Consensus Algorithm and Incentivation Steem uses DPoS. Users earn tokens by creating and curating content, which then gives them the power to vote on a *witness* for block creation. Each round, 20 of the most upvoted witnesses plus one randomly chosen one creates a new block. This generates additional Steem tokens which are distributed among the witnesses, the authors and curators of postings. This mechanism incentivizes the users to create interesting, high-quality content and to read, upvote and comment on postings [25].

Maturity Steem's release is on Beta. It is maintained by an open community with approximately 9 active developers and 46 contributors. It is at version 0.19.4 on its 67th release. There is a *development roadmap* for 2018 posted in December 2017 [44].

Scalability On average, every 3 seconds a new block is added. The maximum block size is 2^{16} bytes.

Application Development It is possible to develop applications for Steem with JavaScript, a Turing-complete language. However, the API documentation is still under development and important information is missing. There is an open source project developed on the Steem blockchain, *DTube*⁷, whose code is available on GitHub.

Cost efficiency Steem does not include a fee.

3.3.4 IOTA

The Internet of Things Association (IOTA) is a cryptocurrency which aims to facilitate payment and communication between Internet of Things devices. It is based on an Directed Acyclic Graph (DAG) in which transactions are approved by being referenced, thus verified by another transaction. A DAG does, unlike a blockchain, not require mining [45]. IOTA was founded in 2015 and its core GitHub repository went online in November 2016 [46].

Adoption Currently there is no mechanism to measure the amount of full nodes in IOTA. However, there are efforts to list a number of known nodes by *iotanodes.org*: One source lists 157 nodes. Other estimations claim a number of 260 629 registered addresses with positive token balance [47].

Usability There is few information about using IOTA for real applications.

⁷d.tube

Consensus Algorithm and Incentivation Consensus is currently achieved by a centralized *Coordinator*, which first approves each transaction to confirm it.

In the future, the DAG aims to reach consensus by having each transaction referenced and validated by a set of following transactions. Thus, a transaction can only be confirmed with a certain confidence. The algorithm is derived from a family of Markov Chain Monte Carlo (MCMC) algorithms.

In order to prevent spam from the network, a PoW algorithm with relatively low difficulty is performed to create a valid transaction [45].

Maturity The IOTA Organization on GitHub includes 8 members, 25 people contributed to its Javascript library which is on early beta 48th release v0.4.7 [46].

Scalability The Coordinator references transactions every two minutes [45].

Application Development A blog post confirmed that there is no support for smart contracts in IOTA until the end of 2018 [48]. Other than that, there is a documentation in development and a public Testnet available [45].

Cost efficiency There is no transaction fee, but purchasing IOTA tokens is necessary to transfer value on the DAG.

3.4 Decision Process

We follow the decision process for blockchain based applications developed by Xu et al. in Figure 3.3. As explained in Chapter 1, we develop a system which does not include a trusted authority. Given the limitations of a blockchain, we will not store large amounts of data on-chain, but will manage the organizational process and computation on-chain and might use the Inter-Planetary File System (IPFS) or another peer-to-peer storage system to store scientific work.

We analyzed popular existing blockchains in Section 3.3 and clarified our interest in a public blockchain. Concerning the data structure we can already exclude IOTA with its DAG. Currently the distributed database is still dependent on a central *Coordinator* to confirm transactions and smart contract development is not possible yet. Further, the user base and community is small compared to other projects, consequently a soon implementation of a decentralized consensus mechanism cannot be expected.

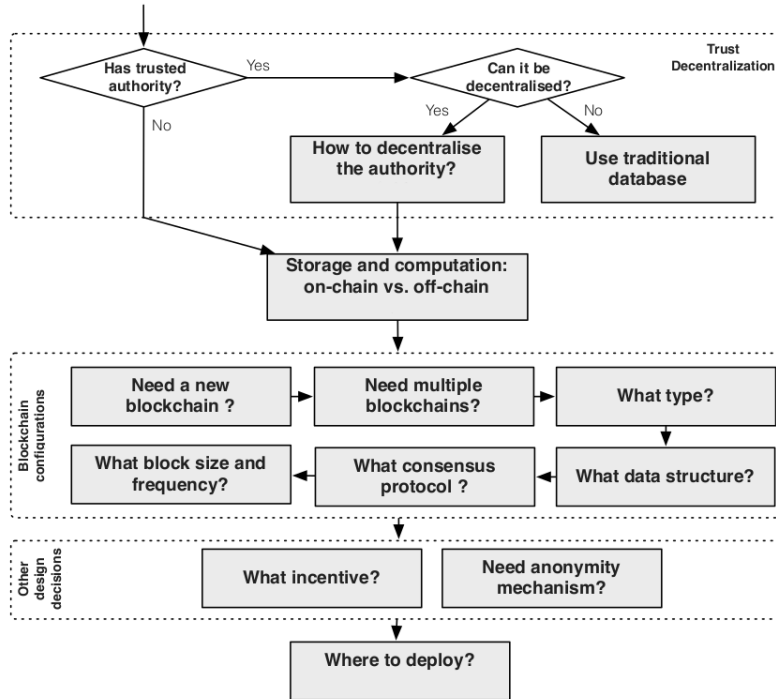


Figure 3.3: Decision process by Xu et al. Source: [9]

