Building a decentralized stock exchange using blockchain

Luiz Pizano Fonseca
New Technologies Department
Votorantim S.A.
Email: luizpizano.lpf@gmail.com

Paulo Andre Lima de Castro
Autonomous Computation Systems Lab
Aeronautics Institute of Technology (ITA)
São José dos Campos, SP, Brazil
Email: pauloac@ita.br

***Abstract*—This paper shows the implementation of the stock exchange on the Ethereum platform, taking advantage, this way, of the platform capacity of executing an application in a decentralized but secure way (secure either against single point of failure, either against a possible fraud). The assets and the currency used to buy those assets are represented at the Ethereum platform as specific Ethereum tokens; and a code implements the application capable of receiving buy/sell orders and capable of detecting automatically appropriate matches of buy/sell orders and perform the transaction with the transference of the currency and the assets.**

***Index Terms*—blockchain, stock exchange, decentralized**

1. Introduction

Cryptocurrencies have dragged attention of many investors and researchers. These coins are completely digital and its system doesn't rely on a central database registering all the transactions and the balance of all accoutns, but promises to have a safe copy of that database at any computer which offers itself to process the application that runs the system. That is the specialty of the technology behind Bitcoin: a decentralized and safe database. Without the need of a middle trustful organization, like a bank, that registers those transactions and demands money for its service.

The name of the technology briefly described above is blockchain. In a few words, it's a block chain where each one of them has a list of transactions (A tranfers 0.003 bitcon to B) already performed. In order to use the techology not just to decentralize the database, but whole applications, Vitalik Buterin proposed a platform which would allow its users to develop codes specifying functions which could for example transfer a digital currency, or specifying the functions to interact with a crowdfunding and under which conditions the collected money would be released, in a way that those "contracts" (smart-contracts as it is written in the official documentation) remains available at the platform for anyone to use it (for example to participate the crowdfunding). Once deployed on the platform, the code specifying the features of the contract wouldn't be corrupted, and in accordance with the interaction of the users, for example donating to the crowdfunding, the variable registering the total collected money (called state variable of the contract) would have its value updated at all the nodes of the network. There wouldn't be interest anymore on thinking of a central server for a crowdfunding application, such as Catarse [4]. Copies of the application and its data would be in multiple machines around the world, offered by any machine owner, and those machines would consitute the processing and data store of the platform. The name of that platform, already operating, and with a conference sponsored by giants like Microsoft and Santander [9], is Ethereum.

Another player of the system described on this paper is the capital market, BM&Bovespa for example, who, in order to negotiate the assets, receives into its system buy and sell orders and automatically recognizes when a buy order proposes a price bigger than the price proposed by a sell order and executes the transaction.

1.1. Motivation

Blockchain technology has been used to enable the creation of several different digital currencies. In this paper, we use it in a very different scenario: virtual stock exchange. Stock exchange are institutions that facilitate the buying and selling of financial assets by many economic agents. These exchanges rely on centralized systems. The exchange and its systems must have the trust of all the market participants in order to be effective. These require significant amount of investment in building trading systems and contracts among the exchange and the participants (brokers, buyers, sellers and so on). Our motivation is investigate the use of the blockchain technology to create a completely decentralized virtual stock exchange.

We may consider the stocks of a company as a specific coin. The Ethereum platform, in turn, allows the creation of a custom currency and of all the instructions to control the authorization of a transaction (for example receive buy and sell orders and automatically perform transactions when there is a proper match), all with safe transactions and without the need of a trustful organization that operates the transaction, like BM&Bovespa.

Being aware mutually of the capital market need and of the Ethereum platform potential, we notice, as it was mentioned, that the assets and the money can be represented on Ethereum platform as custom currencies, and a code (or smart-contract according to Ethereum terminology) can specify an application capable of detecting automatically proper matches of buy orders and sell orders and then performing the transaction with the transference of the assets and the money. That way the capital market can be implemented on the Ethereum platform.

1.2. Objective

This paper aims to represent the assets and the money as custom currencies on the Ethereum platform, and deploy a smart-contract capable of receiving buy and sell orders and automatically detect proper matches of buy and sell orders and execute the correspondent transaction with the transference of asset and money. That way the asset exchange market will be implemented on the Ethereum platform.

2. Ethereum platform

2.1. Ethereum

Figure 1. Last release of the platform.

