UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Blockchain e Smart contract: un esempio di applicazione in linguaggio Solidity

Blockchain and Smart contract: an example of application in Solidity language

Relatore:
Prof. Luca Spalazzi

Laureando:
Ludovico Nardi

Anno Accademico 2020-2021

Indice

1 Introduzione **7**

1.1 Ambito della tesi 7
1.2 Obiettivo della tesi 7
1.3 Struttura della tesi 8
1.3.1 Smart contract, blockchain ed Ethereum 8
1.3.2 Solidity e l'IDE web Remix 8
1.3.3 Orchestrazione e coreografie di processi 8
1.3.4 Architettura e implementazione per forzare la conformità dei processi alle coreografie 8
1.3.5 Potenzialità e limiti di Solidity 8
1.3.6 Conclusioni e sviluppi futuri 8

2 Smart contract, blockchain ed Ethereum **9**

2.1 Smart contract 9
2.1.1 Storia 9
2.1.2 Cos'è uno smart contract e come funziona 10
2.2 Blockchain 11
2.2.1 Introduzione alle blockchain e legame con gli smart contract 11
2.2.2 Struttura e funzionamento di una blockchain . . . 12
2.2.3 Caratteristiche e vantaggi delle tecnologie blockchain 13
2.2.4 Utilizzo di blockchain e database 14
2.3 Ethereum 15
2.3.1 Cos'è Ethereum e come funziona 15
2.3.2 Ether 16
2.3.3 Indirizzi 17

3 Solidity e l'IDE web Remix **19**
3.1 Solidity 19
3.1.1 Interi 19
3.1.2 Conversione di tipo 20
3.1.3 Funzioni 20
3.1.4 Address 20
3.1.5 Boolean 20
3.1.6 Mapping 21
3.1.7 Struct 21
3.1.8 Array 21
3.2 IDE web Remix 22
3.2.1 Remix 22
3.2.2 Truffle 23

4 Orchestrazione e coreografie di processi **25**
4.1 BPMN(Business Process and Model Notation) 25
4.2 Modelli del BPMN 25
4.2.1 Processi 25
4.2.2 Coreografie 26
4.2.3 Collaborazioni 27
4.2.4 Conversazioni 27

5 Architettura e implementazione per forzare la conformità dei processi alle coreografie **29**
5.1 Codice e funzioni 31
5.1.1 Address 31
5.1.2 messages e numtasks 31
5.1.3 Step 31
5.1.4 Configuration 31
5.1.5 Mapping 32
5.1.6 taks 32
5.1.7 Funzione "get" 32
5.1.8 Funzione "add" 32
5.1.9 Funzione "enableTask" 32
5.1.10 Funzione "disableTask" 33
5.1.11 Modifier "isEnabled" 33

5.1.12 Funzione "getTasks" 33
5.1.13 Funzione "send" 34
5.1.14 Modifier "messageSent" 34
5.1.15 Funzione "signActor1" 34
5.1.16 Funzione "signActor2" 34

6 Potenzialità e limiti di Solidity **35**
6.1 Potenzialità di Solidity 35
6.2 Limiti di Solidity 36

7 Conclusioni e sviluppi futuri **37**
7.1 Conclusioni 37
7.2 Sviluppi futuri 37

8 Ringraziamenti **39**

Elenco delle figure **41**

Bibliografia **43**

Capitolo 1

Introduzione

1.1 Ambito della tesi

L'ambito di questa tesi è quello delle blockchain, degli smart contract, e la loro scrittura in un linguaggio di programmazione.
Il linguaggio scelto per questa tesi è Solidity e ciò è anche legato alla selezione dell'IDE web Remix, dedicato a Solidity per l'appunto.

1.2 Obiettivo della tesi

L'obiettivo di questa tesi è quello di conoscere il panorama tecnologico delle blockchain, il legame con gli smart contract e realizzare uno smart contract in linguaggio Solidity, che permetta di forzare la conformità dei processi ad un dato digramma di coreografia.

1.3 Struttura della tesi

1.3.1 Smart contract, blockchain ed Ethereum

In questo capitolo introdurremo le blockchain, gli smart contract ed Ethereum analizzandone alcuni aspetti specifici.

1.3.2 Solidity e l'IDE web Remix

In questo capitolo introdurremo le basi del linguaggio Solidity e analizzeremo la scelta di Remix rispetto al suo concorrente Truffle.

1.3.3 Orchestrazione e coreografie di processi

In questo capitolo introdurremo la notazione BPMN, orchestrazione e coreografia.

1.3.4 Architettura e implementazione per forzare la conformità dei processi alle coreografie

In questo capitolo vedremo il codice dello smart contract sviluppato al fine di forzare la conformità dei processi alla coreografia data.

1.3.5 Potenzialità e limiti di Solidity

In questo capitolo vedremo gli strumenti messi a disposizione dal linguaggio Solidity e i suoi punti a sfavore.

1.3.6 Conclusioni e sviluppi futuri

Conclusioni e spunti di ulteriore sviluppo.