Regarding the consensus protocol we prefer PoS variants, because of better cost-efficiency, smaller energy consumption and shorter latency [9]. Thus, Bitcoin is not suitable to host our application, especially because it does not enable turing-complete application development and because of high transaction fees. Though Ethereum is currently operating with PoW, there are well developed plans to switch to PoS [49] backed up by a strong developer and community background. The Steem DPoS is in the hands of only a few powerful witnesses, visualized at steemd.com/witnesses. Due to its smaller user base it is less distributed, thus less stable and secure compared to Ethereum. Block sizes and frequency in Ethereum and Steem are both sufficient. However, the incentive mechanism of Steem, which encourages curating and creating high quality content fits our peer-reviewing project well. It further does not, unlike Ethereum, require a fee. But considering the stronger community background, the more mature technology of Ethereum, a huge number of full nodes, its fee-less testnet *Rinkeby* on which a prototype could be implemented and existing code examples, we ultimately chose Ethereum as the most suitable blockchain for our project.

3.4.1 Conclusion

We compared four popular distributed systems, Bitcoin, Ethereum, Steem and IOTA using seven features: Adoption, Usability, Consensus Algorithms and Incentivation, Maturity, Scalability,

Application Development and Cost efficiency based on the requirements of a distributed peer-reviewing platform prototype. Most importantly, the systems support for application development and usability is considered, which has led to the decision of using Ethereum's testnet *Rinkeby* to implement the prototype of a distributed peer-reviewing platform. This decision answers RQ2: "Which existing blockchain platform is suitable?".

Chapter 4

Architecture Design

In this chapter, the design of the systems architecture is explained in detail. The platform is developed as a decentralized application (*dapp*), which results in a different architecture than common web applications. The front end is written in HTML, using the JavaScript framework *react* and *Bootstrap*. This will not be explained further in this work.

4.1 General Architecture

Figure 4.1 depicts the interaction between the back end, web interface and file storage in form of an UML deployment diagram¹. Smart contracts, deployed on the Ethereum testnet *Rinkeby* perform internal computations like conference and account management. This computation is executed via the EVM on each miner's computer. The web interface is deployed on a web server, while conference management information is fetched from the blockchain via Remote Procedure Calls (RPC). This communication is enabled by the JavaScript API *web3*, which is injected by the add-on *MetaMask* into the user's browser. Using RPC does allow us to use the contracts data and functions without downloading the whole blockchain. Detailed conference information, paper PDFs and review texts are stored on the P2P file storage network IPFS.

Thus, this section answers RQ3: What is the system architecture?

¹<https://www.uml-diagrams.org/deployment-diagrams.html>

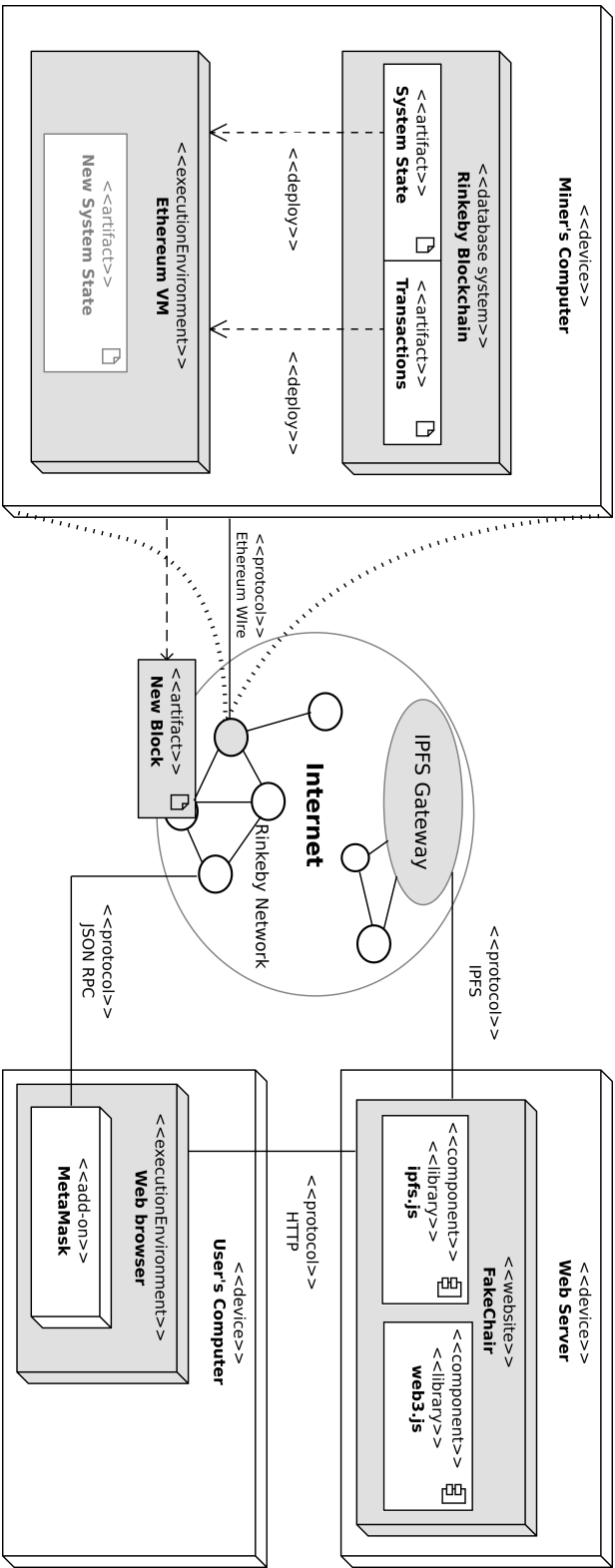


Figure 4.1: The overall system structure. Through a browser add-on the Ethereum testnet Rinkeby is selected and the content is displayed by a simple web interface. Smart contracts are executed on each miner's EVM, the results are fetched via the API web3. Larger files are stored on IPFS.