A blockchain-based platform which allows not only the decentralization of a database, but of a whole application, which, besides variables, contains program instructions, as presented in section Introduction:Motivation 1.1. “With Ethereum, a piece of code could automatically transfer the home ownership to the buyer and the funds to the seller after a deal is agreed upon without needing a third party to execute on their behalf.” [3]. The applications for this platforms, called “dapps” (decentralized applications), are developed in Solidity, a language created by the Ethereum project.

As presented in Ethereum project homepage: “Ethereum is a decentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference.

These apps run on a custom built blockchain, an enormously powerful shared global infrastructure that can move value around and represent the ownership of property.

This enables developers to create markets, store registries of debts or promises, move funds in accordance with instructions given long in the past (like a will or a futures contract) and many other things that have not been invented yet, all without a middle man or counterparty risk.

The project was bootstrapped via an ether presale in August 2014 by fans all around the world. It is developed by the Ethereum Foundation, a Swiss nonprofit, with contributions from great minds across the globe.”

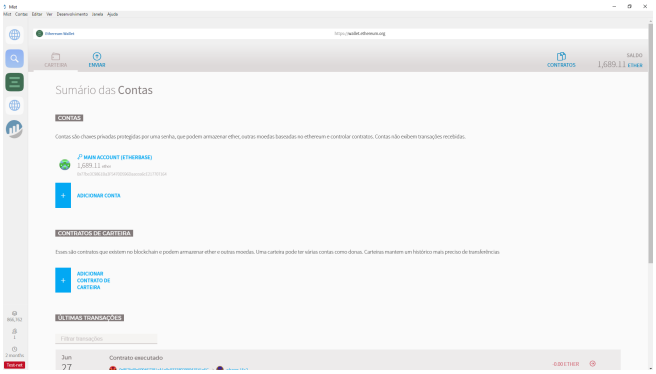

Figure 2. Graphical interface to client users, the Mist.

In order to just create accounts or deploy smart-contracts, capable of, for example, manage a crowdfunding, there is a friendly graphical interface, the Mist. In that case, the platform user will play the role of client (not of server, or, better saying, miner).

A contract is considered an account, though it is controlled by its contract code, differently from the externally owned accounts, the accounts owned by someone with a password and that does not have a contract code associated. More detailed explanation can be found at Ethereum documentation [1].

2.2. Solidity

Solidity is the language created by the Ethereum project to write smart-contracts. As it is written in Solidity documentation [19]: “Solidity is a contract-oriented, high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. As you will see, it is possible to create contracts for voting, crowdfunding, blind auctions, multi-signature wallets and more.”

Tutorials to learn Solidity, for example to create a cryptocurrency [7], can be found at Ethereum website [11].

Since the development required a debugger, the Remix IDE was used [18]. It’s “a browser-based IDE with integrated compiler and Solidity runtime environment without server-side components” [19].

Figure 3. Remix editor and debugger tab.

3. Virtual asset exchange

3.1. Contracts structure

Two contracts implement the system: *Currency*, managing the currency balance of the accounts, and *Assets*, managing the asset balances of the accounts, buy/sell orders of assets, and automatically performing transactions when proper conditions are satisfied.

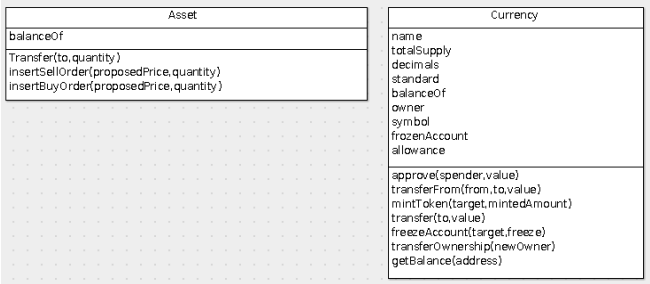

Figure 4. Three sections describe each contract: name, public attributes and public functions.

A complete use of the system will be described in section "Virtual Asset Exchange:Usage" 3.2, but it can be visually summarized with the diagram of figure 5.

