Peer to Peer and Block Chain

Universit di Pisa

Scuola Superiore SantAnna

Development of a Dapp for Smart Auctions

Author:
Gianluca D'Amico
Id: 575750
damicoglc@gmail.com

Professoressa:
Lausa Ricci

27 luglio 2019

1 Scelte di progettazione

Partendo dal final-term sono state eseguite differenti modifiche ai vari contratti, sia alla struttura del progetto stesso. I contratti scritti in *Solidity* sono adesso 3: `EnglishAuction`, `VicreyAuction` e `AuctionManager`. Quest'ultimo stato inserito per gestire la creazione, la distruzione delle varie aste da parte degli utenti e per dare un punto di riferimento per l'utilizzo dei contratti.
L'`AuctionManager` il contratto che viene caricato all'interno delle *Block-chain*, all'interno sono presenti tutti gli indirizzi corrispondenti ai contratti delle aste attive. Grazie ad una variabile booleane `couldCreateAuction` possibile impedire o consentire la creazioni delle aste, di seguito sono riportate delle brevi descrizioni dei metodi definiti:

- `createEnglishAuction(...)`: se `couldCreateAuction == true`, crea una nuova istanza di un contratto `EnglishAuction`, chiamando il suo costruttore e salvandone l'indirizzo;
- `createVicreyAuction(...)`: se `couldCreateAuction == true`, crea una nuova istanza di un contratto `VicreyAuction`, chiamando il suo costruttore e salvandone l'indirizzo;
- `deleteEnglishAuction(...)`: rimuove un indirizzo dalle aste `EnglishAuction` salvate, questa funzione di tipo `internal` (pu essere richiamata solo da altri contratti);
- `deleteVicreyAuction(...)`: rimuove un indirizzo dalle aste `VicreyAuction` salvate, questa funzione di tipo `internal` (pu essere richiamata solo da altri contratti);
- `getAllAuctions()`: restituisce tutte le aste attive in due array di indirizzi;
- `stopCreation()`: se `couldCreateAuction == true`, lo pone a `false`;
- `startCreation()`: se `couldCreateAuction == false`, lo pone a `true`;
- `destroyManager()`: se non vi sono aste attive, chiama il metodo `selfDestruct`.

LA flag `couldCreateAuction` stata pensata per permettere al "manager" di fermare le future creazioni di aste, in modo tale da poter richiamare il metodo `selfDestruct` senza perdere riferimenti ad aste attive. Infatti, per poter interagire con le varie aste tramite la **DAPP** sviluppata, occorrono gli indirizzi salvati all'interno del contratto "manager".

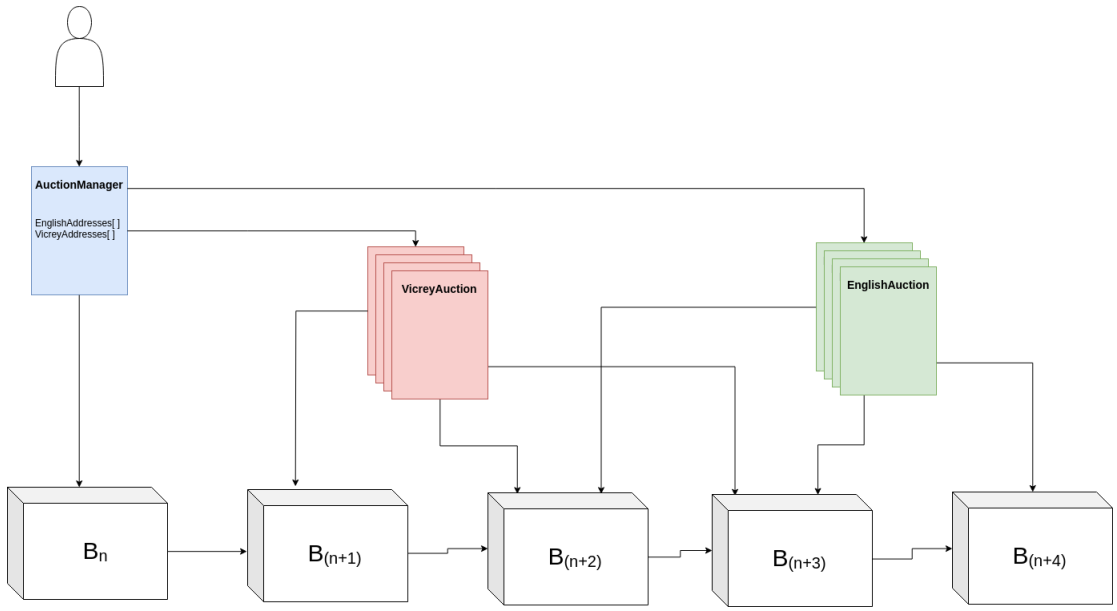

Figura 1: Diagramma della struttura dei contratti.