4.2 Smart Contracts

The Ethereum blockchain offers a built-in turing-complete programming language, allowing developers to write small programs, called *contracts* [14]. Data, stored and manipulated by smart contracts remain permanently and publicly on the blockchain, therefore we use smart contracts to provide the back end for our application.

Figure 4.2 depicts the system’s internal logic in form of an UML component diagram². This system uses Role Based Access Control (RBAC) to assign roles to already existing accounts on the blockchain. Section 4.3 explains the access control model in further details. Conferences are created via a conference registry. Each conference possesses a list of papers, each paper possess a list of reviews. Certain actions, such as adding authors to a conference, are restricted to *admins*.

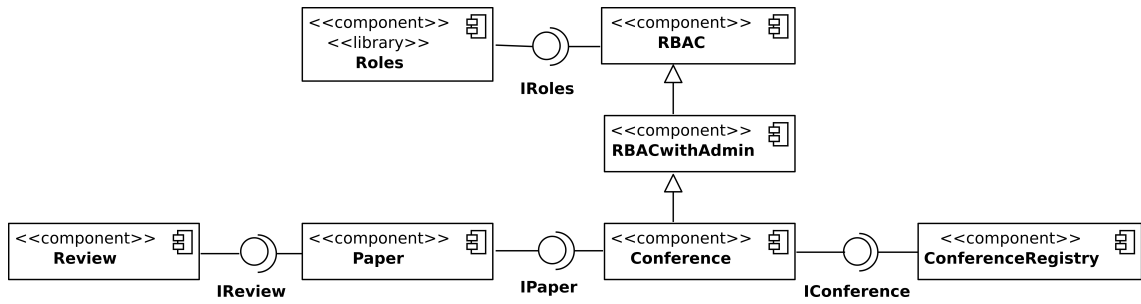


Figure 4.2: The back end: Smart Contracts, featuring role based access control (RBAC).

4.3 Access Control

Within this system, there are subjects S (users or accounts), objects (conferences, paper, reviews) and permissions P on operations (create/write or read object instances). Further, we identified three roles R essential to the peer-reviewing process in Chapter 2. A suited access control model needs to be applied to manage basic security requirements efficiently. We decide to use role based access control (RBAC). In [50] Ferraiolo and Kuhn define three rules for RBAC:

1. Role assignment: A subject can execute an operation only if the subject has selected or been assigned a role.
2. Role authorization (RA): A subject’s role must be authorized for the subject.
3. Permission assignment (PA): A subject can execute an operation only if the the subject’s role has been assigned the corresponding permission.

²<https://www.uml-diagrams.org/component-diagrams.html>

Roles may be ordered hierarchically (Partial Hierarchy = PH), inheriting permissions from one role to another. One subject might be assigned to multiple roles and one role might be assigned to multiple permissions. A permission for an operation can be assigned to more than one role. Equation 4.1 gives a simple formal description of these rules:

$$\begin{aligned} PA &\subseteq P \times R \\ RA &\subseteq S \times R \\ PH &\subseteq R \times R \end{aligned} \tag{4.1}$$

In this system, roles are bound to a conference instance and are authorized by the conferences' admin. A subject is authorized as an admin automatically by creating a new conference. Further we relax rule 1, by assuming that each account in the blockchain network is assigned to the default role *anyone*, which has only read permissions to conferences, papers and reviews. The mapping in Figure 4.3 describes the system's roles and permissions.

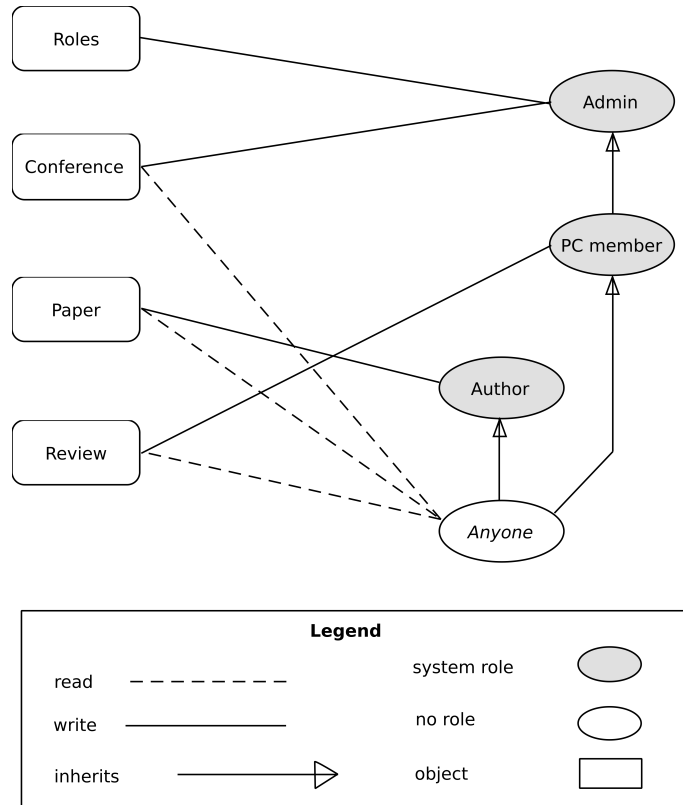


Figure 4.3: The systems roles, objects and permissions.