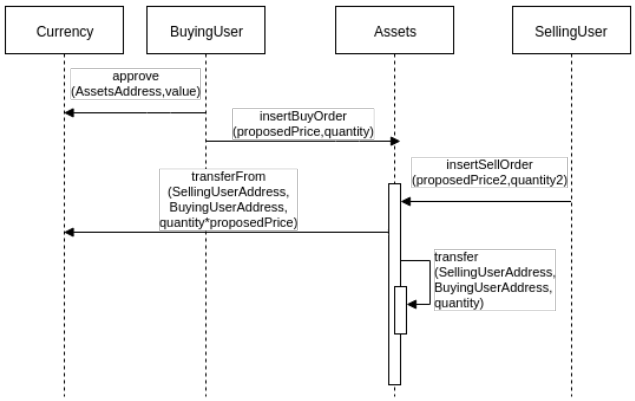

Figure 5. The interaction between the contracts is: after the approval by the buying user for the *Asset* contract to manipulate its currency, when the *Asset* contract performs a transaction, it calls a public function of the *Currency* contract to transfer currency from one account to another according to the price and quantity of assets involved in the transaction.[2]

Asset public attribute:

balanceOf maps account addresses to its balances.

Asset public functions:

transfer(to,quantity) transfers a certain "quantity" of asset unities from the calling account to the account whose address is the "to" argument address.

insertSellOrder(proposedPrice,quantity) inserts a sell order with a certain "proposedPrice" and a certain ordered "quantity".

2. This sequence diagram was drawn with the online tool accessible at https://www.draw.io [23].

insertBuyOrder(proposedPrice,quantity) inserts a buy order with a certain "proposedPrice" and a certain ordered "quantity".

Currency public attributes:

name is arbitrarily chosen, like "Real".

totalSupply is the total existent quantity of the currency.

decimals is the number of decimal algarisms of the currency, so a quantity of 1000 of the currency could actually be represented as 10,00 if decimals=2.

standard is just a way versioning the currency contract, although it's not needed. It's actually an inheritance from the Ethereum tutorial to create a cryptocurrency [7].

balanceOf maps account addresses to its balances.

owner is the address of the owner of the *Currency* contract, who can for example mint currency.

symbol is the symbol of the currency, "R$" for example.

frozenAccount maps account addresses to a boolean determining if the account is frozen or not.

allowance maps account addresses to another map with all the addresses allowed to manipulate the currency of the account address and the quantity each address can manipulate.

Currency contract public functions.

approve(spender,value) allows "spender" address to manipulate the "value" quantity of the caller account.

transferFrom(from,to,value) transfers the quantity "value" if the caller's account address is allowed to do so.

mintToken(target,mintAmount) creates the "mintAmount" quantity of currency and allocates it at the "target" balance.

transfer(to,value) transfers the quantity "value" to the "to" account.

freezeAccount(target,freeze) freezes/unfreezes the "target" account depending on the value of the boolean "freeze".

transferOwnership(newOwner) transfers the ownership of the *Currency* contract to the "target" account. Only the owner of the contract can do that.

getBalance(address) returns the balance of the provided account.

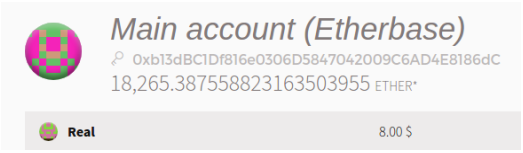

Figure 6. Currency is exhibited at the client graphical taking into account its symbol and the number of decimal algarisms.

The attributes of the *Currency* contract were not randomly chosen, they were inherited from the cryptocurrency creation tutorial [7]. The proposed currency is recognized as a currency by the client application, so it's exhibited by the graphical interface at an exclusive currency frame, taking into account its symbol and the number of decimal algarisms, as show in figure 6. Probably the set of attributes and functions needed for the client application

to recognize the currency contract are the ones defined at the standard token interface ERC20 [8].

3.2. Usage

A complete use of the system, as it is represented at figure 5, happens when:

1) An user inserts a sell order just like the figure 7.

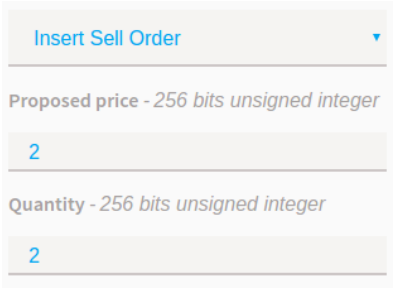

Figure 7. 2 assets are offered at a price of 2 each.

2) Another one inserts a buy order such that the proposed price in the sell order is equal or smaller than the proposed price of the buy order just like figure 8.

