# Design of a Smart Contract Based Autonomous Organization for Sustainable Software

Alper Alimoğlu and Can Özturan
Department of Computer Engineering
Bogazici University, Istanbul, Turkey
emails: {alper.alimoglu,ozturaca}@boun.edu.tr

***Abstract*—The emerging blockchain technologies have enabled development of crypto-currencies and autonomous smart contracts that can operate in decentralized and trustless settings. Distributed autonomous organizations can be implemented using smart contracts available on the Ethereum blockchain. In this paper, we propose a distributed autonomous software organization model and its Ethereum smart contract implementation called AutonomousSoftwareOrg for providing a continuously operating virtual organization for software development communities and users. AutonomousSoftwareOrg facilitates a funding mechanism based on crypto-currencies, a decision making mechanism based on voting and record keeping for software usage citations and executions. AutonomousSoftwareOrg is deployed and tested on our local Ethereum based blockchain system (http://ebloc.cmpe.boun.edu.tr). Its Solidity language source code is available at https://github.com/ebloc/AutonomousSoftwareOrg.**

## I. Introduction

The emerging blockchain technologies have applications in many areas such as finance, governance, economy, citizen science, E-science and Internet of Things. It was first developed in order to implement the Bitcoin digital crypto-currency [1]. Later, with the introduction of Ethereum [2, 3], a virtual machine that can run programs called *smart contracts* have been provided. A blockchain platform runs continuously on a peer-to-peer (P2P) network to which anyone can connect. Blockchain keeps immutable records of transactions, acts as a notary and pays incentives for running its software as a node. Merkle [4] provides an informal list of the features of the Bitcoin blockchain: (i) non-stoppable, (ii) non-interruptible, (iii) acts as a notary, (iii) keeps public records, are some of its important features.

For efficiency reasons, each block contains a number (group) of transactions, a fingerprint (hash) of the previous block which forms a hash link to the previous group and a timestamp. Blocks and their hash linked structure form the blockchain. Blocks are inserted to the main blockchain by going through a consensus process. The process of creating valid blocks is called mining. This requires demonstrating proof of work. The miners are rewarded with crypto-currency of the blockchain for performing this work. Blocks are mined around every 10 minutes on average on the Bitcoin blockchain. On Ethereum, block time has been averaging around 15 seconds [5].

In eScience, a new software development usually occurs within the life cycle of a funded project. The software is usually uploaded to a public code repository. During the project duration, there are funded developers who maintain the software. But once a project is over, and if the project team is not granted a new follow-up project, maintenance and further development of the software is at the mercy of volunteer developers. There have been successful developer communities that have been formed this way for some software. But there is also a great risk that follow-up communities may not be formed and all the effort that goes into development may be wasted because there is no longer anyone to maintain and further develop the software. This paper addresses mainly this problem, i.e., can we develop a non-stoppable virtual organization that will: (i) represent the software, (ii) continue to exist even after the associated projects are over, (iii) provide decision making, crowd funding and citation mechanisms to its developers?

Before the emergence of blockchain and smart contract technology, one approach for achieving software sustainability would involve establishing a company and selling the software or the support which would in turn provide funding for further developments. This, however, requires a lot of capital and bureaucratic work. This approach may work if a massive market exists for the software. However, for open source and free software developed by communities, establishment of a company may not be a feasible approach. On the other hand, a virtual organization that runs as a smart contract can be viable alternative.

Smart contracts run exactly as programmed and autonomously, hence they can be used to implement distributed autonomous organizations (DAOs). This paper proposes blockchain and smart contract based solutions on the following topics about software sustainability:

1) *Development and Community:* A distributed autonomous software organization model and its Ethereum smart contract implementation is contributed for providing a non-stop virtual organization for software community with a funding mechanism based on crypto-currencies and a decision making mechanism based on voting.
2) *Professionalization:* Since blockchains are open public data; records of funding, membership in software organization, usages and citations of software by others are kept in one place (blockchain) and updated continuously (around every 15 seconds on average of block time on Ethereum). Funding agencies and employers can

978-1-5386-2686-3/17 $31.00 © 2017 IEEE
DOI 10.1109/eScience.2017.76

monitor the blockchain for impact, hence use this data for giving incentives, promotions and grants. This in turn leads to a higher quality sustainable software since the efforts of developers are better assessed quantitatively using the blockchain data.

