Supply Chain on Blockchain

Miles. Frain
University of Colorado, Boulder

Nicholas. Zimmerer
University of Colorado, Boulder

Aakash. Kumar
University of Colorado, Boulder

Abstract

In this paper, we outline a prototype supply chain model built on the Ethereum blockchain. We describe the architecture in Section 2, bid execution and rewards in Section 3. Section 4 discusses trade-offs between multi-contract and monolithic contract designs, along with additional notes on implementation.

1 Introduction

User-facing products are often built from numerous components, each with several vendors and dynamic prices. With component prices dependent on quantity purchased and lead time required, the final product's price may change drastically depending on the number of products ordered, and shipping information such as lead time.

Current crowdfunding solutions are very limited with regards to the above, often requiring a designer statically set the quantity and/or price of their product before launch. Additionally, both ends of the transaction rely on trust: the customer allocates funds in good faith that the designer will follow through, and the designer must trust the number of backers will not be inflated by holders of invalid or expired payment methods.

To enhance crowdfunding capabilities, we built a system of smart contracts for suppliers to list their inventories, designers to list their products, and consumers to place bids. Purchase execution may be triggered whenever bid conditions are met, distributing the funds recursively through the contract chain. Validation guarantees that consumer funds exist before triggering, and consumers can verify their funds are being used to procure their desired product's components.

2 Architecture

For our initial design we used four contracts for implementing the supply chain on blockchain. Later iterations and refactoring consolidated this design to a single monolithic contract, which is discussed further in Section 4. The original contracts are Inventory, Product registry, Customer bids, Supply chain.

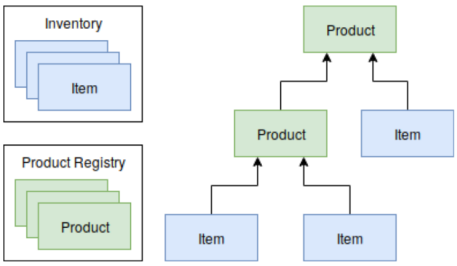

Figure 1: Example product hierarchy

2.1 Inventory

The Inventory contract is used by suppliers and vendors to list items from their inventories, along with pricing information. Each supplier's inventory is linked to a single address, allowing them to update currently available quantities and prices for each of their items. To reduce the complexity of the contract, the pricing scales with the quantity of items at discrete thresholds rather than a true curve dictated by a continuous pricing function.

2.2 Product Registry

The Product registry contract is used by product designers to create products consisting of a collection of other products and / or items listed in Inventory contracts by suppliers. Designers can list the quantity of each component required, creating their own items to include as needed (such as a "service" item for charging a designer fee). Figure 1 shows how a sample product could be produced. Note that products may consist of both items and sub-products.

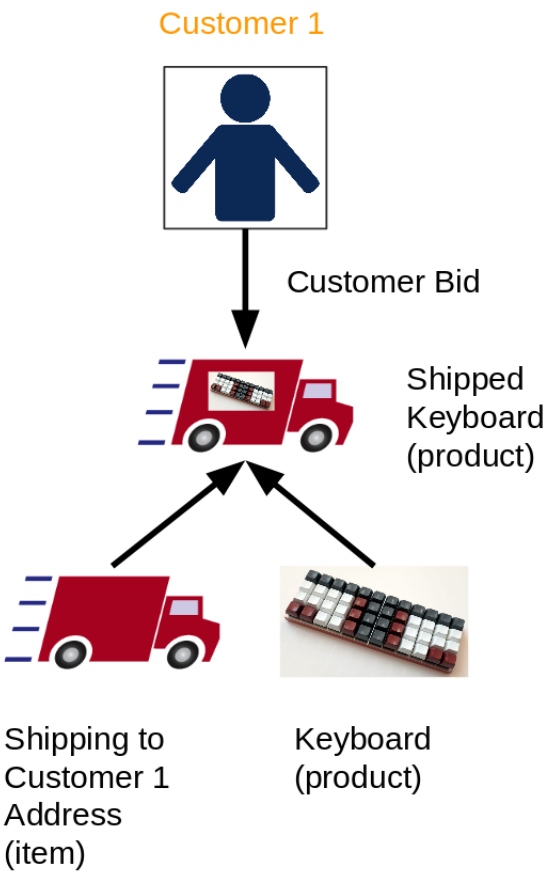

Figure 2: Shipping integration

2.3 Customer Bids

The customer bids contract is what customers interact with, allowing them to deposit or withdraw funds, and place bids on products listed by the product registry contract. In situations where the product needs to be shipped, the customers use an off-chain fulfilment service to generate an item representing their shipping cost, and a composite product containing their personal shipping item and the product they want to bid on. Figure 2 shows a potential composite shipping product for a customer bidding on a keyboard.

2.4 Supply Chain

The supply chain contract is the super contract in charge of managing the other three contracts. This is how the the external trigger interacts with the rest of our system. The trigger logic is fired by an executor when they can identify a valid collection of bids. This contract also collects payments from customers and pays the vendors and suppliers once the contract is executed.