Capitolo 2

Smart contract, blockchain ed Ethereum

2.1 Smart contract

2.1.1 Storia

Gli smart contract sono stati oggetto di sperimentazione negli anni '90, ma l'idea di "contratto intelligente" si può far risalire già agli anni '70 [1] in relazione alla necessità di gestire l'attivazione o disattivazione di una licenza software in funzione di determinati requisiti [2].
Uno dei primi a effettuare tali sperimentazioni e a coniare il nome stesso fu Nick Szabo, un esperto di crittografia statunitense di origini ungheresi [3] il quale iniziò a ipotizzare già nel 1993 , che determinati oggetti potevano essere gestiti in modo digitale sulla base di specificate condizioni.
Il termine "smart contract" è stato coniato negli anni '90 [4] Scrive infatti Nick Szabo: *"L'idea di base dello smart contract è che molti tipi di clausole contrattuali (come la garanzia, l'assunzione dell'obbligazione, la delimitazione di un diritto di proprietà, ecc.) possono essere incorporati nell'hardware e nel software che trattiamo, in modo da rendere la violazione del contratto costosa (se desiderato, addirittura proibitiva) per il soggetto inadempiente".* [5]

2.1.2 Cos'è uno smart contract e come funziona

Con smart contract si intende la redazione di un contratto contenente delle condizioni/clausole che devono essere rispettate per fare sì che le definizioni operative possano essere compiute.
La logica che viene rispettata è quella del "if-this-then-that", ovvero "se questo accade allora succede".
Uno smart contract è la trasposizione in codice di un contratto in modo da verificare in automatico il verificarsi di determinate condizioni (controllo di dati di base del contratto) e di autoeseguire in automatico azioni, dare disposizione affinché si possano eseguire determinate azioni, nel momento in cui le condizioni stabilite tra le parti sono raggiunte e verificate.
Dunque uno smart contract è basato su un codice che legge sia le clausole che sono state concordate sia la condizioni operative nelle quali devono verificarsi le condizioni concordate e si autoesegue automaticamente nel momento in cui i dati riferiti ai casi reali corrispondono ai dati riferiti alle condizioni e alle clausole concordate.
Lo smart contract è dunque un programma per elaboratore che opera su tecnologie basate su registri distribuiti e la cui esecuzione vincola automaticamente due o più parti sulla base di effetti predefiniti dalle stesse.
Lo smart contract può essere immaginato come un programma la cui esecuzione e i cui risultati sono garantiti integri dalle proprietà di una blockchain pubblica [6], tale accezione discende dalla scelta del progetto Ethereum di denominare tale codice in esecuzione come smart contract.

2.2 Blockchain

2.2.1 Introduzione alle blockchain e legame con gli smart contract

Gli smart contract, qualora inseriti in una Blockchain, hanno la possibilità di un enforcement pressoché automatico ed immodificabile delle obbligazioni in essi contenute. [7]
La blockchain è un registro condiviso e immutabile che facilita il processo di registrazione delle transazioni e di tracciamento degli asset in una rete di business.
Un asset può essere tangibile (case, soldi, automobili) o intangibile (proprietà intellettuali, marchi, copyright).
Praticamente qualsiasi cosa che abbia un valore può essere rintracciata e scambiata su una rete blockchain, abbattendo i rischi e i costi per tutti gli interessati.
La blockchain è importante anche perché il business dipende dalle informazioni.
Se sono rapide e accertate, ciò costituisce un vantaggio.
La blockchain è ideale per veicolare queste informazioni perché offre informazioni immediate, condivise e completamente trasparenti archiviate in un registro immutabile a cui possono accedere solo i membri di rete autorizzati.
Una rete blockchain può, tra le altre cose, tracciare ordini, pagamenti, account e produzione.
Essendo che i membri condividono una visione singola della verità, è possibile vedere tutti i dettagli di una transazione end-to-end, generando così maggiore trasparenza, oltre a nuove efficienze e opportunità. [8]

2.2.2 Struttura e funzionamento di una blockchain

La blockchain ("catena di blocchi") è una struttura dati condivisa e immutabile.
È definita come un registro digitale le cui voci sono raggruppate in "blocchi", concatenati in ordine cronologico, e la cui integrità è garantita tramite la crittografia.
Tale struttura è immutabile in quanto, di norma, il suo contenuto una volta scritto non è più né modificabile né eliminabile, a patto di non invalidare l'intera struttura.
Tali tecnologie fanno parte famiglia dei Distributed Ledger, cioè sistemi che si basano su un registro distribuito, che può essere letto e modificato da più nodi di una rete.
Non è richiesto che i nodi coinvolti conoscano l'identità reciproca o si fidino l'uno dell'altro.
Infatti, per garantire la coerenza tra le varie copie, l'aggiunta di un nuovo blocco è globalmente regolata da un protocollo condiviso.
Una volta autorizzata l'aggiunta del nuovo blocco, ogni nodo aggiorna la propria copia privata: la costruzione intrinseca di questa struttura dati garantisce l'assenza del rischio di una sua manipolazione futura.