3) *Credit:* In addition to providing a mechanism for receiving crypto-currency donations, citations by paper authors as well as records of software usages by other softwares can also be recorded on the blockchain. Hence, a novel blockchain based new credit and citation ecosystem can be implemented.
4) *Software publishing:* Blockchains are accessible by everyone operating an Ethereum node. Software related activities such as formation of autonomous organizations, announcement of new versions, funding records, citations and others can be recorded on the blockchain and announced as events. If standardized smart contract for softwares and interfaces are developed, then these can help software discoverability and reuse.
5) *Software reproducibility:* Conferences, journals and funding agencies can require software execution records to be stored on the blockchain. Hashes of inputs and outputs can be recorded on the blockchain. Since blockchain is immutable and can act as a notary, these records are in some way declarations by the software executers that given the specific input, the corresponding output should be reproduced.

In the rest of the paper, we first present an overview of the related work in Section II. Blockchain based autonomous software organization model is covered in Section III. Details of the smart contract that implements the software organization model are given in IV. Demonstration of the smart contract on our local Ethereum based blockchain system (http://ebloc.cmpe.boun.edu.tr) is presented in Section V. Finally, the paper is concluded with a discussion and future work in Section VI.

## II. Previous Work

Crowdsourced development efforts are becoming very important and are leading to development of various successful platforms such as OpenStreetMap, Instagram, Weather Underground and Kickstarter. Software development and documentation [6] are also being outsourced to communities. A survey of the use of crowdsourcing in software engineering is given in [7]. Blockchain technologies are the new trend in obtaining crowdfunding for new projects. Whereas earlier crowdsourcing and funding efforts have been based on centralized services, the newly emerging Blockchain technologies that operate in decentralized and trustless manner enable crypto-currencies to be used for crowdfunding. Under the name Initial Coin Offering (ICO) that replaces the traditional Initial Public Offering (IPO), companies can use smart contracts on the Ethereum blockchain to collect funding from crowds.

The first Distributed Autonomous Organization on the Ethereum platform called The DAO was deployed in Spring 2016 [8]. During its ICO, millions of dollars worth of Ether were collected. Unfortunately, due to a recursive Ethereum send exploit, a hacker drained millions of dollars worth of Ether from the contract [9]. The DAO hack taught a important lesson about storage of value and doing transactions on blockchains and that is, what we should be very careful about correctness of the smart contracts deployed on the blockchain. We should verify their correctness and test them thoroughly.

The DAO smart contract implemented a virtual organization with rules to coordinate decision making by voting of the investors. Project proposals could be made by the investors and their funding could be decided by voting.

Autonomous Software Organization (called *AutonomousSoftwareOrg*) that we implement in this paper is similar to The DAO in spirit but is much simpler and is meant for small software development communities. AutonomousSoftwareOrg also provides a mechanism to collect funds and select and fund software development proposals. The importance of distributed collaboration and doing science on the Internet has been noted by [10]. AutonomousSoftwareOrg have the potential to facilitate sustainable collaborations for software development since it will present a living, non-stoppable, non-interruptible contract that provides fund collection and delivery as well as a democratic mechanism for decision making by members. Since software can be characterized as being a hidden infrastructure behind the world's largest scientific facilities [11], a smart contract running continuously on a blockchain can be the autonomous entity that represents the software infrastructure.

The topics of software citation for credit, identification, discovery, and reuse has also gained importance recently [12]. Our AutonomousSoftwareOrg smart contract provides functions to record citations, usages and software executions on the blockchain. If different entities such as funding agencies, users, researchers, evaluators, companies are represented as addresses on the Ethereum blockchain, then our AutonomousSoftwareOrg can receive citations and usage links from them and record them as transactions on the public blockchain. Hence, continuously updated rich linkage information among various entities shown in Figure 1 can be made publicly available on the blockchain.