3 Execution and Rewards

We provide the logic in the `execute()` function to verify that a provided list of bids is valid to invoke their respective orders and cover all expenses. Anyone is allowed to act as an executor, and it is up to them to perform the more computationally expensive work of scanning for and putting together a valid list of bids. Fortunately, this scanning effort can be performed off-chain, and there are many algorithms which can be used for detecting triggering conditions. We outline a few possible strategies later in the section.

In exchange for this executor effort and to offset gas expended during an `execute()` call, the executor may request a percent commission from all orders. This commission is applied uniformly to all items at their calculated price breaks, and all customer bids must have enough margin to cover this commission. The executor must be careful to select an appropriate commission that both covers their estimated gas expenditure and is within the bid margins. The executor must also not be too greedy in holding-out for an excessively high commission, because anyone else is free to jump in and execute at an earlier point for a more reasonable commission. We hope that this free-market system of leaving this execution detection logic up to enterprising developers will encourage the use of the most capable detection algorithms with the lowest allowable margins for customers.

Some possible detection algorithms are the following:

- Brute-force check of all possible sets of bids. This quickly becomes unmanageable as the possibilities to check are $2^{(numberOfBids)}$.
- Elimination of bids on products where the sum of the prices of all child items at the most generous price breaks still exceed the bid amount. This list-thinning strategy can be combined with other techniques.
- The above elimination strategy, but use the price breaks that correspond to item quantities equal to the sum of all the items in existing bids. This is a more aggressive thinning than only selecting the most generous price break at maximum quantity.
- Consider each product as a "pivot". For bids involving products that are parents of this pivot, deduct the most conservative cost of the pivot's siblings from the available bid amount. Then sort the remaining bid amounts in descending order, and check if any of these intersect the pivot product's price curve at a given quantity.

A potential exploit on the executors is for a lazy node to listen to `execute()` calls being broadcast, then steal the bid list and re-broadcast their own `execute()` call with this stolen list. We do not have a strategy to prevent this attack.

4 Implementation Notes

4.1 Truffle framework

We adhered to a test-driven-development workflow using the Truffle Framework and a local Ganache blockchain. This allowed for continuous testing as features were added and the code was refactored. We would not have been able to progress through development with as much confidence and speed if we had to manually test with Remix on the slower Ropsten Testnet. We also used Ganache to setup a small React app that we were able to successfully link to MetaMask and our local Ganache chain. We would like to eventually use our Web3.js testing code to quickly populate our web app with trees of products and items.

4.2 Multiple vs Monolithic Contract

Our original multiple-contract strategy treated contracts like classes. This modular approach seemed forward-thinking at the time, but we found interactions between contracts to be very restrictive. The biggest inconveniences involve sharing structure definitions between contracts and accessing the structure contents of other contracts. The workaround requires writing and maintaining all the getter and setter methods necessary for accessing these fields.

4.3 An honest "State of the Code"

The bookkeeping functionally involving tracking items, products, customers, and bids is complete and tested. There are some enhancements and additional testing for these bookkeeping features on the to-do list including:

- Adopt the storage pattern used for Bids to Products and Items. This improved storage pattern supports deletion.
- Move away from address-based (designer/manufacturer) tracking of products and items. This will streamline execution logic, which will then only need to track parts by ID.
- Add more tests for error cases.

Some of the order execution verification logic is written and compiles, but a vast majority of it only exists as pseudo code. There appear to be some Solidity language limitations with writing this function as originally intended, as well as concerns with gas prices and storage, but these challenges can likely be overcome with additional development effort.

4.4 Challenges and Issues

We cannot figure out how to update Web3.js independently of the Truffle framework. We are on the latest version of Truffle (v5.0.15) which includes Web3.js v1.0.0-beta.37, but we need to move to the latest Web3.js v1.0.0-beta.54 for fixes to many issue we are encountering. Here are some of those problems:

- Disappearing structure fields (we filed this ticket): https://github.com/ethereum/web3.js/issues/2736
- Problems with the Web3 BigNumber object https://github.com/ethereum/web3.js/issues/

4.5 Code

- Most recent monolithic contract (and some additional features):
  - Contract code: github
  - Test code: github
- Earlier multiple contracts:
  - Contract code: github
  - Test code: github

5 Conclusion

Our inexperience with the Solidity language and frameworks made this mostly an exercise in learning the quirks of the language. Focusing on a sufficiently complex and interesting endeavor to force ourselves to gain more competency with blockchain development was certainly a goal of this project. There is enough remaining on this project's roadmap to fill upcoming semesters.

6 Future Work

Future work includes a full fledged supply chain with an easy-to-use interface to the external world. External interfaces would include a UI scripting mechanism, giving suppliers and vendors more control over their price curves used by the supply chain contract. This could also include additional features not currently supported, such lead time based pricing.

The Designer contract should have an associated website that will compile the bill of materials for all materials used by the designer and the associated manufacturing services. This site should incorporate secure file transfer to send designs securely to the manufacturer.

The customer bids contract should have an associated website that will be similar to other online shopping and bidding websites like Kickstarter.

Further research is required on the ideal price point execution algorithms, and the most efficient implementation. Research is also required on the incentives and protocols for the executioner. Because running a solver algorithm takes considerable computation, it is worth revisiting the incentive mechanism to ensure that purchases are triggered reliably.