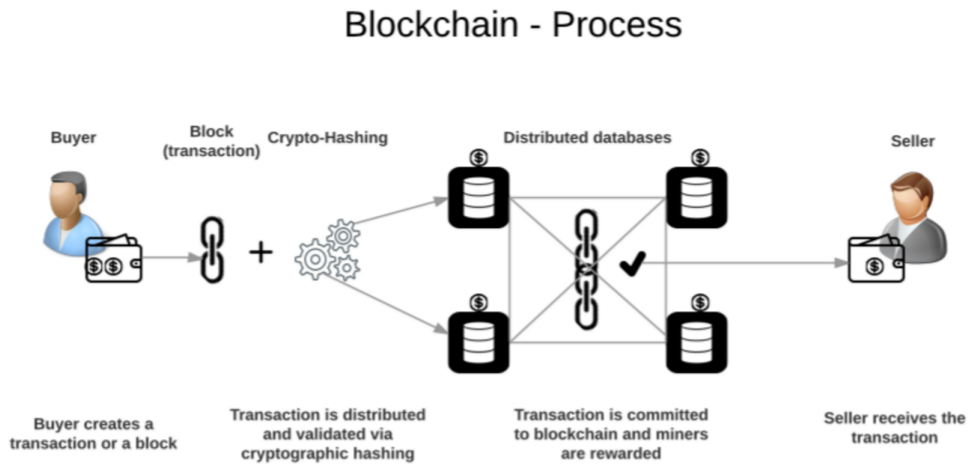

Figura 2.1: B140970324, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

2.2.3 Caratteristiche e vantaggi delle tecnologie blockchain

Le caratteristiche che accomunano i sistemi sviluppati con le tecnologie Blockchain e Distributed Ledger sono digitalizzazione dei dati, decentralizzazione, disintermediazione, tracciabilità dei trasferimenti, trasparenza/verificabilità, immutabilità del registro e programmabilità dei trasferimenti.
Grazie a tali caratteristiche, la blockchain è considerata pertanto un'alternativa in termini di sicurezza, affidabilità, trasparenza e costi alle banche dati e ai registri gestiti in maniera centralizzata da autorità riconosciute e regolamentate (pubbliche amministrazioni, banche, assicurazioni, intermediari di pagamento, ecc.). [9]
La blockchain utilizza una rete decentralizzata.
Decentramento ovvero tutti i nodi della rete memorizzano un backup della blockchain.
I nodi memorizzano i backup di tutti i nodi, sia di mining, sia di entrambi.
Non ci sono amministratori per verificare i trasferimenti di blocchi.
C'è bisogno di un minatore in grado di risolvere il problema di crittografia, ma questo è basato sulla proporzione dell'intera potenza di calcolo della rete.
Aggiunto il blocco alla catena, l'informazione è immutabile e trasparente.

2.2.4 Utilizzo di blockchain e database

Per la sua caratteristica di stabilità, il database, è più adatto alle reti aziendali.
Sono anche più semplici da utilizzare per gli utenti e hanno molti sistemi di gestione di supporto per amministratori e sviluppatori.
Il database può scalare fino a milioni di record e può elaborare migliaia di trasferimenti al secondo.
Per il trattamento di grandi quantità di traffico, il database è la soluzione migliore.
La blockchain non ha bisogno di memorizzare una grande quantità di elaborazione per l'analisi, il database può archiviare più dati e la velocità di elaborazione è più veloce in quanto i nodi non sono necessari.
Oltretutto, non è necessario crittografare tutti i dati.
Spesso, il db non è crittografato perché la crittografia aggiunge molta ridondanza al database.
I database tradizionali utilizzano l'autorizzazione per aumentare la crittografia.
Utilizzare una blockchain per memorizzare informazioni private, può essere costoso.
Le informazioni che solo alcune aziende possono conoscere, come la sicurezza sociale e le cartelle cliniche, sono archiviate nel database.
Le informazioni che possono essere utilizzate dal sistema di verifica pubblica possono essere basate sulla blockchain.
Queste informazioni personali possono essere autenticate sulla blockchain in base a un algoritmo di crittografia a chiave pubblica. [10]

2.3 Ethereum

2.3.1 Cos'è Ethereum e come funziona

Ethereum è una piattaforma decentralizzata del Web 3.0 per la creazione e pubblicazione peer-to-peer di contratti intelligenti smart contracts. [11]
Per poter girare sulla rete peer-to-peer, i contratti di Ethereum pagano l'utilizzo della sua potenza computazionale tramite una unità di conto, detta Ether, che ha il ruolo quindi sia di criptovaluta sia di carburante.
In altre parole, a diffrenza di molte altre criptovalute, Ethereum non è solo un network per lo scambio di valore monetario, ma una rete per far girare contratti basati su Ethereum.
Tali contratti possono essere utilizzati in maniera sicura per eseguire un vasto numero di operazioni: sistemi elettorali, registrazione di nomi di dominio, mercati finanziari, piattaforme di crowdfunding, proprietà intellettuale.
La validità di ciascun Ether è garantita da una blockchain, che è un elenco di record in continua crescita, chiamati blocchi, i quali sono collegati tra loro e protetti mediante crittografia.
Per definizione, la blockchain è in sé resistente alla modifica dei dati.
È un libro mastro contabile, aperto e distribuito che registra le transazioni tra due parti in modo efficiente e permanentemente verificabile.

2.3.2 Ether

Ether fornisce un libro mastro per le transazioni.
È utilizzato per pagare il gas, un'unità di calcolo per le transazioni e altre transizioni di stato.
È indicato con l'icona di transazione ETH, in modo tale da poter essere scambiato nei mercati elettronici denominati in criptovalute, cioè pagare le commissioni di transazione e servizi aggiuntivi alla rete Ethereum.

2.3.3 Indirizzi

Gli indirizzi della rete Ethereum iniziano e sono identificati dal prefisso "0x", comune per i numeri in base 16, seguito dai 20 byte più a destra (ordine dei byte) dell'hash Keccak-256 della chiave pubblica ECDSA dove la curva utilizzata è chiamata secp256k1.
Gli indirizzi Ethereum contengono 40 cifre esadecimali.
Un esempio di indirizzo Ethereum è il seguente:
0x7949635E2877ef8ca37B8526507AC214B0423Ebf.
Gli indirizzi di contratto sono nello stesso formato, ma sono determinati dal mittente e dal nonce, cioè uno scalare di valore pari al numero di transazione inviato dal mittente (contract creation).
In altre parole, i conti utente sono indistinguibili dai conti-contratto, ai quali viene associato un singolo indirizzo per ognunoe nessun dato blockchain.
Un utente, invece, può avere molteplici livelli di chiave privata, dai quali sono creati altrettanti indirizzi Ethereum.
Ogni hash Keccak-256 valido inserito nel formato descritto è valido, anche se non corrisponde a un account con una chiave privata o a un contratto.

