Decentralized Marketplace Protocol

Br. Sebastian Alejandro Tuyu Piñeiro
alextpineiro@gmail.com

April 2021

Abstract

A decentralized marketplace is where there is no third-party needed for making transactions of physical assets. Due to its nature, there are several problems related to the structure and trustability between users. Following traditional implementations, a full decentralized marketplace is really hard to implement due delivery is usually relayed in centralized services. In this paper, we explore a potential solution for making a decentralized marketplace, its limitations, and further implementations.

Introduction

The definition of a marketplace is referred to as a group of buyers and sellers for a particular good or service[1], from this definition it is developed a decentralized marketplace where there is no third-party involved or a centralized authority that can regulate transactions and avoid bad behavior between users.

A general solution requires an integration of multiple actors across the marketplace ecosystem. Inside this system, regulations are made with economic incentives to users, *deals* are introduced for showing how bad behavior should be avoided on users.

If we talk in terms of *buyer-seller* on centralized organizations many times users falls on schemes where the *trust* given to a seller is not because is expected the users good behavior is just because the third-party involved has created an artificial trust created for regulating the transactions. This is why monopolistic marketplaces can charge high fees on services used by the users when they sell a product. A complete decentralized market place should be deployed agnostically and with a prior focus on the user past history to reinforce the expected behavior from every user also should be used by users based on the complete transparency of the system.

Another main problem of the solution could be that users are used to *trusting* in central authorities that can regulate sellers or buyers behavior and even penalize fraudulent users, implementing a decentralized that can operate without this can be quite difficult even considering the costs of implementation. However, deal pools are introduced fixing this problem and another implementations complementing and building a nicer user experience from a simple buy and sell to *package delivery.*

The full implementation of this solution is built with the main goal of not-delivering services on third-party entities, making a complete decentralized and open-source solution.

Minimum data with maximum trust.

General operations

Users

Each user only needs a public key to participate in buying nor selling on the market. *(talking in terms of data).*

Sellers

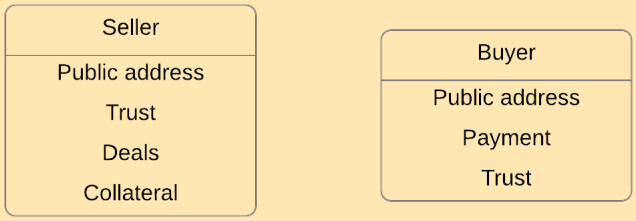
In a seller situation, for obvious reasons is needed to complement with characteristics of the articles. Besides, to participate in the network and generate *deals* is necessary that each seller has a collateralized amount of tokens or coins that backed his total volume of the current articles sold.

If the collateralization value goes below the actual volume the users could only sell simultaneously products where the volume price is smaller or equal to the total current collateralization.

Trust

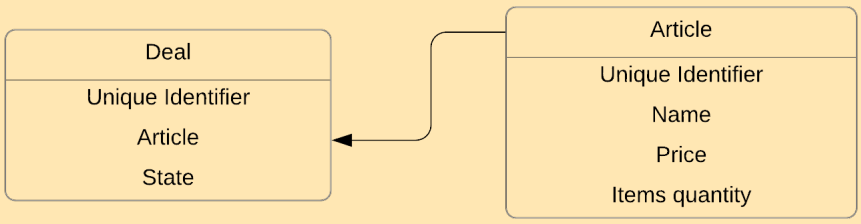
Trust is a *unsigned integer of 8-bit* matrix metric associated with each user as a complement with his collateralization(*in the case of sellers*). This metric is a reference of the past user behavior on a deal, after finalizing a deal, buyers and sellers have to give a number that is bigger than 0, but below 15.

This trust is intended as an auto regulation mechanism where the users can see if the users have incurred on bad behavior and avoid *a priori* making a deal with the bad-history-user.



Brief introduction to Deals

A Deal is described as a set of rules where the sale of a physical product and the characteristics of it are determined. One single seller can have multiple and unlimited quantity of deals, each one of the deals is designed in a way that permits an easy maintain of the price or the number of items sold and even the current state of the deal. A unique ID is generated depending on the pool that depends of the order of creation and is this ID that prioritize queries and shows products above others, this position can be boosted by paying *Deal Tokens* to the pool where the deal was created.



