UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Blockchain negli archivi digitali: sviluppo e analisi di applicazioni decentralizzate

Relatore:
Dott. Andrea Marchetti

Presentata da:
Lukasz Szczygiel

Correlatori:
Dott. Angelo Mario Del Grosso
Dott.ssa Clara Bacciu

Sessione autunnale
Anno Accademico 2017/2018

Indice

Elenco delle figure

1. Introduzione

Le blockchain sono un paradigma tecnologico che consente l'implementazione di un insieme di concetti fondamentali per lo stato di Internet attuale: la trasparenza, la sicurezza, l'immutabilità e la decentralizzazione. Dall'unione di questi concetti segue la nascita di un sistema di comunicazione, di gestione delle operazioni e di dati in rete senza precedenti. Il complesso di elementi costituenti il paradigma permette di creare e gestire basi di dati distribuite, formate da registri replicati su tutti i nodi partecipanti alla rete. Molti sono i modi in cui è possibile utilizzare le blockchain, il campo delle valute digitali è stato il primo ambito applicativo, ma le potenzialità più promettenti e innovative derivano dal fatto che il sistema può essere esteso a un insieme arbitrario di entità rappresentabili mediante la digitalizzazione. Una base di partenza per introdurre e descrivere l'utilità delle blockchain può essere indotta dalla loro implementazione della tecnologia dei registri distribuiti[1] (Distributed Ledger Techology, di seguito DLT) che, per ragioni di chiarezza dell'esposizione, è legittimo considerare situata ad un livello di astrazione superiore al modello di riferimento in questione.

Strettamente correlate a DLT e centrali per questa tesi sono le nozioni di registro e transazione. Per secoli varie istituzioni tra cui banche e governi hanno utilizzato registri e sistemi contabili per tenere traccia di una moltitudine di operazioni, a partire da registrazioni di natura commerciale e contabile, ai passaggi di proprietà, alle relazioni tra privati e così via. Si parla essenzialmente di transazioni aventi per oggetto beni e servizi. Queste operazioni di scambio tra uno o più soggetti sono regolate principalmente da un corrispettivo monetario il cui valore è garantito da un'autorità centrale.

Un altro modo consiste nel registrare i movimenti a credito e a debito (dare e avere) in un opportuno registro, ad esempio in un libro mastro. Questo tipo di transazioni legate

[1] UK Government Chief Scientific Adviser. *Distributed Ledger Technology: beyond block chain*. Government Office for Science, 2016. URL: `https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf`.

a un registro hanno la caratteristica intrinseca di dover essere mantenute da qualcuno, un intermediario. Indipendentemente che sia un'organizzazione privata o un'autorità centrale, sorge il problema della fiducia necessaria verso chi lo mantiene. Ecco che alla base di queste considerazioni iniziali, di registro e transazione, si introduce la tecnologia delle catene di blocchi (di seguito blockchain) che consente l'implementazione di un registro distribuito di transazioni che non necessita di un'autorità centrale fidata. Come espresso in una delle definizioni: "Le blockchain rappresentano una modalità particolarmente trasparente e decentralizzata per la registrazione di elenchi di transazioni"[2].

L'obiettivo di questo lavoro è di presentare in maniera sistematica le caratteristiche principali e lo stato dell'arte del paradigma, in modo da poter valutare in maniera consapevole se e quando scegliere questa tecnologia con i rispettivi benefici e complicazioni per quanto riguarda lo sviluppo di applicazioni decentralizzate. In rete è possibile trovare molti articoli e spiegazioni che descrivono le blockchain in maniera più o meno completa. Qui l'attenzione è posta su alcuni argomenti ritenuti più rilevanti per la programmazione in un ecosistema distribuito. Per inquadrare meglio tutte le caratteristiche del progetto finale si è scelto di presentare il procedimento a partire da concetti alla base della tecnologia fino alla creazione di un'applicazione reale sulla blockchain.

La struttura di questo lavoro rispecchia una divisione a tre strati[3]: il primo strato contiene i concetti, l'insieme di caratteristiche alla base della tecnologia. Il secondo strato si riferisce alle implementazioni, cioè a come questi concetti sono implementati nella rete, ad esempio Bitcoin, Ethereum. Nella terza fase abbiamo le istanze di una particolare implementazione della blockchain, in questo caso lo sviluppo e pubblicazione dell'applicazione con Ethereum nella blockchain di prova, Ropsten. Riassumendo, il lavoro è stato svolto seguendo questi passi,

- Concetti principali della blockchain presentati nel capitolo 2: Blockchain: il funzionamento e i concetti principali

[2]Philip Boucher, Susana Nascimento, Mihalis Kritikos. ≪How blockchain technology could change our lives≫. In: (2018). DOI: 10.2861/926645. URL: https://publications.europa.eu/en/publication-detail/-/publication/9964fbfd-6141-11e7-8dc1-01aa75ed71a1.