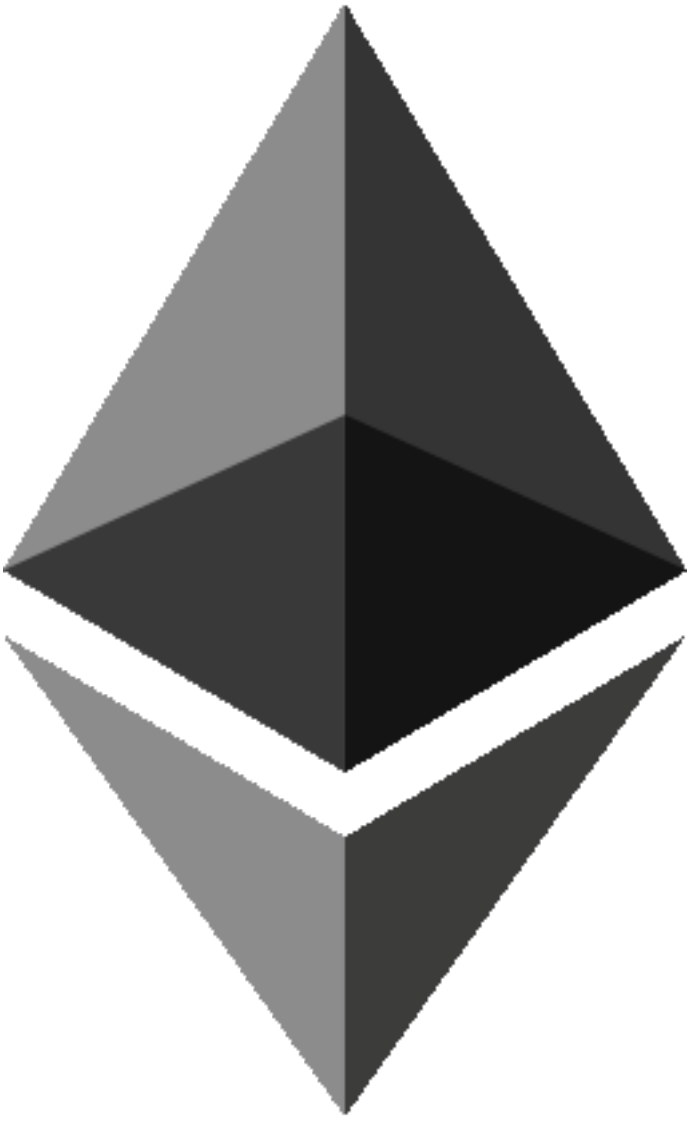

Figura 2.2: Ethereum, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>, via Wikimedia Commons

Capitolo 3

Solidity e l'IDE web Remix

3.1 Solidity

Solidity è un linguaggio tipizzato. Ecco i tipi di variabile che è possibile utilizzare in uno Smart Contract:

3.1.1 Interi

uint, uint8, uint16, uint24, uint32, uint64, uint128 e uint256 Uint significa unsigned integer.
Ad esempio, uint8 è in grado di memorizzare qualsiasi numero positivo con il numero massimo di 2^8 che è 256.
Dichiarando una variabile di tipo uint di default verrà dichiarata una variabile di tipo uint256.

```
pragma solidity 0.5.12;
contract Prova {
uint256 num;
}
```

3.1.2 Conversione di tipo

```
uint8 num2 = uint8(num); // num viene convertito in uint8 da uint256
```

3.1.3 Funzioni

```
function modificaNumero (uint256 _valore) public { num = _valore; }
//assegna il valore del param _valore alla variabile num
//la parola public vuol dire che questa funzione può essere eseguita
//da qualsiasi contratto e utente [11]
```

3.1.4 Address

É l'indirizzo pubblico di ogni utente.
Creando un account, si ottiene un indirizzo che le persone possono utilizzare per inviare Ether.
É possibile memorizzare tali indirizzi sul contratto per questioni come limitare l'accesso al contratto a determinati indirizzi o semplicemente registrare chi sta usando il codice.
Vediamo come si dichiara una variabile per l'indirizzo:

```
pragma solidity 0.5.12;
 contract Prova {
 uint256 provaValore;
 address indirizzo_proprietario;
 function modificaNumero (uint256 _value) { provaNumero = _value; }
 }
address indirizzo_proprietario = 0x055E36b2BB4eF...
```

3.1.5 Boolean

```
bool isTrue;  //è comune utilizzare isTrue come nome
//per questioni di leggibilità
```

3.1.6 Mapping

Sono un tipo speciale di variabili, consentono di memorizzare informazioni illimitate, ad esempio:
Nome: Giovanni, Età: 22
Nome: Mario, Età: 32
Nome: Bruno, Età: 25:

```
mapping (string = uint256) etaPersone;
```

Si utilizzano delle parole chiave e quindi si definiscono i tipi.
Le informazioni saranno memorizzate in questo modo:
Giovanni = 22
Mario = 32
Bruno = 25
E sarà possibile accedere alle informazioni come in un array:

```
etaPersone ["Giovanni"]; // Restituirà 35
```

3.1.7 Struct

```
struct persone {
uint256 id;
string nome;
uint256 anni;
}
```

Per creare istanze si può fare così

```
persone inserisciPersona = persone (1, "Giovanni", 35);
```

3.1.8 Array

Possono essere dichiarati così

```
uint256[] myNumbers;
bytes32[] myStrings;
string[] myTexts;
Tree[] myTrees; //dove Tree è una struttura dati
```

3.2 IDE web Remix

Nel caso di Ethereum, ci sono due grandi IDE ufficialmente supportati dalla community. Il primo è Remix [13].