4.4 File Storage

A blockchain is not suited for file storage, because of its massive redundancy. However, this system requires to store several bigger pieces of data: Conference description, paper PDFs and review texts. Therefore, those items are stored off-chain, in the IPFS P2P file storage network.

IPFS is a stack of protocols based on BitTorrent, Git file-versioning and self-certified file systems, outlined in the whitepaper of J. Benet [51]. It aims to provide a flexible interface for modern, decentralized file sharing.

Each participant, namely an "ipfs node", has a Public Key Infrastructure (PKI) based identity. A node running on the participants computer can find, publish, and replicate objects in the network. Such a node is required to offer some local storage when *adding* an object. When peers request a file, its location is looked-up and it is stored locally for a limited period of time. However, permanent local storage is only achieved by using the *pin* method. In order to retrieve neighboring nodes' addresses and their offered files, IPFS uses a routing interface in form of a Distributed Hash Table (DHT) by default. A DHT caches routing information for file identifiers and defines efficient algorithms for look-up operations as well as protection against distributed denial-of-service attacks. An incentivization and punishment mechanism ensures balanced contribution of files and location records among the untrusted peers.

Files are uniquely identified by a *multihash* checksum on their content. A multihash is a self-describing, flexible hash format, which consists of a function code, digest size and the hash digest. For example one could choose the hash function SHA2-256 in base58 encoding. A valid multihash identifier may then look like this:

```
QmRZxt2b1FVZPNqd8hsikyDL3TdBDetSPX9Kv46HmX4Gx8
```

We embed IPFS file storage in our application by storing identifying metadata of each object on-chain: A conference's title, year and admin, a paper's title and author and a review's author and score. Additionally, each object stores a reference to the file on IPFS, namely the files' multihash. For demonstration purposes, those files are accessed through a public IPFS gateway that caches the data temporarily without requiring the user to keep an IPFS node running.

Chapter 5

Implementation

In this chapter, the implementation of the web interface, called *FakeChair* is described. Code examples for key functionalities in the back end will be shown and explained. See the GitLab repository¹ for an overview of front end development tools, frameworks, code sources and a full usage and deployment description locally as well as remote². GitLab is a platform for code versioning, distribution and documentation and thus better suited for further explanations than this document.

In the following sections, selected parts of the architecture are explained using code examples. In Section 5.1 some aspects of smart contracts are presented and Section 5.2 describes a JavaScript Object Notation (JSON) schema, used for form input validation. Further Section 5.3 shows an example on how to call a contract from a JavaScript front end application and finally Section 5.4 explains how files are added to IPFS.

5.1 Solidity Contracts

Solidity contracts are similar to classes in object oriented languages³. A short example is given using the Review.sol contract in Listing 5.1. It stores important meta-information on chain: The reviewer's 20-byte Ethereum address, an integer between -3 and 3 for the paper's score and the multihash of a JSON file containing a detailed review. All data and functions are marked with the *public* visibility specifier, thus it is possible to call functions and data from outside the contract.

```
1 pragma solidity ^0.4.21;  
2
```

¹<https://gitlab.gwdg.de/f.buschendorf/fakechair>

²<http://fakechair.fabiolabuschendorf.de>

³<http://solidity.readthedocs.io>

```
3 contract Review{
4
5     address public reviewer;
6     int8 public score;
7
8     // IpfsMultihash
9     bytes32 public digest;
10    uint8 public hashFunction;
11    uint8 public size;
12
13    function Review(address _reviewer, int8 _score, bytes32 _digest, uint8 _hashFunction,
14                    uint8 _size) public {
15        score = _score;
16        reviewer = _reviewer;
17        digest = _digest;
18        hashFunction = _hashFunction;
19        size = _size;
20    }
21 }
```

Listing 5.1: A simple contract. It stores review meta-information on the blockchain and includes a hash to an IPFS file which stores the review text.

Another example taken from `Conference.sol` in Listing 5.2 shows a way to inherit another contract's properties with the `is` keyword. This allows `Conference` to define and use roles and associated functions such as `hasRole()`. An important feature of Solidity contracts are *Events*, which provide logging facilities. Information emitted by an event is logged with a transaction on the blockchain. If a condition is not satisfied, the `require()` function will revert the transaction and make changes to the accounts state invalid.

```
1
2 contract Conference is RBACWithAdmin{
3
4     event PaperAdded (address author, bytes32 digest, uint8 hashFunction, uint8 size);
5
6     Paper[] paper;
7
8     [...]
9
10    modifier onlyAuthor(){
11        require(hasRole(msg.sender, ROLE_AUTHOR));
12        _;
13    }
14 }
```

```

15     function addPaper(address _author, bytes32 _title, bytes32 _digest, uint8
        _hashFunction, uint8 _size) onlyAuthor public {
16
17         paper.push(new Paper(_author, _title, _digest, _hashFunction, _size));
18         emit PaperAdded(_author, _digest, _hashFunction, _size);
19     }
20
21     [...]

