UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A project report submitted for the award of

Supervisor: Timothy Norman
Examiner: Timothy Norman

Multi Agent Systems on the BlockChain

by Jake Labelle

April 15, 2018

Contents

Chapter 1

Project Problem and Goals

1.1 Problem

Most university clubs collect registration fees to finance their activities for the year. The current solution for collecting the fees is either with cash or online. Both have the problem that the money is collected by one entity, and cash has the added negative that the payment could be repudiated. Then at the annual general meeting (AGM) , members vote for representatives and for managers. The problem with this is that the AGM is often held before the start of semester one, this means that many members and first years, who are a large proportion of the club members, are unable to vote.
After representatives and managers are chosen, they decide on several places the budget can be spent and how much money each of those places get. The disadvantage with this is that deciding where the money should go can take long time. Another problem with the negotiation phase is that quiet individuals may be bullied into making concessions by someone more confident. Lastly the final budget may not be known by normal members, making it unaccountable and opaque.

1.2 Goals

For the collection of the fees, I want my solution to have these characteristics. First off, all registrations are non repudiable. This means that there is a method for confirming the user has paid the fee. Also we can check the authenticity of a registration. This means that we a user can identify himself as the one who paid

the registration. Another feature is transparency, everyone should be able to see how much money has been collected from fees. Lastly until the budget has been decided no one should be able to access the money.
For the voting, the features are as follows. First off all members who have paid their registration fees should have equal voting power and the ability to apply as a representative. Another feature is the ability for a user to submit a full ranking of representative to allow him a larger ability to customize his vote. We also want to give representatives weights, to take account of the fact that some of them will receive more votes than their counterparts and therefore should have a larger mandate. Lastly the system should discourage tactical voting, as we want people to submit their real votes.
For the negotiation and distribution of the budget, the beneficial characteristics are thus. Representatives can submit sources of expenditures and what their preferred budget is. During this pre negotiation phase all messages sent to each other is recorded and non repudiable, this should increase accountability. Then the system will run a automated negotiation algorithm, which solves the problem of representatives personality affecting the outcome. The negotiation algorithm should be deterministic and therefore others users can check the integrity of the budget. Again the algorithm should be difficult to manipulate by submitting a fake preferred budget. Lastly for the distribution of the budget, after the negotiation the money should be sent automatically to the managers of the expenditures.

Chapter 2

Background Literature

My backgroud reading could be broadly separated into two fields; multi agent systems and smart contract design.

2.1 Multi Agent Systems

An Introduction to MultiAgent Systems Wooldridge (2009)

From this book, there was as description of various voting systems and the desirable properties. First off there was a distinction between social welfare and social choice functions. They both took in a list of preferences from agents, however in social welfare it outputs a list of preferences, whereas social choice outputted a single option. In my system, I will require a social welfare function, as I need to select a number of representatives. For my program, voting preferences are important so that the representatives are representative by of all the voters wishes. Second order Copeland rating satisfies this. This is where each of the options has a pairwise election with all other options. Its then adds the sum of all defeated opponents to see what option is the best. This algorithm satisfies the Condorcet winner condition where w should be ranked first if it would defeat each other option in a pairwise election. This algorithm is Pareto optimal which means that if every one votes $w1 > w2$ then the output should be $w1 > w2$. Lastly, it also mentions the Gibbard-Satterwaite theorem which states that all voting systems are vulnerable to strategic manipulation. For my system, this is made worse by that fact that all transactions are transparent on a blockchain, making the manipulation even easier. However, second order Copeland rating is NP-complete to

manipulate which make it difficult to manipulate.
Also from the book, there was a description on how to run a negotiation algorithm. The negotiation set will be an array of sinks to values, where the values adds up to the collected money. The negotiation algorithm is where Agent 1 makes a proposal. If it is rejected then Agent 2 makes a proposal, then one etc. However if no deal is reached after a certain number of rounds, then the players receive a conflict deal. In the book, two assumptions are made, agents seek to maximise utility and disagreement is the worst outcome. However in my system disagreement is not always the worst outcome. Another important point is that time is valuable and that agents are impatient. This means a deal w at round t1 is better than deal w at round t2 if $w1 < w2$. The algorithm should create a Nash equilibrium, this means that no agent can increase his utility by changing actions.
Lastly, reading continued with the coalition section of the book. My system would be a weighted voting game. This is were each agent has a weight, we have a quota and a coalition C is winning if the sum of the weights exceeds the quota. A coalition is said to be stable if every agent can not get a higher utility by defecting.