3.2.1 Remix

Remix è un IDE web che possiamo utilizzare dal nostro browser e con il quale possiamo programmare e testare le nostre applicazioni scritte in Solidity senza dover installare nulla, ed quello che è stato effettivamente utilizzato per questa tesi.

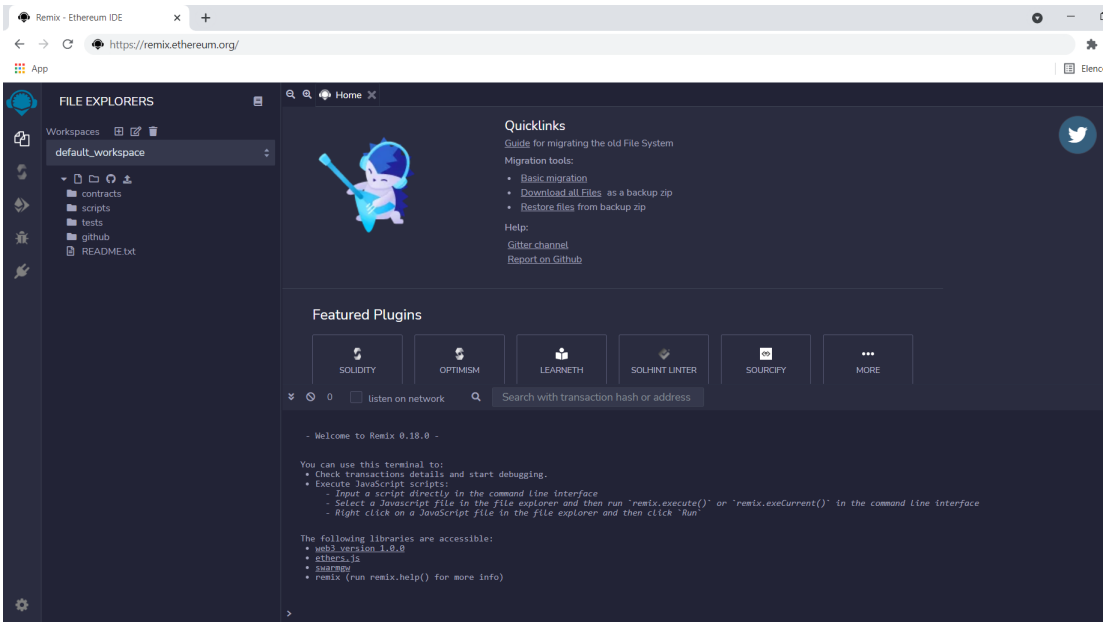

Figura 3.1: https://remix.ethereum.org/

3.2.2 Truffle

Un altro strumento ampiamente utilizzato nel mondo dello sviluppo su Ethereum e Solidity è Truffle [14].
Truffle è infatti di gran lunga lo strumento meglio integrato nel mondo Ethereum per la programmazione di Solidity.
D'altra parte, la quantità di opzioni di Truffle lo rende uno strumento complesso da usare, soprattutto se si sta appena iniziando a imparare, quindi è consigliato solo a sviluppatori con conoscenze più avanzate. [15]

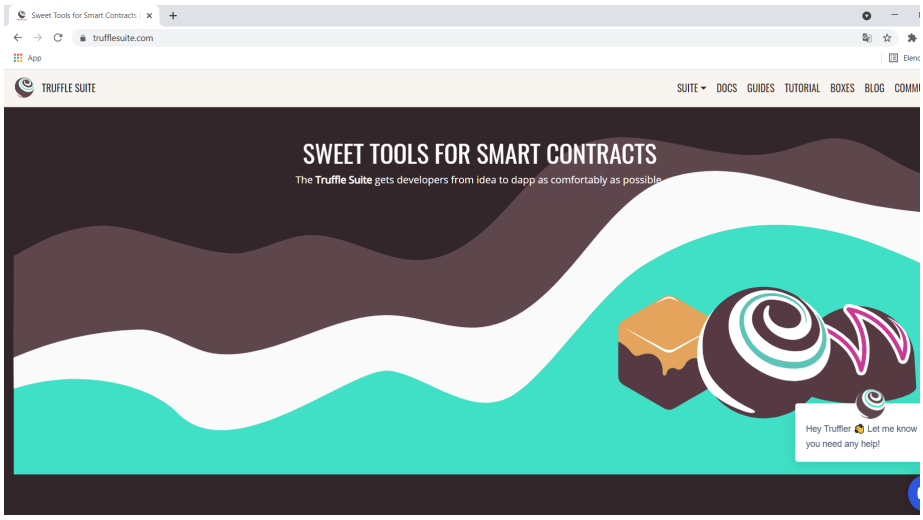

Figura 3.2: https://www.trufflesuite.com/

Capitolo 4

Orchestrazione e coreografie di processi

4.1 BPMN(Business Process and Model Notation)

Si tratta di una notazione per gli informatici e non, facile da tramutare in XML.
I cambiamenti dello standard presentati da un'impresa, vengono esaminati prima di consentire la modifica dello standard stesso.

4.2 Modelli del BPMN

4.2.1 Processi

L'orchestrazione è molto simile al diagramma di flusso.
Ogni elemento all'interno dell'orchestrazione ha una sua funzione e contribuisce a diverse azioni, alla finalità del processo.
Ogni orchestrazione si trova all'interno di una "pool" ("vasca") e può essere di due tipi: pubblica (aperta verso l'esterno) o privata (riservata ai clienti interni all'azienda).
L'orchestrazione privata ci appare nel modello come un'entità vuota.