```

Listing 5.2: Inheriting properties of another contract.

5.2 Defining Input Forms with JSON Schema

The JSON is an independent, lightweight data-interchange format, used to structure information to make it human- and machine-readable. Objects are defined by a collection of key-value pairs and are verified by a *schema*, a vocabulary that allows to annotate and validate JSON documents⁴. There are tools that create HTML forms and verify form-input data using such a schema⁵. This prototype uses a JSON schema for conference and review creation. The example below shows the schema for a review: It requires input information based on EasyChair's review format, which includes description of properties, types and constraints. Default values are used to easily create reviews during the application's development and testing.

```

schema = {
  "$schema": "http://json-schema.org/draft-06/schema#",
  "title": "Review",
  "description": "Review form.",
  "type": "object",
  "properties": {
    "PaperAddress": {
      "type": "string"
    },
    "PCMember": {
      "type": "string",
      "default": "Helga Examplewoman"
    },
    "Evaluation": {
      "type": "object",

```

⁴<http://json-schema.org/>

⁵<https://github.com/mozilla-services/react-jsonschema-form>

```

"properties": {
  "text": {
    "type": "string",
    "description": " Enter the text for the field
Overall evaluation below. This field is required.",
    "default": "Very good!"
  },
  "score": {
    "type": "integer",
    "minimum": -3,
    "maximum": 3,
    "description": "3 strong accept, 2 accept, 1 weak accept,
0 borderline paper, -1 weak reject,-2 reject, -3 strong reject",
    "default": 3
  }
}
},
"Confidence": {
  "type": "integer",
  "description": " 5 (expert), 4 (high), 3 (medium), 2 (low), 1 (none)",
  "minimum": 1,
  "maximum": 5,
  "default": 5
},
"Remarks": {
  "type": "string",
  "description": "Confidential remarks for the program committee",
  "default": "Nothing to say."
},
"Reviewer": {
  "type": "object",
  "properties": {
    "FirstName": {
      "type": "string",
      "default": "Luke"
    },
    "LastName": {
      "type": "string",

```

```

        "default": "Delegateman"
    },
    "E-Mail": {
        "type": "string",
        "default": "luke.delegateman@cool-university.com"
    },
}
}
},
"required": ["PaperAddress", "Reviewer", "Evaluation", "Confidence"]
};

```

5.3 Accessing Contracts with web3

The web3 JavaScript API provides methods to communicate with smart contracts of an Ethereum blockchain ⁶. It requires an RPC provider, which connects the application with the blockchain network. In this project, we use the add-on MetaMask⁷.

A compiled contract is specified by a JSON file which contains an Application Binary Interface (ABI) object, specifying functions and data getters and setters. The framework *Truffle*⁸ provides methods to compile, migrate and use this ABI in web projects. By combining both tools, we can fetch data like a conference's list of papers and display them in our web interface, see Listing 5.3.

```

1 import ConferenceContract from '../build/contracts/Conference.json'
2
3 [...]
4
5 // get conference ABI
6 const contract = require('truffle-contract')
7 let conference = contract(ConferenceContract);
8 conference.setProvider(this.state.web3.currentProvider)
9
10 const accounts = await this.state.web3.eth.getAccounts()
11
12 // get conference instance
13 const conferenceInstance = await conference.at(this.state.conferenceAddress)
14 // call function from Conference.sol
15 const paperLength = await conferenceInstance.getPaperLength({from: accounts[0]})
16

```

⁶<https://web3js.readthedocs.io/en/1.0/>

⁷<https://metamask.io>

⁸truffleframework.com

```

17 for (let i = 0; i < this.state.paperLength; i += 1) {
18   let paper = await conferenceInstance.getPaperByIndex(i, {from: accounts[0]})
19   paperAddresses[i] = paper[0];
20   cleanTitles[i] = cleanTitle(paper[2]);
21   hashes[i] = multihash.getMtHashFromContractResponse([paper[3].toString(), paper
22     [4].c[0].toString(), paper[5].c[0].toString()])
23   this.props.getConferenceForPapers(paperAddresses[i], this.state.conferenceAddress);
24 }
25 [...]

```

Listing 5.3: Using web3.js to retrieve contract data in `showPaper.js`.

5.4 Adding Files to IPFS

Adding files to IPFS requires using a JavaScript library and running a node with `ipfs daemon`. The function `add` will add files and data to IPFS, where the input data is a buffer instance or readable stream⁹. Listing 5.4 shows `addPaper.js`, in which a PDF file is converted into a buffer instance and added to IPFS. The `ipfsHash` is then split into its multihash parts and added to the conference contract instance.

```

1 import ConferenceContract from '../build/contracts/Conference.json'
2 import ipfs from './utils/ipfs';
3 import getWeb3 from './utils/getWeb3'
4 import multihash from './utils/multihash';
5
6 [...]
7
8 captureFile = (event) => {
9   event.stopPropagation()
10  event.preventDefault()
11  const file = event.target.files[0]
12  let reader = new window.FileReader()
13  reader.readAsArrayBuffer(file)
14  reader.onloadend = () => this.convertToBuffer(reader)
15 };
16
17 convertToBuffer = async(reader) => {
18   //file is converted to a buffer for upload to IPFS
19   const buffer = await Buffer.from(reader.result);
20   //set this buffer -using es6 syntax
21   this.setState({buffer});