Questo sistema garantisce che, l'unico indirizzo che occorre salvare all'interno di un server (lo stesso server che fornisce il codice della **DAPP**) sar quello dell'`AuctionManager`, aumentando cos la sicurezza dell'applicazione. Inoltre,

il "manager" si fa carico della consistenza delle aste e dei pagamenti. All'interno dei contratti `EnglishAuction` e `VicreyAuction` stata infatti aggiunta una fase dopo la finalizzazione dell'asta. In quest'ultima fase, il pagamento del vincitore viene "congelato" nel deposito del contratto stesso, permettendo all'utente di ricevere il bene prima di estinguere il pagamento (un po' come funziona PayPal). Nel caso in cui per, il vincitore non dovesse estinguere il pagamento anche dopo aver ricevuto il bene, il "manager" (i.e. l'account che ha instanziato il contratto `AuctionManager`) pu estinguere egli stesso il pagamento nel caso in cui sia passato un sufficiente lasso di "tempo" (per la precisione si parla di numero blocchi minati, tale periodo stato impostato a 1440 blocchi, circa 5 giorni).
Inoltre, l'`AuctionManager`) pu eventualmente cambiare la fase delle aste. Le fasi vengono scandite dal trascorrere dei blocchi, ogni 5 blocchi la fase **puo'** cambiare, ma essa non cambia fintanto che qualcuno non interagisce con il contratto. Ogni funzione infatti include un `modifier` che controlla se siano passati o meno 5 blocchi dall'ultimo cambio di fase e, in caso affermativo, passa alla fase successiva (in particolare ci non avviene nelle fasi di `finilizing` e `pending`). Il venditore dell'asta (i.e. l'account che l'ha creata) per pu richiamare la funzione `nextPhase` per cambiare manualmente fase, sempre rispettando il periodo dei 5 blocchi. Potrebbe accadere che se nessuno fosse interessato all'asta anche il venditore potrebbe non interagire pi con il contratto e lasciare che esso non venga mai distrutto. Perci, anche il "manager" ha la possibilit di cambiare manualmente fase attraverso la stessa funzione.
Infine si deciso di utilizzare un server MySql per salvare "Titolo" e "Descrizione" delle varie aste. Utilizzare direttamente i contratti per salvare stringhe di grande dimensioni costoso in quanto l'utilizzo efficiente e minimale della memoria all'interno della blockchain una propriet fondamentale per lo sviluppo di un buon SmartContract, in quanto l'operazione associata allo `STORAGE` di variabili in memoria la pi costosa di tutte. Da un punto di vista della sicurezza, questa decisione non implica nessuna minaccia in quanto "attaccare" il database per modificarne le informazioni salvate al suo interno, non produrrebbe alcun effetto sulla blockchain e potrebbe essere facilmente individuato e corretto.

2 Scelte implementative

2.1 Contratto AuctionManager

Nel particolare nel contratto `AuctionManager`) si scelto di salvare gli indirizzi delle aste attiva tramite l'utilizzo di un `mapping` (usando come chiavi l'indirizzo del contratto) e di un `array` di supporto. La prima viene utilizzata per controllare l'esistenza di un dato indirizzo all'interno di quelli attivi, in modo da avere un accesso a costo unitario. L'`array` viene invece utilizzato laddove necessario usare l'indirizzo stesso del contratto, infatti in `Solidity` non possibile accedere alle chiavi di un `mapping`. Visto questo tipo di struttura, quando occorre eliminare un'asta dal contratto "manager" necessario spostare l'ultimo elemento dell'`array` nella posizione lasciata vuota dall'elemento eliminato e , soprattutto, bisogna diminuire la lunghezza dell'`array`. Gli `array` dinamici, infatti, mantengono la proprio dimensione anche dopo aver chiamata la funzione `delete` sul l'elemento specificato, portando a comportamenti indesiderati soprattutto nel caso in cui si vogliano accedere gli elementi nelle utlime posizioni.