4.2.2 Coreografie

Consentono di descrivere il modello comportamentale che si verifica tra le varie orchestrazioni.
Le coreografie rispetto alle collaborazioni ci forniscono anche la sequenza di scambio delle conversazioni tra le orchestrazioni.
Mittente chiaro, destinatario scuro; ruoli e colori si invertono nelle orchestrazioni.
Se non ci sono attività intermedie rilevanti, il messaggio inviato e quello di risposta compaiono sulla stessa attività.
Ad ogni task potrebbero corrispondere diversi stati alternativi e tutti validi della coreografia (ogni coreografia potrebbe lasciare un certo margine di manovra ai singoli attori, l'importante è che vengano rispettati i momenti di sincronizzazione).

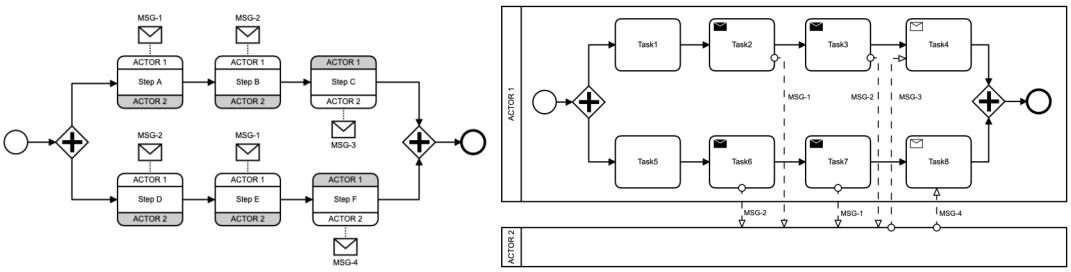

Figura 4.1: Diagrammi. Fonte: Blockchain Based Choreographies: the Construction Industry Case Study L.Spalazzi F.Spegni A.Corneli B.Naticchia

4.2.3 Collaborazioni

Consistono in uno scambio di informazioni fra due orchestrazioni ma senza conoscerne la sequenza.

4.2.4 Conversazioni

Sono un'evoluzione delle collaborazioni, per venire incontro ai molti scambi di messaggi che possono avvenire tra i vari attori.
I messaggi vengono racchiusi in argomenti di discussione.
In questo modo sarà più semplice avere una visione di insieme.
L'istanza di un processo è un'esecuzione specifica del processo, dall'inizio alla fine.

Capitolo 5

Architettura e implementazione per forzare la conformità dei processi alle coreografie

Consideriamo i seguenti diagrammi e sulla base di questi, scriviamo uno smart contract che permetta di adeguare i processi alla coreografie. [16]

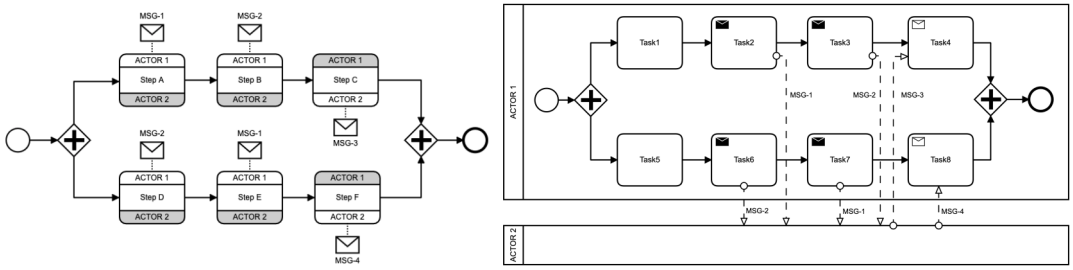

Figura 5.1: Diagrammi. Fonte: Blockchain Based Choreographies: the Construction Industry Case Study L.Spalazzi F.Spegni A.Corneli B.Naticchia

Ad ogni task che viene notarizzato bisogna identificare a quale stato della coreografia corrisponde, per decidere se il task è ammissibile o meno.
In realtà ad ogni task possono corrispondere diversi stati alternativi e

tutti validi della coreografia.
Per rappresentare una computazione dove a ogni passo si può finire in un insieme di possibili stati, tutti ammissibili, realizzare dei computation tree, ossia degli alberi.
Il fatto è che Solidity non ha come struttura dati nativa gli alberi, e realizzarli utilizzando le strutture native di Solidity risulta non banale.

5.1 Codice e funzioni

5.1.1 Address

```
address payable public actor_1 = address(0x0);
address payable public actor_2 = address(0x0);
```

5.1.2 messages e numtasks

```
string[] private messages;
uint numtasks = 6;
```

5.1.3 Step

```
// 0 : idle
// 1 : start
// 2 : Step A
// 3 : Step B
// 4 : Step C
// 5 : Step D
// 6 : Step E
// 7 : Step F
// 8 : end
```

5.1.4 Configuration

```
struct Configuration {
bytes32 name;     // name of node
bytes32 parent;   // parent node's path
//uint64 data;     // node's data
uint8[] tasks;
bytes32[] configurations;  // list of linked nodes' paths
        }
```

5.1.5 Mapping

```
mapping(bytes32 => Configuration) configurations;
```

5.1.6 taks

```
uint8[] tasks;
```

5.1.7 Funzione "get"

```
function get(bytes32 _name, bytes32 _parent) public view

returns (bytes32, bytes32, uint8[] memory, bytes32[] memory) {
     Configuration storage node = configurations[keccak256(abi.encodePacked(_parent, _name))];
        return (node.name, node.parent, node.tasks, node.configurations);
    }
```