```

⁹<https://github.com/ipfs/interface-ipfs-core/>


```

22     };
23
24     addPaper = async () => {
25         try{
26             this.setState({transactionHash: '... waiting'})
27
28             const contract = require('truffle-contract')
29             let conference = contract(ConferenceContract);
30             conference.setProvider(this.state.web3.currentProvider)
31
32             // Get accounts.
33             const accounts = await this.state.web3.eth.getAccounts()
34             // get conference instance
35             const conferenceInstance = await conference.at(this.state.conferenceAddress)
36             // Add file to IPFS
37             const ipfsHash = await ipfs.add(this.state.buffer)
38
39             this.setState({ipfsHash: ipfsHash[0].hash })
40             const { digest, hashFunction, size } = multihash.getBytes32FromMultiash(ipfsHash
41                 [0].hash);
42
43             // Add Paper to Conference Contract
44             const transactionHash = await conferenceInstance.addPaper(accounts[0], this.state.
45                 title , digest, hashFunction, size, { from: accounts[0] , gasLimit: 6385876})
46             this.setState({transactionHash: transactionHash.tx})
47         }
48         catch(error){
49             this.setState({transactionHash: 'Transaction failed. Only Authors can add papers
50                 to a conference.'})
51             console.error(error)
52         }
53     }
54 }

```

Listing 5.4: Adding a paper to a conference and storing the PDF file on IPFS. Code source: [52].

5.5 Challenges

Writing a decentralized application including a web interface involves the knowledge of many different frameworks and technologies. The main tasks while implementing this prototype were to

- identify most useful frameworks,
- find good code sources,
- gain experience with selected frameworks/ languages.

The main difficulty is, that programming languages and frameworks associated with the Ethereum blockchain are immature and fast developing. In reality,

- some frameworks are not maintained anymore (e.g. Ethereum `testrpc`),
- code is deprecated after short periods of time,
- this prototype had a fixed deadline after 12 weeks.

The most recent version of `web3.js`, for example, is 1.0, but the documentation warns:

"This documentation is work in progress and `web3.js` 1.0 is not yet released! You can find the current documentation for `web3 0.x.x` at github.com/ethereum/wiki/wiki/JavaScript-API." ¹⁰

However, `web3` version 1.0 has necessary features for this project. The same difficulty exists in `Truffle` version 4, which was released in October 2017 and features new commands and mechanisms which are not part of many tutorials written more than 8 months ago.

A general problem in writing JavaScript applications are the fast changing standards and the slow adaption of those by popular browsers ¹¹ and other frameworks. Modern JavaScript code is standardized by ECMA International and its 2015 release, ES6, provides many useful concepts like classes, `async/await`, promises, arrow functions, that are used in this project [53]. Up to date, one needs to include a *transpiler* that translates ES6 code to older JavaScript versions. In the end, using many different frameworks and a transpiler results in conflicts [54].

In `Solidity`, defining `events` is popular, but in fact they are problematic to use for logging purposes, or state-watching. An event is emitted, even if conditions of that contract are not met. In this case the EVM reverts all changes and makes the whole transaction without effect. However, it is the most important use case of events, to be watched and to perform actions as a reaction to success notifications. As a result, an event is emitted even if the transaction was not successful. Therefore this project does not rely on event messages from contracts.

¹⁰<https://web3js.readthedocs.io/en/1.0/>

¹¹<http://kangax.github.io/compat-table/es6/>

Chapter 6

Conclusion and Future Work

This chapter concludes this thesis and summarizes the results of this work. Section 6.1 answers the research questions of Section 1.1. Finally, Section 6.2 provides an outlook on future work.

This work describes the development of a minimum viable product of a peer-reviewing platform on the blockchain. The prototype is accessible online and the code is publicly available. It implements the following functionalities:

- Create conferences
- Specify detailed conference information
- Add authors and PC members to a conference
- Authors can upload papers
- PC members can create reviews
- All content can be read by anyone

Within these features, useful tools and frameworks for current application development on the Ethereum blockchain are identified and working code examples are provided.

6.1 Answering the Research Questions

The main research question "**How to design a blockchain based peer-reviewing system?**" is answered in three sub-questions.

RQ1: What are its functional requirements? This thesis proposes a fundamental requirements engineering process, in which the systems functionalities are defined using the free online conference management platform *EasyChair*. Its main actors, namely chairs, reviewers and authors are identified and their goals are specified in an AOM Goal Model.

RQ2: Which existing blockchain platform is suitable? An analysis is dedicated to identify the best suited blockchain platform for our system. Main criteria were, among others, the community support for application development, the presence of a free test network, a possibly mature technology, a large number of full nodes and existing open source projects. We found the Ethereum testnet Rinkeby to be best suited.

RQ3: What is the system architecture? The systems architecture is divided into a back end (smart contracts on the Rinkeby test network), a file database (IPFS: P2P file storage) and a front end made with JavaScript and react.js. Access control is organized with a Role-based Access Control interface, implementing the roles identified within RQ1.