It's through the deals that users can buy or sell any article. These deals accepts initially any kind of token which follows the ERC-20 standard, however for avoiding issues with the implementation deals only would accept fixed-value coins or stablecoins and the native cryptocurrency where the application is deployed. On further implementations, the payments with other ERC-20 tokens on the deals are considered.

It's important to remark that any data refering to the physical location of the assets until now, this characteristic would be introduced in the *deal pool* section.

Due to its nature, the *deals* work as the product guardians where they were created, that is why on this section the business rules are defined and also penalization-patterns are defined too.

Deal states

Each deal has 4 possible states for working:

- OPEN: In this state, any user can buy the deal offered by the owner. On its creation, every deal is set by default as *OPEN*.
- PENDING: In this state, the deal has been paid by a buyer (but the payment has not been authorized to the seller), and its delivery is pending between the users.
- CLOSE: The deal has been paid to the correspondent parts (buyer-seller) and is also has been delivered.
- CANCELED: The deal has been canceled for any of the parts, or the deal is not currently available.

Exceptions

- A deal has to be confirmed manually by the current buyer. However, if the confirmation is not made after the **20 days** of the delivery day, the payment would be authorized to the seller its withdraw of the payment.
- In case that a seller does not make the delivery of the next **20 days** since the payment has been made, the seller would be penalized and the buyer could withdraw his funds.

Deal Pools

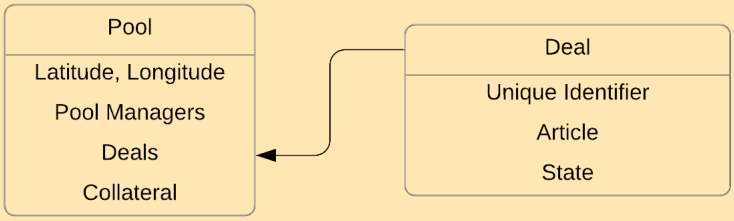
For avoiding the creation of more complex infrastructure, the *deal pools* would be introduce introduced as a solution for a medium of object localization between users in a more efficient way for matching users with sellers on closest locations as is explained shortly.

A Deal Pool is *essentially* a group of contracts that has the capacity of hosting multiples deals depending on the location where the deals are physically located, this location is picked by the user when the deal is originally created, in this way we can find multiple articles which belongs to the same region without reveling the actual location of the package from the deal before the deal is even bought, this can be translated into user experience improvements due there is no need of revealing confidential information on the early stages of the incorporation to the ecosystem.

The creation of the pools is open, any *set* of users which holds *DealTokens* [1] has the capacity of making a pool deploy. When a pool is created, the location reference uses coordinates of latitude and longitude for its localization, these pair of coordinates should be on a neutral position[2] inside a the location where the pool wants to be incorporated since **the location does not represent a meeting point, is an abstract representation of the *deals*.**

[1] DealTokens: is a reference name of the token used by the protocol, and mentioned as DealToken to avoid confusion with other concepts.

[2] *Neutral* position means that the coordinates do not reveal location of any user inside the network



Impartial mechanism of creation

The minimum amount of tokens that a pool can accept(β_{min}) has a non-fixed nature, this parameter can be changed by a voting by all the *Deal Tokens* holders. However, is required to set an initial amount by default for making the pool work is described as:

$$\beta_{min} = 1000 \tag{1}$$

Each user can collateralize a maximum amount(*represented by* ϕ_{max} *and expressed as a percentage*) of tokens for a pool creation, this is intended as a guarantee to avoiding that a single user has too much percentage inside a single pool decision.

$$participants_{minimum} = \frac{100\%}{\phi_{max}} \tag{2}$$

Initially, the maximum percentage accepted that a user can collateralize in a pool is 2.5%. Given this amount, we solve for the minimum participants:

$$participants_{minimum} = \frac{100\%}{2.5\%} \rightarrow 40 \tag{3}$$

A pool can be overcollateralized, which means that can has more than 100% of the tokens required for making it work. With more users, the deals are more safely secured and more users can participate for earning rewards solving issues through solvers.

Pool managers

When a user gives his token for a pool creation, the user has the possibility of being a pool manager. There is a minimum quantity of pool managers that should exist in a single pool (*explained on eq.2*), due to is required the active participation of the pool managers.

When there are not enough users to maintain a pool, new deals can't be generated until the minimum threshold is accomplished, at this point, only deals that are on *PENDING* state can be ended even if there are not enough pool managers.