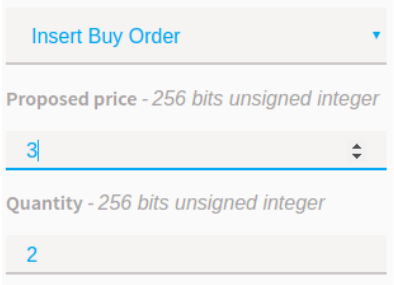

Figure 8. The user intends to buy 2 assets at the price of 3 each.

But before that the user needs to approve the *Currency* contract to manipulate its own money, just like figure 9. The function *Approve* is available at the *Currency* contract; *Currency* contract address must be provided and the quantity approved must be at least 6 in this case since the buy order intends to buy 2 assets at the price of 3 each.

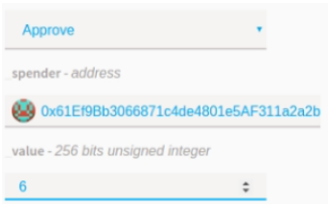

Figure 9. Account owner allows the *Asset* contract to manipulate six units of currency from its own account.

3) Under such conditions, a transaction automatically happens and so assets and currency are transferred.

Figure 10. The transaction happened and the solicitant of the buy order received 2 asset units.

Figure 11. If we check the currency balance of the buy order solicitant, we notice that it was decreased by 6 currency units, since he proposed to buy 2 assets by 3 each. As in this case he owned 50 currency units originally, now he owns 50-6=44.

3.3. Contract deployments

The contract *Currency* was deployed using Mist, figure 12.

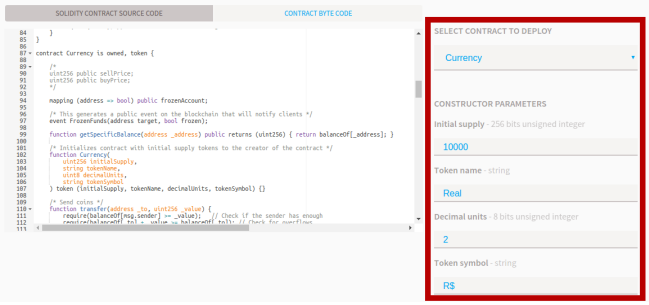

Figure 12. Highlight of the parameters needed to deploy the *Currency* contract.

Next, the contract *Asset* was deployed, figure 13.

Figure 13. Highlight of the parameters needed to deploy *Asset* contract.

Figure 14. Obtaining, in Mist, *Currency* contract address. The provided token name was *Real* in this case, as in figure 12.

Notice in figure 13 it was needed to provide the *Currency* contract address to deploy the *Asset* contract. Figure 14 shows it could be obtained in Mist.

3.4. Solidity implementation

First of all, a contract to manage the currency used to buy the stocks was implemented. The contract is called *Currency*, but the name could be replaced conveniently, by *Real* for instance.

The contract *token*, is the basic parent contract to implement a customized cryptocurrency. It is basically the code available at Ethereum website [7]. *Currency* inherits *token*.

The contract *owned*, is used to assure that the only account that can use the function *mintToken* is the owner of the *token* contract account. *Currency* inherits *owned*.

Next contract *Asset* was created to manage sell/buy orders.

The buyList and the sellList arrays work as a max-heap and a min-heap respectively, since to evaluate if there is a match of buy order and sell order, only the maximum/minimum buy/sell order price is needed.

The source code can be accessed at my Github [20].

4. Experiment

4.1. Environment setup

In order to perform a transaction at the Ethereum network, deploying a contract for example, it's necessary to pay a tax with ether, the platform currency, which is received as a reward by the miner who builds a valid block. Since mining at the official, public network would require a GPU and would take too much time, a private network was created with a low difficulty math challenge so that mining with the CPU would be fast, a valid block building every two seconds approximately.

genesis.json

```
{
    "config": {
        "chainId": 15,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty": "20000",
    "gasLimit": "2100000",
    "alloc": {
        "0x8f2b49ef99467381aa1e9c9333802990a7641eec": { "balance": "300000" }
    }
}
```

Figure 15. Genesis file specifying a new private network.

In order to create a new private network, it's needed to provide configuration file, named *genesis.json*. One of its configuration parameters is "difficulty", the parameter setting the aforementioned math challenge difficulty.

Once the private network is created, the mining can start, so that the required transactions may enter the blockchain.

Figure 16. Mining running.

4.2. Experiment

The experiment conducted next was precisely the one described at section 3.2. All the steps are described there with details, but just as a refresh they are summarized here.