6.2 Future Work

When comparing the implemented features and the functional requirements analyzed in Section 3.1, following features are missing:

- Create identifiable accounts
- Match paper to a reviewer
- Rate content
- Send messages within the system

The systems needs a module which registers a user with her real name and associates an academic identity with that account. One could link a unique user identification number that serves as a key for a real world name and a communication address. This number could possibly be an ORCID. Within that module pseudonymisation and anonymization methods could be implemented to allow flexibility in review methods [55].

Another module should offer different paper-to-reviewer matching processes, copied from the model platform EasyChair. Possible options to choose are: Open reviewing, where no matching process is required, a randomized assignment, that takes care of conflicts of interest or paper bidding, where reviewers select preferred papers.

Further, a module could provide mechanisms to rate content and reward. This rating system could incentivize users to create high-quality reviews and discourage spammer by giving a reward in form of a token. This token could work like an internal currency that is used to pay for system services.

Finally different kinds of messages, warnings, reminders and success notifications from the chair to all conference members or a selected group of authors or reviewers should be enabled. For example, an author should receive notifications for an accepted or rejected paper and all conference members should receive reminders of important deadlines.

Bibliography

- [1] F. Gannon, "The essential role of peer review." *EMBO Reports*, vol. 2(9), no. 743, 2001.
- [2] N. Z. Zeitung, "Springer-Verlag muss 16 Nonsens-Artikel löschen," URL: <https://www.nzz.ch/wissenschaft/springer-verlag-muss-16-nonsens-artikel-loeschen-1.18253386>, 2014, (Accessed March 28, 2018).
- [3] S. Schleim, "Telepolis: Zitier mich oder vergiss es! ," URL: <https://www.heise.de/tp/features/Zitier-mich-oder-vergiss-es-3867294.html>, 2017, (Accessed March 28, 2018).
- [4] H. Maier-Borst, "Deutschlandfunk: Warum Elsevier das "Forscher-Facebook" Researchgate verklagt," URL: http://www.deutschlandfunk.de/open-access-warum-elsevier-das-forscher-facebook.676.de.html?dram:article_id=399672, 2017, (Accessed March 28, 2018).
- [5] M. Spearpoint, "A proposed currency system for academic peer review payments using the blockchain technology," *Publications*, vol. 5, no. 3, 2017.
- [6] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 276–294.
- [7] Steem, "Steem - An incentivized, blockchain-based, public content platform," URL: <https://steem.io/steem-whitepaper.pdf>, 2018, (Accessed June 20, 2018).
- [8] A. D. Michael Wolf, Marlies Wiegand, "PEvO - Decentralized Open Access and Evaluation," URL: https://pevo.science/files/pevo_whitepaper.pdf, 2016, (Accessed June 20, 2018).
- [9] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, April 2017, pp. 243–252.
- [10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." URL: <https://bitcoin.org/bitcoin.pdf>, 2008, (Accessed November 3, 2017).

- [11] G. Wood, "Ethereum Yellowpaper: Ethereum: A secure decentralized generalised transaction ledger," URL: <http://gavwood.com/paper.pdf>, 2014, (Accessed June 10, 2018).
- [12] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ, USA: Princeton University Press, 2016.
- [13] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st ed. O'Reilly Media, Inc., 2014.
- [14] V. Buterin, "Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform," URL: <https://github.com/ethereum/wiki/wiki/White-Paper>, 2013, (Accessed June 10, 2018).
- [15] Ethereum, "Wiki: Development Tutorial," URL: <https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial>, 2018, (Accessed June 13, 2018).
- [16] Elsevier, "What is peer review?" URL: <https://www.elsevier.com/reviewers/what-is-peer-review>, 2018, (Accessed May 23, 2018).
- [17] C. Analytics, "Impact factor: Journal citation report," URL: <http://jcr.incites.thomsonreuters.com/JCRJournalHomeAction.action>, 2017, (Accessed May 23, 2018).
- [18] EasyChair, "EasyChair Help Section," URL: <https://easychair.org/help/>, 2018, (Accessed March 30, 2018).
- [19] R. Smith, "Peer review: a flawed process at the heart of science and journals," *Journal of the royal society of medicine*, vol. 99, no. 4, pp. 178–182, April 2006.
- [20] S. P. Horst Hippler, "Wissenschaftliche Publikationen: "Wir streiten darüber, was das Lesen kosten soll"," URL: http://www.deutschlandfunk.de/wissenschaftliche-publikationen-wir-streiten-darueber-was.680.de.html?dram:article_id=377165, 2017, (Accessed June 16, 2018).
- [21] A. Herbold, "Showdown zwischen Bücherregalen," URL: <https://www.tagesspiegel.de/wissen/streit-um-teure-wissenschaftsjournale-showdown-zwischen-buecherregalen/19412772.html>, 2017, (Accessed May 23, 2018).
- [22] K. T. Leon S. Sterling, *The Art of Agent-Oriented Modeling*. MIT Press, July 2009.
- [23] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," 12 2017.
- [24] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, ser. Addison-Wesley Signature Series (Fowler). Pearson Education, 2010. [Online]. Available: <https://books.google.de/books?id=6ADDuzere-YC>