Each time a deal is sold through a pool it gets compensation fees on a percentage over the selling price. This percentage fee is given by:

$$reward = \sum_{0}^{j} 0.35\% \cdot deal_{price_j} \tag{4}$$

The fee is split between the pool maintain and the pool managers who await to resolve a solver. A 0.20% is for the pool maintain and 0.15% for the pool managers.

When a *solver* is listed, is not emitted to all managers but managers are picked randomly across all users who have collateralized founds into the pool, picking a minimum 60% of all users that are currently participating in the pool.

Deal Pools as guardians

As is mention before, any seller who decides to generates deals in a particular pool needs to have a collateral amount equivalent to the total volume of the price of the products and the total items of the product sell as is described on eq.5.

$$volume = \sum_{0}^{n} item_{quatinty} * price_{actual} \tag{5}$$

To withdraw the collateralized amount for backing deals, is needed that the user has no current contracts on *PENDING* state for generating no-conflicts on current deals due to suddenly no-backed founds of sellers.

Solvers

On each pool, users can send solvers that are issues between users that need to be verified by multiple pool managers (*as is mentioned previously*). Is very important to say that there is an economic incentive for the pool managers to resolve any solver that is designed for them since on each deal solver there is a fee that is paid (*and split*) directly through every pool manager who was designed and voted for the solver resolution.

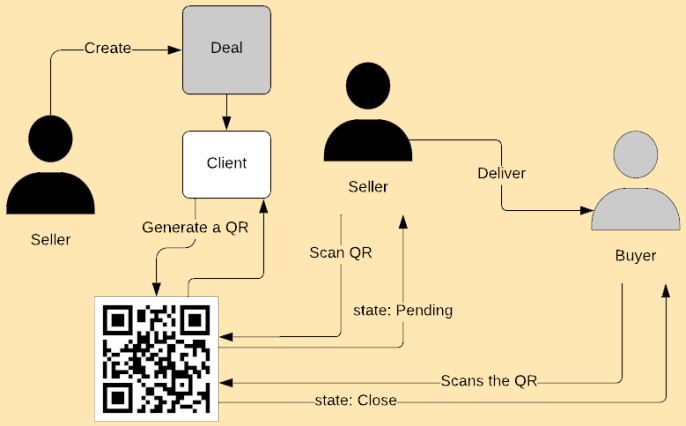
A solver response is made with a fixed format, in this way is ensured that there is congruence between votes on users. Each number represents a percentual part of the current amount that a user has deposited into the deal. So, the parameters are numbers bigger than 0 but smaller or equal to 100.

```
function sendResponseToSolver(uint solverID, uint perBuyer,uint perSeller);
```

Once that a certain amount of pool managers has emitted a vote, the action is executed and the pool managers can get their rewards proportional to the contracts. Initially, the fee is set to 5% however this value also can be modified through voting on pool managers if at a certain point of time users are not attracted to the current fee charged.

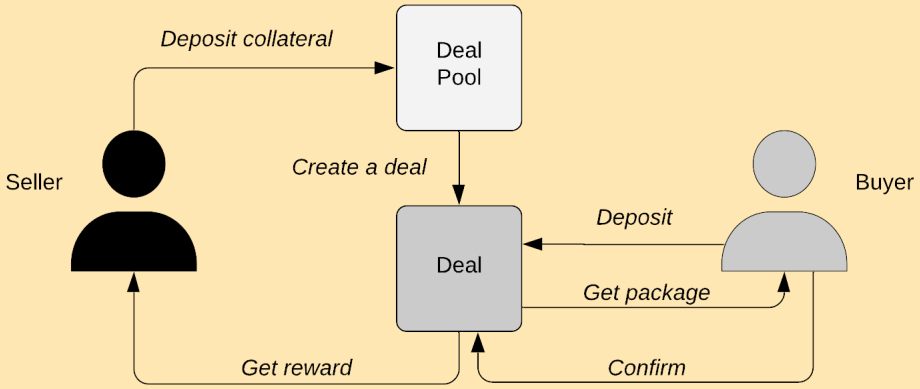
Tracking packages

For making a better user experience through the whole process of changing the state of the deal depending on the step of the process is suggested that a client runs on top of the described contract on this paper should generate a QR code based on the step of the process and the unique ID of the deal. This implementation can replace even interfaces for updating if a package was sent and received.