[3]Una divisione analoga a quella presentata durante la conferenza Devoxx in Belgio, tenuta da Sebastien Arbogast e Said Eloudrhiri, http://chainskills.com/2016/11/21/what-qualifies-as-a-blockchain/

- Implementazioni nel capitolo 3: Ethereum: Sviluppo di applicazioni decentralizzate
- Istanza dell'applicazione sviluppata nel capitolo 4: Progetto di Archivi Digitali

Nella prima parte di questa tesi viene approfondita l'innovazione portata dalla tecnologia blockchain con le caratteristiche e i vantaggi che essa porta con sé. Viene analizzato l'ecosistema decentralizzato, applicato concretamente per la prima volta e su larga scala nel sistema Bitcoin. A partire da questa base reale si procede astraendo alcuni fattori costitutivi che permettono di contrastare problemi derivanti dalla centralizzazione grazie alla risoluzione del problema dei generali bizantini attraverso un sistema di consenso distribuito. Di seguito si procede estendendo il concetto di transazione in modo che possa rappresentare ogni sorta di bene rappresentabile, concludendo con alcuni esempi reali di applicazioni.

Nella seconda parte si procede sistematicamente con l'ampliamento del concetto delle transazioni, in quanto atte a rappresentare non solo semplici scambi di valuta ma qualunque oggetto della vita reale. Lo sviluppo di programmi memorizzati sulla blockchain stessa, chiamati contratti (smart contracts) permette che le sue funzioni vengano eseguite automaticamente nelle transazioni. In questa sezione verrà analizzato il processo di sviluppo delle applicazioni decentralizzate con la blockchain Ethereum.

Infine, il progetto sugli archivi digitali. Si tratta di un'applicazione costruita sulla piattaforma Ethereum che permette la catalogazione di opere d'arte minori in maniera distribuita. Questa scelta di registrare opere definibili come minori è dovuta al possibile impiego reale di tale tecnologia, ma è applicabile a qualsiasi tipo di opere d'arte e in generale di beni culturali. La documentazione mette insieme il processo di sviluppo illustrato nelle sezioni precedenti per costruire un'applicazione decentralizzata pubblicata sulla blockchain Ethereum.

2. Blockchain: il funzionamento e i concetti principali

Seguendo l'impostazione del lavoro presentata nell'introduzione è possibile individuare una prima nozione ricorrente, quella di "*registro distribuito*". Negli ultimi anni l'estensione di Internet ha permesso di muovere i registri e, generalmente parlando, le informazioni e i dati, da un supporto fisico a un supporto digitale attraverso l'utilizzo delle basi di dati sparse in tutto il mondo. Tuttavia, la struttura attuale delle basi di dati e, per certi aspetti, l'architettura odierna di Internet, presentano dei problemi e degli svantaggi che, con l'andare del tempo, non sono stati risolti in maniera definitiva. Basti pensare ai problemi relativi alla sicurezza, affidabilità e centralizzazione delle risorse memorizzate in rete.

Problematiche che, nella maggior parte dei contesti, possono essere ricondotte alla commercializzazione e centralizzazione delle risorse. Il punto chiave è che Internet allo stato attuale, Internet delle informazioni, si avvale di intermediari. Aziende come Google, Microsoft, Amazon e Facebook controllano una quantità enorme di informazioni e attività dei propri utenti. Un esempio del fenomeno può essere indotto dall'offerta di servizi di cloud computing (*aaS, anything as a service[1]). Senza dubbio il vantaggio di questi servizi è che permettono agli utenti di usare risorse senza dover investire in un'infrastruttura propria e che sono offerti ad un costo relativamente basso. Si tratta pur sempre di intermediari verso quali è necessaria la fiducia che, con il passare del tempo, le corporazioni non sempre hanno dimostrato di gestire in maniera soddisfacente. Sono sorte preoccupazioni circa questioni importanti come la sicurezza, la censura e l'affidabilità di questi sistemi. Più in generale, i sintomi negativi della centralizzazione sono evidenti quando si parla dei problemi relativi alla privacy e all'uso non autorizzato di dati privati, temi che ultimamente emergono di frequente. Ecco che avere opzioni, alternative in più a quelle già esistenti, può essere importante al fine di mitigare i rischi che possono essere causati

[1] *Cloud Computing - Servizi*. URL: https://en.wikipedia.org/wiki/Cloud_computing#Service_models. (Visitato: 10.11.2018).

da una situazione di monopolio e di concentrazione del potere nelle mani dei pochi.