2.2 DAPP

Per lo sviluppo della DAPP, sono stati necessari diversi FrameWork e libreria in modo da impostare tutto l'environment necessario per testare e implementare sia i contratti che la DAPP stessa. Per la precisione si fatto uso di:

- **Truffle**: un FrameWork necessario a testare e deployare i contratti;
- **Web3**: una libreria necessaria per interagire tramite javascript con i contratti sulla blockchain;
- **Truffle-contracts**: una libreria simile a Web3, con qualche funzionalit in meno, rende l'interazione con i contratti pi semplice ed intuitiva;
- **NodeJS**: un FrameWork utilizzato per gestire la DAPP sviluppata in javascript e jquery, e per interagire con Truffle;
- **lite-server**: una libreria utilizzata come server per "servire" la DAPP e i suoi moduli;
- **bn.js**: una libreria in javascript necessaria a manipolare i valori dei numeri utilizzati dai contratti della blockchain (e.g. uint256, o valori in Wei), evitando errori di integer overflow e di precisione;

- **MetaMask**: un'estensione del browser che permette di collegarsi ai nodi della blockchain, in modo da poter interagire con diverse blockchain sia di testi che reali;
- **Ganache-cli**: una blockchain locale di testing, permette di simulare una vera rete su cui possibile deployare e conseguentemente interagire con i contratti scritti in Solidity;
- **Boostrap**: una serie di librerie di stile CSS e javascript per renderizzare la DAPP;
- **Express** : una libreria javascript server-side per salvare i dati delle aste quali "titolo" e "descrizione" in un server MySql;
- **Truffle-assertion** : una libreria di NodeJS per testare i contratti.

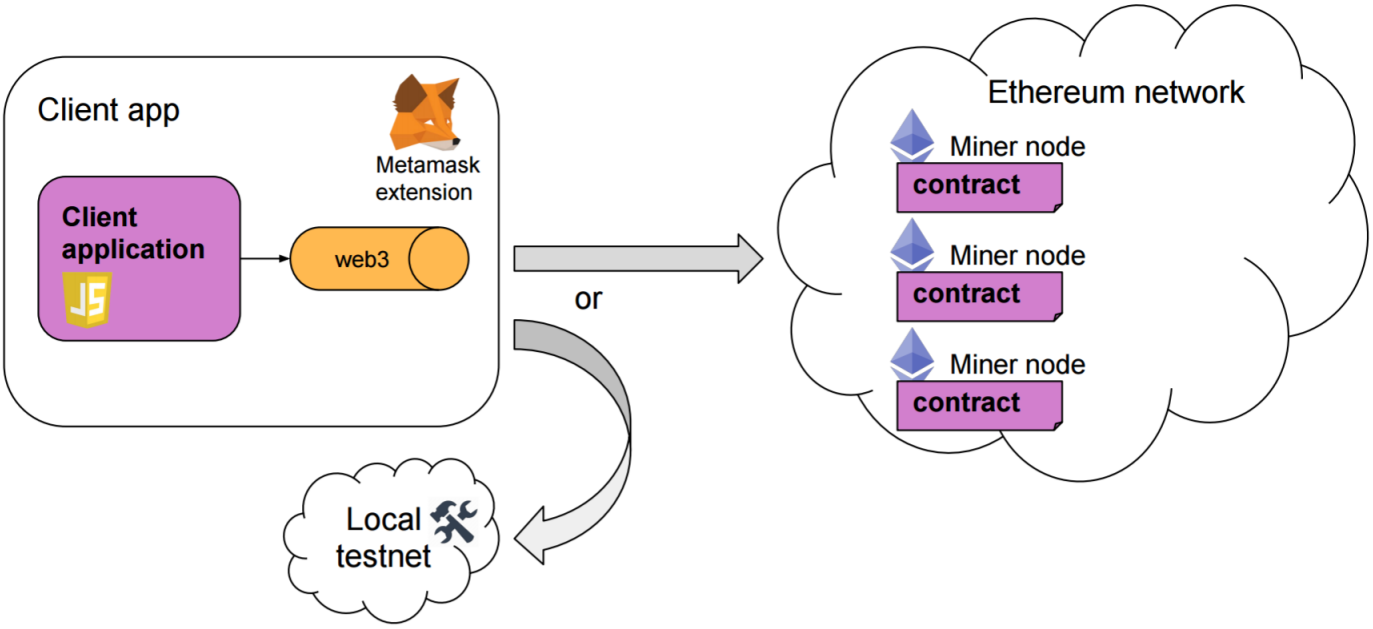

Figura 2: Environment parziale per interfacciarsi al mondo delle blockchain.