Computational Aspects of Cooperative Game Theory Chalkiadakis (2012)

From this book, there was a description of coalition formation by self interested, rational agents. In particular, dynamic coalition forming where coalitions can form and fall apart. One approach is a Markov process, where agents explore suboptimal coalition formation actions and at each stage there is a small probability that they try to move to a new coalition. A player switches coalitions if her expected utility in the best available coalition exceeds her current utility. The player also demands as much as they can from the coalition.

2.2 Smart Contract Design

Pet-shop tutorial Truffle (2017)

I also did research on writing smart contracts, and found the Truffle development environment. I then read through the Pet Shop tutorial . It showed how to set up the environment, create new contracts with functions and variables, and how to migrate and deploy the contracts. Once the contracts are deployed on Truffle's

test network, it gives a number of addresses with Ethereum on them in order to test your contracts. These addresses can then interact with the functions on those contracts. Once you create the contract and deploy them, an interface can be made for the contract with web3. On my system, you will be able to register and vote on the interface, so that users do not have to use the command line. In order to use your Ethereum on a website, downloading the extension Metamask allows you to set your default account, and when you click register on the interface, it will send funds from that account.

Stateless Smart Contracts Childs-Maidment (2016)

In this article, Childs-Maidment describes how he was able to reduce the gas cost of his smart contract. The majority of the gas consumption on smart contracts come from storing data and to reduce the amount of data stored he used IPFS and Stateless Smart Contracts. IPFS take your key-value pair and store it as a hash, and then when access is needed your key-value pair you send the hash to IPFS, and it will return the key-value pair. IPFS is also decentralised so it does not compromise the trust less nature of smart contracts. Stateless Smart contracts have two parts, Dumb functions and filters. A dumb function will take the parameters required but will do nothing. Then when you want to access the data on your interface, since all transactions are public, using InputDataDecoder, find what parameters where passed to the contracts functions.
However, there are trade-offs with this approach. The first being tampering; the person who controls the interface could lie about what was sent. Although users could validate the transactions themselves which reduces the problem. It could also be a positive as anyone can build a filter which could add extra data to improve user experience e.g media, metadata. Another problem is that smart contracts do not have access to all the transactions and will not be able to see your data. Lastly if your contract has no events, it must process all transactions to find your data, but this can be mitigated by firing off random empty events. In my system I will use a mix of standard and stateless functions. The voting and registering will be standard, but for functions such as messaging, gas can be saved by making it stateless.

Ethereum Alarm Clock Merriam (2017)

This was a smart contract Merriam had put on the block chain that allowed you to pay someone to run functions after a certain number of blocks. This would be useful as I need time limits on voting, submitting a budget, term limits and registration and I need to run a function periodically to check that the time limits had not been reached yet.

The ultimate guide to audit a Smart Contract + Most dangerous attacks in Solidity Grincalaitis (2017)

This article described the best way to secure your smart contract from hackers. One common attack is Reentrancy. This is where users can interrupt the normal running of the function. Therefore in my system before any actions occur I will set the conditions to false. Another attack is under and overflows, for uint256 any number above 2**256 will be set to 0 and 0 -1 will be 2**256. Another attack is replay attack. This is where Ethereruem has a hard fork and you use the hard forked Ethereum to send the original Ethereum. Lastly, any function which is not changing the state of the application should be labelled as constant, to save gas.

Chapter 3

Final Design

Chapter 4

Implementation

Chapter 5

Testing

Chapter 6

Conclusions

It works.

Chapter 7

Future Work

Bibliography

Georgios Chalkiadakis. Computational aspects of cooperative game theory, 2012.

James Childs-Maidment. Stateless smart contracts, 2016.

Merunas Grincalaitis. The ultimate guide to audit a smart contract + most dangerous attacks in solidity, 2017.

Piper Merriam. Etherum alarm clock, 2017.

Truffle. Pet shop tutorial, 2017.

Michael Wooldridge. An introduction to multiagent systems, 2009.