When a function of a smart contract is invoked, each one of the thousands of nodes in the P2P network that executes the body of the function and then later participates in the consensus process. Hence, if massive number of users all of the world are to use the Ethereum blockchain, scalability problem needs to be resolved. This issue has been raised in [13, 14] and solution proposals discussing the use of sharded blockchains are available [15]. We note that even though in the world of finance, massive number of transactions may occur every second. It is critical that these are recorded in a timely manner on the blockchain. This is not, however, the case for our AutonomousSoftwareOrg. Activities such as becoming a member, donation, voting and citation that will be supported by AutonomousSoftwareOrg do not happen frequently, i.e. large intervals of time elapse between such activities. Hence,

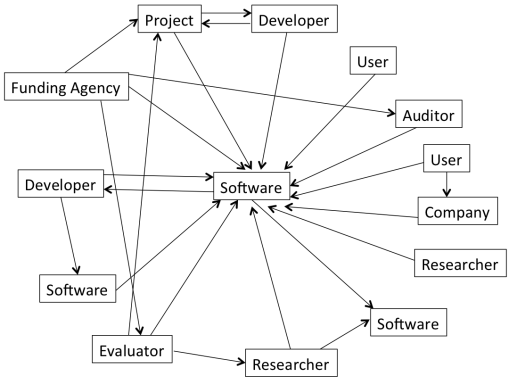


Fig. 1. Interactions of various entities

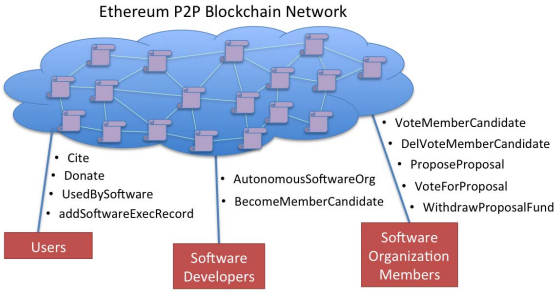


Fig. 2. Blockchain based autonomous organization model

if any transaction associated with such activities gets delayed in getting into a confirmed block, it will not cause critical problems.

## III. Blockchain Based Autonomous Organization Model

Distributed Autonomous Software Organization that has been designed in this paper aims to provide a virtual organization for the developers of a software, allows them to collect funding for the development of the software and makes decisions by a voting process. Decisions are taken by at least M/N majority rule which is configured when the virtual organization is first created. At least M/N of the members of the organization must vote affirmatively in order for a decision to pass. Figure 2 depicts the design of our autonomous organization. Our system consists of four main entities:

1) *Users:* use the software developed by the autonomous organization. If a user likes the software, he can invoke *Donate*() function to donate money to the organization. Companies can also use *Donate*() to provide monetary grants to the software developers. If a user is a researcher who publishes in journals or conferences, he can cite the software by calling the *Cite*() function and providing the doi number of the publication. If the user is a developer of another software which is managed by another smart contract, he can call *UsedBySoftware*() to record the address of the smart contract for the other software. *UsedBySoftware*() is similar to *Cite*() but it forms a linkage between two software using smart contracts whereas *Cite*() forms linkage between a publication and the software using smart contract. Users can also use *addSoftwareExecRecord*() function to record and notarize hashes of input and output files. This can be useful for reproducibility checks.
2) *Software Developers:* are the candidates who want to contribute development efforts to the software within the organization. The smart contract for the software organization is first created (deployed) by a single developer who calls the constructor function *AutonomousSoftwareOrg*(). This person (who deploys the contract) becomes the first member of the organization. The candidates express their candidacy by calling the *BecomeMemberCandidate*() function and provide information about themselves. The subsequent members are determined by a voting process carried out by existing members.
3) *Members of the Software Organization:* are the software developers who are officially affiliated with the software organization and who have voting rights. They are able to propose development proposals and ask for funding from the organization by calling the *ProposeProposal*(). If there are software developer candidates, they can also call the *VoteCandidateMember*() to vote for candidates to make them a member. They can call the *DelVoteCandidateMember*() to delete their vote for a candidate or a member. If voting majority is reached, a candidate becomes a member. If voting majority is lost, a member loses his membership. Members use voting to get funding for software development proposal. They vote using the *VoteForProposal*() function. If voting majority is reached for a proposal, the proposer member can withdraw funding by calling *WithdrawProposalFund*().
4) *AutonomousSoftwareOrg Smart Contract:* is available on the Ethereum blockchain network and works autonomously when its functions are invoked. It is implemented as Solidity Language code and contains the operation rules of the software organization. It has the rules for majority voting decisions for membership and proposal funding. It also has various getter and event functions that provide information about the number of votes, proposals and members.