Prima di illustrare il funzionamento della DAPP nello specifico, vanno fatte alcune considerazioni sul "versioning" delle varie parti utilizzate per il suo sviluppo. In particolare la versione di Web3 utilizzata la 1.0.0.beta37, essendo una versione in via di sviluppo non tutti i metodi funzionano correttamente soprattutto dato che l'environment necessario composto da un grande numero di FrameWork e libreria che devono interagire tra loro.
La DAPP si presenta divisa in tre sezione, una per tipologia di contratto (i.e. `EnglishAuction`) e `VicreyAuction`) pi una sezione che varia a seconda di quale sia l'account di MetaMask utilizzato per collegarsi alla DAPP. Nel caso in cui si connetta con l'indirizzo "manager" verranno mostrati tutti i contratti salvati all'interno del'`AuctionManager`) (nelle rispettive sezioni) che si trovano nella fase di "Pending" e in pi verranno mostrati tutti quei contratti in cui la fase non cambia da pi di 546 blocchi (i.e. circa 2 giorni). In questo modo al "manager" verranno mostrati solo i contratti di vero interesse e non tutti gli altri (con cui nello specifico non pu interagire se non per far progredire la fase). La terza sezione comprende alcune utilit che permettono al "manager" di bloccare la creazione di altre aste e in caso di distruggere il contratto stesso.
Se invece ci si connette con un account qualsiasi che ha o meno interagito con i vari contratti gi "deployati", le prime due sezioni mostreranno tutti i contratti con cui l'utente pu interagire, mostrando le varie fasi in cui essi si trovano. Nello specifico verranno mostrati quei contratti che seguono le seguenti specifiche:
`EnglishAuction`:

- Fase = **Glory Phase**, l'utente pu comprare il bene prima dell'inizio dell'asta;
- Fase = **Glory Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, l'utente non pu comprare il bene ma pu sottomettere una puntata (cambiando di conseguenza la fase tramite il `modifier`);
- Fase = **Submitting Phase**, l'utente pu sottomettere la sua puntata;

- Fase = **Submitting Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, in questo caso tale asta verr mostrata se e solo se l'utente il vincitore dell'asta. In tal caso avr la possibilit di finalizzare l'asta, passando automaticamente alla fase successiva;
- Fase = **Finilizing Phase**, se e solo se l'utente il vincitore dell'asta, permettendogli di finalizzare l'asta;
- Fase = **Pending Phase**, se e solo se l'utente il vincitore dell'asta, permettendogli di estinguere il pagamento l'asta;

`VicreyAuction`:

- Fase = **Glory Phase**, l'utente non pu interagire con l'asta in alcun modo;
- Fase = **Glory Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, l'utente pu "committare" una puntata (cambiando di conseguenza la fase tramite il `modifier`);
- Fase = **Committing Phase**, l'utente pu "committare" la sua puntata;
- Fase = **Committing Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, in questo caso tale asta verr mostrata se e solo se l'utente tra i partecipanti (i.e. ha inserito na puntata). Inoltre l'utente potr direttamente scartare la sua puntata in quanto la fase cambier in automatico;
- Fase = **Whitdrawal Phase**, se e solo se l'utente tra i partecipanti, permettendogli di scartare la propria puntata;
- Fase = **Whitdrawal Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, in questo caso tale asta verr mostrata se e solo se l'utente tra i partecipanti (i.e. non ha scartato la sua puntata), l'utente potr quindi aprire la sua puntata;
- Fase = **Opening Phase**, se e solo se l'utente tra i partecipanti, permettendogli di aprire la propria puntata;
- Fase = **Opening Phase**, ma sono passati pi di 5 blocchi dall'ultimo cambio di fase, in questo caso tale asta verr mostrata se e solo se l'utente il vincitore dell'asta, permettendogli di finalizzarla;

Le ultime due fasi sono equivalenti al caso dell'`EnglishAuction`. Inoltre, le aste di cui l'utente il venditore, mostreranno sempre un pulsante per cambiare fase (disabilitato nel caso in cui non sia passato il tempo necessario), permettendogli inoltre di finalizzare l'asta (senza cambiare fase) se si trovasse nella dase di Opening Phase al momento giusto. L'ultima sezione invece comprende un form per creare una nuova asta, di qualsiasi tipo, per mettere in vendita un qualunque bene.
Tutte le istanze dei contratti che vengo visualizzate nella DAPP (e.g. quelle con cui l'utente pu interagire), si sottoscrivono a tutti gli eventi di interesse che portino a cambiare lo stato dell'asta stesso (e.g. cambio di fase, sottomissione di una puntata pi alta, finalizzazione o estinzione di un asta di cui l'utente proprietario etc...). Inoltre ogni asta pu essere aggiornata singolarmente tramite un apposito pulsante che ne ricarica le varie informazioni.