1) An user inserts a sell order.
2) Another user (who will be the buy user) needs to approve the *Currency* contract to manipulate its own money. The function *Approve* is available at the *Currency* contract; *Currency* contract address must be provided and the quantity approved must be at least all the buy user is willing to spend since an automatic transaction will need approval to manipulate the money.
3) The buy user inserts a buy order such that the proposed price in the sell order is equal or smaller than the proposed price of the buy order.
4) Under such conditions, a transaction automatically happens and so assets and currency are transferred.

A video record of the experiment can be watched at https://youtu.be/TMg4yJEnnK4 [6].

5. Conclusion and future work

Our implementation of a decentralized stock exchange on the Ethereum platform is based on the creation of two contracts: *Asset*, which manages the balances and the transferences of a particular kind of asset, and the contract *Currency*, which represents a currency, like the brazilian Real and manages the balances and the transferences of the currency. Any user who wants to buy assets needs to approve the *Asset* contract to manipulate its currency. That can be made using the public function *Approve* of the *Currency* contract. If the Ethereum network proves to be secure along time, the implementation wont't be just possible, but acceptable.

The source is available in Github [20] and a video of the complete usage of the system can be watched at my channel on Youtube [6].

All was done at local computer, with just one node. A next step could be creating a node at another computer with the same *genesis.json* file and add the first node as a peer so that both nodes starts to synchronize. The Geth Github Wiki provides instructions on how to synchronize two private nodes [21]. I antecipate I did it as a trial and it has not worked with what they call *boot node*, but with normal complete nodes. So when I used "admin.addPeer(enodeUrlOfFirstInstance)" command, I used a normal complete node address.

References

[1] "Account types, gas, and transactions - ethereum homestead 0.1 documentation," ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html.

[2] "Account types, gas, and transactions; ethereum homestead 0.1 documentation," http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html?highlight=gas#what-is-gas, [accessed 2017-05-04].

[3] "A begginer's guide to ethereum - the coinbase blog," https://blog.coinbase.com/a-beginners-guide-to-ethereum-46dd486ceecf.

[4] "Catarse crowdfunding platform," https://www.catarse.me/, [accessed 2017-11-13].

[5] "Choosing a client - ethereum homestead 0.1 documentation," http://www.ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html, [accessed 2017-10-26].

[6] "Complete use of the capital market system on ethereum," https://youtu.be/TMg4yJEnnK4, [accessed 2017-11-16].

[7] "Create a cryptocurrency contract in ethereum," www.ethereum.org/token.

[8] "Erc20 token standard - the ethereum wiki," https://theethereum.wiki/w/index.php/ERC20_Token_Standard, [accessed 2017-11-11].

[9] "Ethereum developers conference devcon2," https://www.youtube.com/watch?v=66SaEDzlmP4&t=234s, [accessed 2017-11-13].

[10] "Ethereum documentation," http://www.ethdocs.org/en/latest/.

[11] "Ethereum project website," https://www.ethereum.org/.

[12] "Ethereum stack exchange, an ethereum questions and answers platform," https://ethereum.stackexchange.com/.

[13] "ethereum/go-ethereum: Official go implementation of the ethereum protocol," https://github.com/ethereum/go-ethereum, [accessed 2017-10-26].

[14] "ethereum/yellowpaper: The yellow paper: Ethereum's formal specification," https://github.com/ethereum/yellowpaper, [accessed 2017-10-26].

[15] "Geth's github wiki," https://github.com/ethereum/go-ethereum/wiki.

[16] "How bitcoin works under the hood," www.youtube.com/watch?v=Lx9zgZCMqXE&t=57s . Acessado em: 2017/06/20.

[17] "Installation instructions for ubuntu · ethereum/go-ethereum wiki," https://github.com/ethereum/go-ethereum/wiki/Installation-Instructions-for-Ubuntu, [accessed 2017-10-26].

[18] "Remix, the solidity compiler," https://remix.ethereum.org/.

[19] "Solidity documentation," https://solidity.readthedocs.io/en/develop/.

[20] "Source code." https://github.com/luizpizano/Ethereum-capitalMarket, [accessed 2017-11-17].

[21] "Synchronization of private nodes." https://github.com/ethereum/go-ethereum/wiki/Setting-up-private-network-or-local-cluster, [accessed 2017-11-16].

[22] "Synchronization of private nodes." https://en.bitcoin.it/wiki/Confirmation, [accessed 2017-11-17].

[23] "Tool to draw diagrams," https://www.draw.io.