## IV. Smart Contract Details

Figure 3 shows the signatures of the functions available in the AutonomousSoftwareOrg smart contract. Solidity language [16] is used to code the smart contract. Solidity language provides conditional modifiers that can apply to function bodies. We make use of Condition Oriented Programming (COP) style proposed by [17]. Function bodies are executed

when the conditional modifiers are true. The descriptions of conditional modifiers are given in Table I. The full source code of the AutonomousSoftwareOrg smart contract is available at https://github.com/ebloc/AutonomousSoftwareOrg.

We note that, in addition to the contract functions shown in Figure 2, getter functions (functions that return the values of various variables) are provided. These getter functions are defined as constant functions.

In the AutonomousSoftwareOrg contract, events are also defined and emitted when transactions such as proposal submissions or voting are performed. Events can be watched by anyone who runs an Ethereum client such as Geth or Parity.

TABLE I
CONDITIONAL MODIFIERS IN THE SMART CONTRACT

| Modifier | Condition |
|---|---|
| `enough_fund_balance(propno)` | there is enough balance to pay for proposal *propno* funding |
| `valid_proposal_no(propno)` | *propno* is a valid proposal number |
| `member(addr)` | *addr* is the Ethereum blockchain address of a member |
| `not_member(addr)` | *addr* is not the Ethereum blockchain address of a member |
| `valid_member_no(memberno)` | *memberno* is a valid member/candidate number |
| `valid_deadline(deadline)` | *deadline* is a valid block number |
| `within_deadline(propno)` | deadline for proposal *propno* has not passed |
| `not_voted_for_proposal(propno)` | function invoker did not vote for proposal *propno* |
| `not_voted_for_member(memberno)` | function invoker did not vote for member with *memberno* |
| `voted_for_member(memberno)` | function invoker did not vote for member with *memberno* |
| `proposal_owner(propno)` | function invoker is the owner of proposal *propno* |
| `proposal_majority(propno)` | at least M/N of the members voted affirmatively for the proposal *propno* |
| `membership_majority(memberno)` | at least M/N of the members voted affirmatively about membership of *memberno* |
| `nonzero_payment_made()` | non-zero payment is also sent to the contract when the function is invoked |

Ethereum operates in *"pay to play"* manner. When calling contract functions, ether money (called gas) is charged in order to execute the instructions of the functions. The unit of currency Ether on the Ethereum blockchain also has smaller denominations. For example, 1 Ether is $10^{18}$ Wei. When a contract function is called, the global variable *msg.sender* contains the address of the caller. The global variable *msg.value* contains the amount of payment received together with the contract call. Note that in order for a contract function to be able to receive payment, it should be declared as *payable* function. As time units, we avoid using timestamps on the blockchain since these may be inaccurate. Instead, we use block numbers for setting and checking deadlines. Constant functions do not modify smart contract state variables; they

```
Contract AutonomousSoftwareOrg {

  ....  // Variable definitions

  ....  // Event definitions

  ....  // Conditional modifier definitions

  function AutonomousSoftwareOrg(bytes32 name,uint8 M,
     uint8 N, bytes32 url)

  function ProposeProposal(bytes32 title,bytes32 url,
     uint256 prophash,uint requestedfund, uint deadline)
     public member(msg.sender) valid_deadline(deadline)

  function VoteForProposal(uint propno) public
     valid_proposal_no(propno) within_deadline(propno)
     member(msg.sender) not_voted_for_proposal(propno)

  function WithdrawProposalFund(uint propno)  public
     valid_proposal_no(propno) within_deadline(propno)
     member(msg.sender) enough_fund_balance(propno)
     proposal_owner(propno) proposal_majority(propno)

  function BecomeMemberCandidate(bytes32 url) public
     not_member(msg.sender)

  function VoteMemberCandidate(uint memberno) public
     valid_member_no(memberno) member(msg.sender)
     not_voted_for_member(memberno)

  function DelVoteMemberCandidate(uint memberno) public
     valid_member_no(memberno) member(msg.sender)
     voted_for_member(memberno)

  function Donate() payable public nonzero_payment_made

  function Cite(bytes32 doinumber) public

  function UsedBySoftware(address addr) public

  function addSoftwareExecRecord(bytes32 softwareversion,
     bytes32 url,uint256 inputhash,uint256 outputhash)

  function addSoftwareVersionRecord(bytes32 url,
     bytes32 version,uint256 sourcehash)

  function getAutonomousSoftwareOrgInfo()
     constant returns (bytes32,uint,uint,uint,uint)

  function getMemberInfoLength() constant returns (uint)

  function getMemberInfo(uint memberno)
     member(membersinfo[memberno-1].memberaddr)
     constant returns (bytes32,address,uint)

  function getProposalsLength() constant returns (uint)

  function getProposal(uint propno) constant
     returns (bytes32,bytes32,uint256,uint,uint,bool,uint)

  function getDonationInfo(uint donationno) constant
     returns (address,uint,uint)

  function getCitation(uint citeno) constant
     returns (bytes32)

  function getUsedBySoftware(uint usedbysoftwareno)
     constant returns (address)

  function getSoftwareExecRecord(uint32 id) constant
     returns(address,bytes32,bytes32,uint256,uint256)

  function getSoftwareVersionRecords(uint32 id)
     constant returns(address,bytes32,bytes32,uint256)

  ... // the remaining getter functions
}
```