5.1.8 Funzione "add"

```
function add(bytes32 _name, bytes32 _parent, uint8[] memory _tasks ) public {
        require(_name.length > 0);
        bytes32 path = keccak256(abi.encodePacked(_parent, _name));
            configurations[path] = Configuration({
            name: _name,
            parent: _parent,
            tasks: _tasks,
            configurations: new bytes32[](0)
        });
        configurations[_parent].configurations.push(path);
    }
```

5.1.9 Funzione "enableTask"

```
function enableTask(uint8[] memory _v, uint _n) public returns(uint8[] memory) {
     uint i = _n / DIM;      //gives the corresponding index in the array A
     uint pos = _n % DIM;    //gives the corresponding bit position in A[i]

       uint flag = 1;   // flag = 0000.....00001
     flag = flag << pos;    // flag = 0000...010...000  (shifted k positions)
     _v[i] = _v[i] | uint8(flag);    // Set the bit at the k-th position in A[i]
        //ho fatto il type-casting a uint8
    return _v;
   }
```

5.1.10 Funzione "disableTask"

```
function disableTask(uint8[] memory _v, uint _n) public returns (uint8[] memory) {

      uint i = _n / DIM;      //gives the corresponding index in the array A
      uint pos = _n % DIM;    //gives the corresponding bit position in A[i]

          uint8 flag = 1;
         flag = flag << pos; // 00001000
         flag = ~flag; // 11110111
         _v[i] = _v[i] & flag;

         return _v;
    }
```

5.1.11 Modifier "isEnabled"

```
modifier isEnabled(uint8[] memory _v, uint _n) { //

       uint i = _n / DIM;     //gives the corresponding index in the array A
       uint pos = _n % DIM;   //gives the corresponding bit position in A[i]

         uint flag = 1;   // flag = 0000.....00001

      flag = flag << pos;  // flag = 0000...010...000  (shifted k positions)
         uint temp = _v[i] & flag;

         require (temp != 0);

         _;
         //return true if _v[i] at position pos is enabled
    }
```

5.1.12 Funzione "getTasks"

```
function getTasks() public view returns (uint8[] memory) {
        return tasks;
        }
```

5.1.13 Funzione "send"

```
function send(string memory msgName) public{
        messages.push(msgName);
        }
```

5.1.14 Modifier "messageSent"

```
modifier messageSent(string memory msgName) {
        bool found = false;
        for(uint i=0; i<messages.length; i++){
        if(keccak256(abi.encodePacked(messages[i])) == keccak256(abi.encodePacked(msgName))){
                found = true;
                break;
            }
        }

        require (found);

        _;
    }
```

5.1.15 Funzione "signActor1"

```
function signActor1() public isEnabled(configurations[0].tasks, 0) {

        actor_1 = payable(msg.sender);

        if(actor_1 != address(0x0) && actor_2 != address(0x0)){
            disableTask(configurations[0].tasks, 0);
            enableTask(configurations[0].tasks, 1);
        }
    }
```

5.1.16 Funzione "signActor2"

```
function signActor2() public isEnabled(configurations[0].tasks, 0) {
        actor_2 = payable(msg.sender);
        if(actor_1 != address(0x0) && actor_2 != address(0x0)){
            disableTask(configurations[0].tasks, 0);
            enableTask(configurations[0].tasks, 1);
    }
    }
```

Capitolo 6

Potenzialità e limiti di Solidity

6.1 Potenzialità di Solidity

Solidity è un linguaggio di programmazione, creato appositamente per lo sviluppo di smart contract su Ethereum.
Oltre ad essere quello più diffuso di tutti è anche l'unico ad essere supportato ufficialmente.
Il motivo di sviluppare un linguaggio ad hoc è data dal fatto che è stato studiato per l'impiego specifico all'interno degli smart contract.
Ci rimanda un po' agli esordi dell'informatica, con architetture nuove e da costruire.
Si sarebbero potuti utilizzare anche altri linguaggi, come ad esempio il C, ma adattare un compilatore a tale scopo, sarebbe stato più costoso rispetto a sviluppare un linguaggio nuovo.
Solidity è un linguaggio tipato staticamente, supporta l'ereditarietà, librerie e tipi di dato complessi definiti dall'utente. [17]
Un contratto dal punto di vista di Solidity è una collezione di codice (le funzioni) e dati (il suo stato) che risiedono in uno specifico indirizzo sulla blockchain di Ethereum.
I contratti sono simili alle classi in linguaggi orientati ad oggetti.
Ogni contratto contiene dichiarazioni di variabili di stato, funzioni, modificatori di funzioni, strutture dati ed eventi, e ciò è un vantaggio per chi è familiare con linguaggi di programmazione object-oriented.

6.2 Limiti di Solidity

Ha un un tipo di architettura sostanzialmente nuova, dove un approccio ottimizzato sul basso livello è ancora richiesto, per via del costo di gas.
Non è possibile gestire in maniera nativa strutture dati complesse come gli alberi, e ciò comporta delle difficoltà nel gestire tali strutture.
Nell'ottica di sviluppo di smart contract, potrebbe essere utile per i programmatori l'introduzione di nuove strutture dati all'interno del linguaggio Solidity, come gli alberi appunto.
Ciò infatti comporterebbe una programmazione più lineare e veloce per lo sviluppatore, che sarebbe in grado così di focalizzarsi più sulla scrittura dello smart contract al fine di risolvere un dato problema, piuttosto che che nel bypassare verosimili "limiti tecnici" del linguaggio stesso.

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Conclusioni

Solidity è un linguaggio relativamente nuovo, così come la tecnologia blockchain e gli smart contract.
Tali tecnologie sono in continuo sviluppo, e costituiscono nuove frontiere da studiare ed esplorare.
Il linguaggio Solidity, come abbiamo visto, per ora non permette di gestire in maniera agevole strutture dati complesse come ad esempio gli alberi, ma allo stesso tempo è un linguaggio creato e dedicato alla scrittura di smart contract.
Solidity, è ancora in fase di sviluppo e verosimilmente verrà aggiornato, per colmare alcune lacune, magari anche quelle riscontrate e analizzate nel corso di questa tesi.