3 Testing

3.1 Test Contratti

Per testare i contratti sono stati scritti dei test specifici con Truffle in cui vengono testate alcune situazioni particolari. Per `EnglishAuction` vengono eseguiti 4 test specifici:

- "Test _Buy Out the good": si eseguono una serie di transazioni per comprare il bene prima che cominci l'asta;
- "Test _Making a bid": si eseguono una serie di transazioni per eseguire una puntata durante l'asta;
- "Test _Same address 2 bids _A lower bid!": prima si cerca di fare due puntate dallo stesso account, poi si cerca di fare una puntata pi bassa di quella attualmente vincente;
- "Test _No bids": si crea e conclude l'asta senza fare alcuna puntata;

Per `VicreyAuction` vengono i seguenti test:

- "Test _deposit transaction": si esegue un withdraw e si controllo che venga perso met del deposito;

- "Test _Highest Bid and Second highest must change": si eseguono una serie di transazioni per arrivare alla fase di opening e controllare che sia il primo che il secondo vengano aggiornati correttamente;
- "Test _No bids": si crea e conclude l'asta senza fare alcuna puntata;

Per eseguire tali test stato utilizzato Ganache con la funzione di "automining" attiva, la fase stata controllata e fatta avanzare tramite un ciclo while richiama la fase del contratto. I test infatti risultano essere lenti, soprattutto necessario impostare l'automining al di sopra dei 5 secondi per non incorrere in errori.
Il contratto `AuctionManager` stato invece testato direttamente sulla DAPP in quanto non presenta particolari casistiche.

3.2 Test DAPP

Per eseguire un test sul funzionamento della DAPP occore installare MetaMask nel browser e lanciare sia lite-server (port:3000) che express (port:5000). Dall'icona di MetaMask possibile selezionare la network di testing (port:8545) che viene emulata ancora una volta con Ganache. Per far si che MetaMask si connetta ad un account della block chain, necessario copiare uno degli indirizzi che Ganache mette a disposizione (il numero 3 stato utilizzato come predefinito) e copiare Api key all'interno di MetaMask. Inoltre, necessario deployare i contratti, a tal proposito, nel file `truffle-config.json` sono state definite due network. La prima quella utilizzata per i test su Ganache, la seconda viene utilizza per la connessione alla block chain di test di Ropsten. Per deployare il contratto manager e eseguire la Dapp, necessario eseguire i seguenti comandi in terminali diversi (di cui gli ultimi due vanno eseguiti nella cartella contenete il progetto):

```
$ ganache-cli -p 8545 -b 5
$ truffle migrate --reset --network  development
$ npm run dev
```

Il flag "network" indica su quale rete verr eseguito il file di "migrazione" dei contratti. Nel file `2_deploy_contracts.js` sono definiti i vari comandi per deployare i diversi contratti a seconda della rete in cui ci si trova. Quindi connettendosi a Ganache verranno istanziati diversi contratti per popolare appropriatamente la DAPP durante il test. In particolare si consiglia di configurare MetaMask con l'account numero 3, o con il numero 0 se si vuole vedere l'interfaccia dal punto di vista del manager. Non essendo necessario un login per diversificare gli account in quanto gi lo sono, occorre settarli manualmente all'interno di MetaMask.

3.3 Test Ropsten

Infine il progetto stato testato anche su Ropsten, con qualche difficolt iniziale, utilizzando come nodo di accesso Infura. Baster eseguire lo stesso comando visto in precedenza senza la flag `--reset` e con la giusta network. Non stato possibile eseguire molti test su tale rete in quanto non si posseggono lo stesso numero di account che offre Ganache e tanto meno lo stesso ammontare di Ether. In ogni contratto inoltre raggiungibile un metodo di `selfdestruct` per evitare di lasciare contratti svincolati sulla network pubblica.