Fig. 3. AutonomousSoftwareOrg smart contract function signatures

just read values. Since, they are not transactions, they incur no gas cost.

## V. Tests

AutonomousSoftwareOrg is tested by deploying it on our Ethereum based local blockchain network (http://ebloc.cmpe.boun.edu.tr) that has been operational since October 2016. The gas cost of AutonomousSoftwareOrg smart contract deployment was 2235815 Wei. Anyone can use the eBloc platform by following the instructions at https://github.com/ebloc/eBloc. Figure 4 shows the screen dump of eBlocXplore tool that reports the transactions while testing AutonomousSoftwareOrg.

The gas spent for each smart contract function executions are given in Table II. These values are obtained from transaction receipts by calling getTransactionReceipt(txid) in Ethereum clients Geth or Parity.

TABLE II
Gas consumption of smart contract functions.

| Method | Used Gas(wei) |
|---|---|
| `ProposeProposal()` | 146086 |
| `VoteForProposal()` | 62997 |
| `WithdrawProposalFund()` | 41286 |
| `BecomeMemberCandidate()` | 77834 |
| `VoteMemberCandidate()` | 89504 |
| `DelVoteMemberCandidate()` | 22409 |
| `Donate()` | 90000 |
| `Cite()` | 47268 |
| `UsedBySoftware()` | 63546 |
| `addSoftwareExecRecord()` | 85078 |
| `addSoftwareVersionRecord()` | 80003 |

Our test scenario is as follows: The contract is deployed by one developer, who registers ten candidate members into AutonomousSoftwareOrg smart contract using the BecomeMemberCandidate() function. Then, the deployer of the contract uses VoteMemberCandidate() to vote for one randomly selected new candidate, who become a valid member. Later, valid members repeat this process for candidate members until all candidates become valid members. Multiple users donate some ether into AutonomousSoftwareOrg smart contract using the Donate() function. Two donation transactions of 10 ether values are shown in Fig 4. A randomly selected member proposes a proposal using the ProposeProposal() requesting 10 ethers for funding and setting a deadline. Members use VoteForProposal() to vote for the proposal until the required majority is reached. Voting for proposal can be watched by using *LogProposalVote*() event function. After the deadline passes the current block number, proposal proposer withdraws his requested fund using the *WithdrawProposalFund*() function.