7.2 Sviluppi futuri

Prendendo consapevolezza di ciò sarebbe molto interessante a partire da questa tesi, e da altre fonti analoghe, sviluppare qualcosa di più ampio.
In particolare partendo da una conoscenza magari di base, ma proprio per questo fondamentale, di queste tecnologie e della scrittura di uno specifico smart contract per un dato diagramma di coregrafia, ovvero parte integrante dell'obiettivo di questa tesi, sarebbe interessante riuscire a realizzare un generatore automatico di smart contract per un qualsiasi diagramma di coreografia.
Come già enfatizzato, questo campo dell'informatica, ovvero quello delle blockchain e degli smart contract è ancora da esplorare, e non ho piena consapevolezza del carico

di lavoro che si cela dietro a questo proposito di sviluppo, ma credo che l'utilità e le applicazioni che ne conseguirebbero in vari settori, siano un buon incentivo per iniziare a lavorare in questa direzione.

Capitolo 8

Ringraziamenti

Ringrazio il Prof. Luca Spalazzi e l'Ing. Francesco Spegni, per avermi accompagnato in questo percorso di tesi.
Ringrazio tutti professori che ho avuto il piacere di conoscere, perché ognuno di loro mi ha lasciato qualcosa di positivo.
Nello specifico vorrei ringraziare il Prof. Matteo Franca, che pur non essendo un suo studente, ha avuto la pazienza e la disponibilità di ricevermi svariate volte e spiegarmi l'analisi matematica, con una chiarezza e una semplicità, tipica di chi padroneggia bene una materia e dona con generosità la sua conoscenza.
Ringrazio anche la Prof.ssa Giuseppa Ribighini per la sua profonda umanità e disponibilità nei confronti degli studenti, la ricordo con affetto.
Ringrazio il Prof. Luigino Criante, per la sua disponibilità, la sua passione e la sua attenzione alla didattica, nello spiegare la fisica agli studenti. Mi ritengo molto fortunato ad aver seguito le sue lezioni, e ho apprezzato molto le parole che mi ha detto a fine esame, le tengo a mente tuttora.
Ringrazio tutti miei colleghi, ho imparato qualcosa da ognuno di loro, e in particolare ringrazio Gabriele.
Ringrazio Luca, per le nottate passate a parlare di filosofia, mi hanno arricchito molto.
Ringrazio anche tutte le persone conosciute ad Ancona e quelle che mi hanno sostenuto, in fondo sono state delle "compagne di viaggio".
Ringrazio D. Goleman e A. Robbins: i loro libri mi hanno ispirato positivamente in dei momenti critici.
Ringrazio "Gio", che mi ha fornito nuovi punti di vista con grande schiettezza, ovvero nel modo che preferisco.
Ringrazio Giacomina, è stata per me un punto di riferimento durante la mia carriera universitaria.
Ringrazio D. che mi ha permesso di vedere e sperimentare sulla mia pelle qualcosa di nuovo, ora credo che esista qualcosa che prima credevo impossibile.

Sono grato per tutti gli eventi che sono accaduti durante il mio percorso universitario, forse alcuni mi hanno rallentato, ma allo stesso tempo mi hanno permesso di maturare come uomo.
Ringrazio la mia famiglia, che inizialmente ha stimolato la mia determinazione, e poi mi ha sostenuto fin qui.
Infine ringrazio me stesso, sempre per essere arrivato fin qui.

Elenco delle figure

2.1 B140970324, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons 12
2.2 Ethereum, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>, via Wikimedia Commons 18

3.1 https://remix.ethereum.org/ 22
3.2 https://www.trufflesuite.com/ 23

4.1 Diagrammi. Fonte: Blockchain Based Choreographies: the Construction Industry Case Study L.Spalazzi F.Spegni A.Corneli B.Naticchia . 26

5.1 Diagrammi. Fonte: Blockchain Based Choreographies: the Construction Industry Case Study L.Spalazzi F.Spegni A.Corneli B.Naticchia . 29

Bibliografia

[1] https://www.attivitaconlestero.net/cenni-sullo-smart-contract/#_ftn1

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system (Bitcoin whitepaper). 31 Ottobre 2008.

[3] https://en.wikipedia.org/wiki/Nick_Szabo

[4] Nick Szabo. Smart Contracts: Building Blocks for Digital Markets.1996.

[5] https://www.blockchain4innovation.it/mercati/legal/smart-contract/

[6] https://it.wikipedia.org/wiki/Smart_contract

[7] https://blog.osservatori.net/it_it/smart-contract-in-blockchain

[8] https://www.ibm.com/it-it/topics/what-is-blockchain

[9] https://it.wikipedia.org/wiki/Blockchain

[10] https://bitecoin.it/2018/08/27/quale-la-differenza-tra-blockchain-e-database/13060/

[11] https://it.wikipedia.org/wiki/Ethereum

[12] https://dariopironi.com/it/solidity-la-guida-definitiva/

[13] https://remix.ethereum.org/

[14] https://www.trufflesuite.com/

[15] https://academy.bit2me.com/it/come-programmare-la-solidit%C3%A0/

[16] Blockchain Based Choreographies: the Construction Industry Case Study L.Spalazzi F.Spegni A.Corneli B.Naticchia

[17] Performance,ottimizzazioni e best-practice nei contratti Solidity di Ethereum C.Laneve M.Costantini