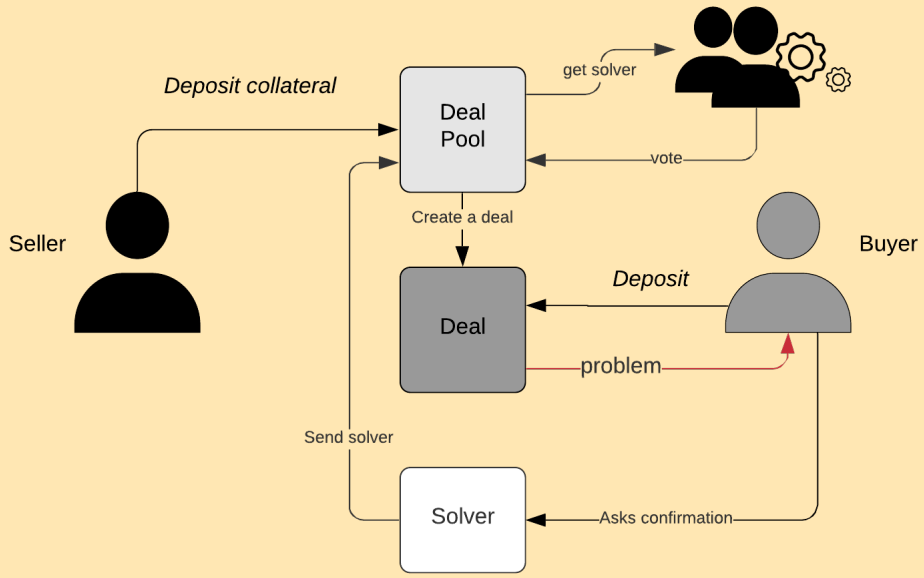


Deal Pools on action

The following diagram is illustrates how a transaction between a user and seller uses the *deals* and the *deal pools*.



In case of exists a problem during the transaction, both buyers and sellers can send a *solver* to the correspondent *deal pool* so the transaction can be reviewed by the pool managers and can be decided how would be the transaction split between the users involved as is illustrated on the following diagram.



Details on deliveries

On the previous sections it barely explained how the logistics across buyer-seller should be do it on detail, however, this characteristic is implemented as a complement to the protocol. This component can be flexible on how is implemented and can be done using a service that provides a **socket** for communication using an only public address for each deal that is made, deleting the chat (*the resources associated with it*) every time that a deal is closed and no solvers have been listed.

This can be done using different technologies and even use a *pseudo-centralized* chat provided by a client, made open-source and reinforcing the privacy for user communication as long of the deal remains pending. This implementation is given to the lector as a further complement.

Tokenomics

Due to the way that the protocol operates is necessary to implement a token that allows to compensate users and allow a governance protocol where users can participate in decisions related to the application behavior and parameters. The token has a non-fixed supply, where the initial amount of tokens is 100,000,000 units and can be also burnable. The token also must have the following characteristics:

- Governance Token. The token can be used for voting on governance, it can modify general parameters which the full protocol works it can be used also for participation in solvers. The governance mechanism is based on the uniswap governance protocol [3] where there is a 3-day voting period for each proposal based on timelocks.
- ERC-20 Compatible. It's needed that the token fulfill the ERC-20 standard due this allow its distribution and would make it easier listed on decentralized exchanges.
- *Utility token.* The token can be used principally for the creation of deal pools and for boosting the appearing position of deals to deals created previously.

Further implementations

Deliveries across Deal Pools

The protocol described in this paper only can be used for users that are currently in the same physical place (*or near enough*) and it can not be possible to send packages across significant distant locations. That is why the deliveries from poo-to-pool are considered for implementation on further versions of the protocol.

Size-fit blockchain

As long as the protocol still getting upgrades, would result in more efficient to adopt a blockchain custom made for the full protocol, using technologies like Cosmos SDK[3] this blockchain should have the capacity for being a bridge between the original blockchain where the contract was deployed and the new blockchain for migrating the services. Making a custom-specific chain would permit expand the protocol by implementing more complex characteristics. Also with this implementation users could participate as verifiers of the network expanding the ecosystem across the whole marketplace.

[3] https://v1.cosmos.network/sdk

References

[1] Mankiw, N Gregory (2014). *Principles of economics.* Page 66.

[2] Vogelsteller F, Buterin V. (2015). *EIP-20: ERC-20 Token Standard. Ethereum Improvement Proposals.* https://eips.ethereum.org/EIPS/eip-20

[3] Uniswap (2020). *Uniswap V2 Governance. Governance Reference.* https://uniswap.org/docs/v2/governance/governance-reference/