## VI. Discussion and Conclusion

If an autonomous software organization like the AutonomousSoftwareOrg proposed in this paper is deployed on the Ethereum blockchain, the software can then become a living entity represented by its deployed non-stoppable smart

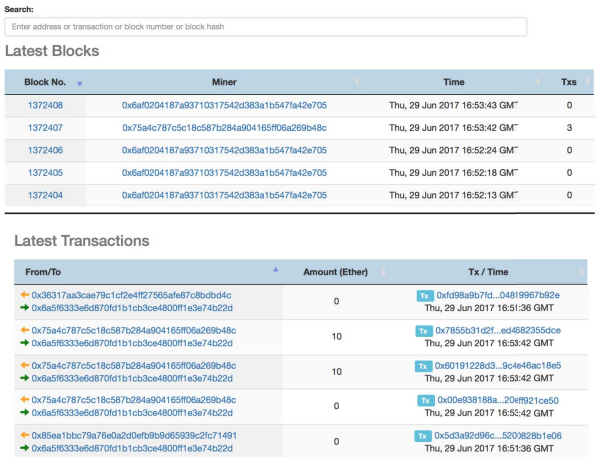


Fig. 4. Screen dump of eBlocXplore tool that reports the transactions while testing AutonomousSoftwareOrg

contract. What is more, it can collect funds and distribute funds to existing as well as new developers who become members of the autonomous software organization. This can then help the software to sustain itself. AutonomousSoftwareOrg can also record usage citations and execution records of the software which can form a valuable data that can be analyzed for software impact assessment. The fact that this data is replicated on thousands of blockchain P2P nodes further guarantees that the data is widely available, is notarized and continuously updated. All of these capabilities are not available in centrally managed services.

A web interface using web3.js library will be available in future versions of AutonomousSoftwareOrg. The topic of setting up software seal of approvals and software assessment frameworks are also becoming important [18]. In future AutonomousSoftwareOrg versions, we will provide functions that will record seal of approvals and results of assessment frameworks.

## Acknowledgement

This work is supported by the Turkish Ministry of Development under the TAM Project number DPT2007K120610.

## References

[1] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.
[2] Ethereum project. https://www.ethereum.org/.
[3] G. Wood. Ethereum: a secure decentralised generalised transaction ledger, homestead revision. http://gavwood.com/paper.pdf.
[4] R. Merkle, "Daos, democracy and governance,alcor, www.alcor.org," vol. 37:4, pp. 28–40, 2016, http://merkle.com/papers/DAOdemocracyDraft.pdf.
[5] Ethereum block time history. https://etherscan.io/chart/blocktime.
[6] A. Pawlik, J. Segal, H. Sharp, and M. Petre, "Crowdsourcing scientific software documentation: a case study of the numpy documentation project," *Computing in Science & Engineering*, vol. 17, no. 1, pp. 28–36, 2015.
[7] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *Journal of Systems and Software*, vol. 126, pp. 57–84, 2017.

[8] C. Jentzsch. (2016) Decentralized autonomous organization to automate governance. Https://download.slock.it/public/DAO/WhitePaper.pdf.
[9] P. Daian. (2016) Analysis of the dao exploit. Http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/.
[10] R. T. Kouzes, J. D. Myers, and W. A. Wulf, "Collaboratories: Doing science on the internet," *Computer*, vol. 29, no. 8, pp. 40–46, 1996.
[11] N. Chue Hong, "Software: the hidden infrastructure behind the world's largest scientific facilities," *Innovation Into Success - The Quarterly Journal of UKSPA*, vol. 31, pp. 28–29, 1 2013.
[12] K. E. Niemeyer, A. M. Smith, and D. S. Katz, "The challenge and promise of software citation for credit, identification, discovery, and reuse," *ACM Journal of Data and Information Quality (JDIQ)*, vol. 7, no. 4, p. 16, 2016.
[13] Eip 105 (serenity): Binary sharding plus contract calling semantics 53. https://github.com/ethereum/EIPs/issues/53.
[14] Horizontal scalability/sharding 142. https://github.com/EOSIO/eos/issues/142.
[15] On sharding blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQ.
[16] Solidity. http://solidity.readthedocs.io/en/latest/.
[17] G. Wood. Condition oriented programming. https://blog.ethcore.io/condition-oriented-programming-2/.
[18] N. Chue Hong, "Setting up a software seal of approval," 3 2017, https://figshare.com/articles/Setting-up-a-Software-Seal-of-Approval/4737178/1.