- [25] Steemit, "Steem FAQ," URL: <https://steemit.com/faq.html>, 2018, (Accessed April 4, 2018).
- [26] S. Nakamoto, "Forum post: Bitcoin open source implementation of P2P currency," URL: <http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source>, 2009, (Accessed April 5, 2018).
- [27] blockchain.info, "Bitcoin statistics and data," URL: blockchain.info/charts, 2018, (Accessed April 6, 2018).
- [28] Bitnodes, "Bitcoin nodes," URL: <https://bitnodes.earn.com/dashboard/?days=90>, 2018, (Accessed March 17, 2018).
- [29] A. Back, "Hashcash - A Denial of Service Counter-Measure," URL: <http://www.hashcash.org/hashcash.pdf>, 2002, (Accessed April 5, 2018).
- [30] Digiconomist, "Bitcoin Energy Consumption," URL: <https://digiconomist.net/bitcoin-energy-consumption>, 2018, (Accessed April 5, 2018).
- [31] M. Bevand, "Bitcoin Energy Consumption," URL: <http://blog.zorinaq.com/bitcoin-electricity-consumption/>, 2018, (Accessed April 5, 2018).
- [32] "GitHub repository Bitcoin," URL: <https://github.com/bitcoin/bitcoin>, 2018, (Accessed April 5, 2018).
- [33] bitcoinfees.info, "Bitcoin fees," URL: <https://bitcoinfees.info/>, 2018, (Accessed April 5, 2018).
- [34] Coinwarz, "Ethereum Network Hashrate," URL: <https://www.coinwarz.com/network-hashrate-charts/ethereum-network-hashrate-chart>, 2018, (Accessed April 5, 2018).
- [35] Ethernodes, "The Ethereum Nodes Explorer," URL: <https://www.ethernodes.org/network/1>, 2018, (Accessed April 5, 2018).
- [36] etherscan.io, "Ethereum pending transactions queue," URL: etherscan.io/chart/pendingtx, 2017, (Accessed November 14, 2017).
- [37] E. community, "Ethereum Homestead Documentation," URL: <http://www.ethdocs.org/en/latest>, 2018, (Accessed April 8, 2018).
- [38] "GitHub repository Ethereum," URL: <https://github.com/ethereum/go-ethereum>, 2018, (Accessed April 5, 2018).
- [39] E. Wiki, "Releases," URL: <https://github.com/ethereum/wiki/wiki/Releases>, 2018, (Accessed April 8, 2018).
- [40] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in *International Workshop on Open Problems in Network Security (iNetSec)*, ser. Open Problems in Network Security, J. Camenisch and D. Kesdoğan, Eds., vol. LNCS-9591,

- Zurich, Switzerland, Oct. 2015, pp. 112–125, part 4: Cryptography. [Online]. Available: <https://hal.inria.fr/hal-01445797>
- [41] BitInfoCharts, “Ethereum fees,” URL: <https://bitinfocharts.com/de/comparison/ethereum-transactionfees.html>, 2018, (Accessed April 5, 2018).
- [42] Drakos, “Top 400 witnesses,” URL: <https://steemian.info/witnesses>, 2018, (Accessed April 10, 2018).
- [43] N. Wack, “Steem seed nodes,” URL: <https://status.steemnodes.com/>, 2018, (Accessed April 10, 2018).
- [44] T. Cliff, “Steemit Roadmap,” URL: <https://steemit.com/roadmap2018/@timcliff/steemit-roadmap-2018-timcliff-s-top-picks-and-recommendations>, 2017, (Accessed April 10, 2018).
- [45] IOTA, “IOTA Docs,” URL: <https://docs.iota.org/>, 2018, (Accessed April 9, 2018).
- [46] R. Semko, “IOTA Github Organization,” URL: <https://github.com/iotaledger/>, 2018, (Accessed April 9, 2018).
- [47] IOTA, “IOTA: Tangle growth update January 2018,” URL: <https://medium.com/deviota/iota-tangle-growth-update-january-2018-161b78e09ca9>, 2018, (Accessed April 9, 2018).
- [48] R. Rottmann, “About Smart Contracts in IOTA,” URL: <https://medium.com/@ralf/about-smart-contracts-in-iota-626d2bd3619e>, 2018, (Accessed April 9, 2018).
- [49] Ethereum, “Wiki: Proof-of-Stake FAQ,” URL: <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ>, 2018, (Accessed April 9, 2018).
- [50] R. K. David Ferraiolo, “Role-based access controls,” *Proceedings of the 15th National Computer Security Conference (NCSC)*, pp. 554–563, 1992. [Online]. Available: <https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1992/10/13/role-based-access-controls/documents/ferraiolo-kuhn-92.pdf>
- [51] J. B. David Dias, “IPFS Architecture Overview on GitHub,” URL: <https://github.com/ipfs/specs/tree/master/architecture>, 2017, (Accessed June 10, 2018).
- [52] G. mcchan1, “Simple Ethereum + InterPlanetary File System (IPFS)+ React.js DApp,” URL: <https://github.com/mcchan1/eth-ipfs>, 2018, (Accessed June 17, 2018).
- [53] B. Morelli, “JavaScript - WTF is ES6, ES8, ES 2017, ECMAScript... ?” URL: <https://codeburst.io/javascript-wtf-is-es6-es8-es-2017-ecmascript-dca859e4821c?gi=8e774ea85ffb>, 2017, (Accessed June 16, 2018).
- [54] Github, “Truffle boxes: Upgrade to web3 1.0 and ES6 , Issue 53,” URL: <https://github.com/truffle-box/react-box/issues/53>, 2018, (Accessed June 16, 2018).

- [55] C. Cap, "Digital Disruptions and Academic Research Processes," 2018, unpublished article.