La blockchain è un paradigma, oppure, semplificando, una tecnologia che potrebbe portare a una nuova evoluzione, un potenziale passaggio da un Internet delle informazioni all'Internet dei valori. Grazie all'uso della crittografia, del sistema di firme digitali, del networking peer-to-peer (P2P) e algoritmi di consenso, gli utenti possono effettuare transazioni che verranno mantenute in registri distribuiti, immutabili e permanenti senza bisogno di intermediari. In questo capitolo verranno descritti i concetti che formano l'insieme delle componenti di base delle blockchain.

2.1 La struttura delle blockchain

I concetti relativi alla struttura possono essere derivati dalla sua prima implementazione. Nel 2008 nasce Bitcoin, la prima applicazione su larga scala della tecnologia di seguito denominata blockchain. Fondata da Satoshi Nakamoto (autore o gruppo di autori anonimo), aveva come finalità la creazione di un sistema monetario digitale[2]. Il sistema utilizza un registro pubblico, distribuito su tutti i nodi, contenente tutte le transazioni ordinate e valide, eseguite al suo interno fin dall'inizio della creazione del sistema stesso. Il concetto di sistema monetario digitale fu descritto già nel 1998 da Nick Szabo[3], il quale aveva coniato il termine di certificati digitali al portatore (digital bearer certitifcate). Ancora prima, nel 1991, nell'articolo "How to Time-Stamp a Digital Document" di Stuart Haber e W. Scott Stornetta[4] furono descritte quelle che sono le basi teoriche di un sistema decentralizzato. Questi documenti rappresentano una fonte rilevante di approfondimento e sono importanti in quanto contengono l'espressione di concetti giudicati utili per la comunità ai fini di migliorare i sistemi digitali esistenti. Furono alla base della successiva ricerca sulla

[2]Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf.* 2008.

[3]Nick Szabo. ≪Contracts with Bearer≫. In: (1997). URL: `https://nakamotoinstitute.org/contracts-with-bearer/`.

[4]Stuart Haber e W. Scott Stornetta. ≪How to Time-stamp a Digital Document≫. In: *Journal of Cryptology* 3 (1991), pp. 99–111.

tecnologia che ha condotto, a partire da Bitcoin, alla possibilità della sua implementazione concreta.

Al momento esistono oppure sono in via di sviluppo molte applicazioni e sistemi che utilizzano le blockchain. Sovente la loro struttura costituisce un'estensione o un adattamento più sofisticato dei concetti iniziali a seconda delle funzionalità offerte. Questa è la motivazione per cui saranno presentati principalmente i concetti universali anche in funzione degli obiettivi finali per fornire un collegamento alle loro implementazioni e infine al caso concreto dell'archivio digitale.

2.1.1 Distribuzione e decentralizzazione

Prima di passare ai dettagli strutturali delle catene dei blocchi è importante distinguere tra due nozioni, quella di *sistema distribuito* e *decentralizzato*. Spesso, quando si parla di blockchain, i due termini sono usati come sinonimi.

In realtà, dalle considerazioni precedenti è emerso che le blockchain implementano la tecnologia dei registri distribuiti collocando quest'ultima ad un livello di astrazione superiore. La DLT consiste nella condivisione di registri oppure, detto in un altro modo, base di dati a tutti i partecipanti del sistema. In questo caso i sistemi in questione sono le blockchain che, a loro volta vengono gestite da regole precise incapsulate nei sistemi di consenso (un argomento che verrà affrontato in dettaglio nel paragrafo 2.2). A questo punto è sufficiente dire che, grazie a questi sistemi di consenso, ogni nodo partecipante al sistema possiede una copia del registro e può interagire con esso effettuando modifiche che successivamente vengono aggregate con quelle di altri partecipanti. L'obiettivo è quello di mantenere una versione condivisa e sincronizzata del registro a tutti i partecipanti del sistema, in funzione delle regole di consenso della rete blockchain. Riassumendo si ha una completa distribuzione quando tutti i partecipanti sono in possesso di uno stesso registro.

Per quanto riguarda la nozione di decentralizzazione un approfondimento rilevante sulla questione è fornito in un articolo di Vitalik Buterin, uno dei fondatori del progetto Ethereum[5]. La sua idea è basata sulla suddivisione di un sistema e la sua centralizzazione

[5] *Ethereum Blockchain App Platform.* URL: `https://www.ethereum.org/`.

o decentralizzazione lungo tre assi: quelli dell'architettura, della politica e della logica.

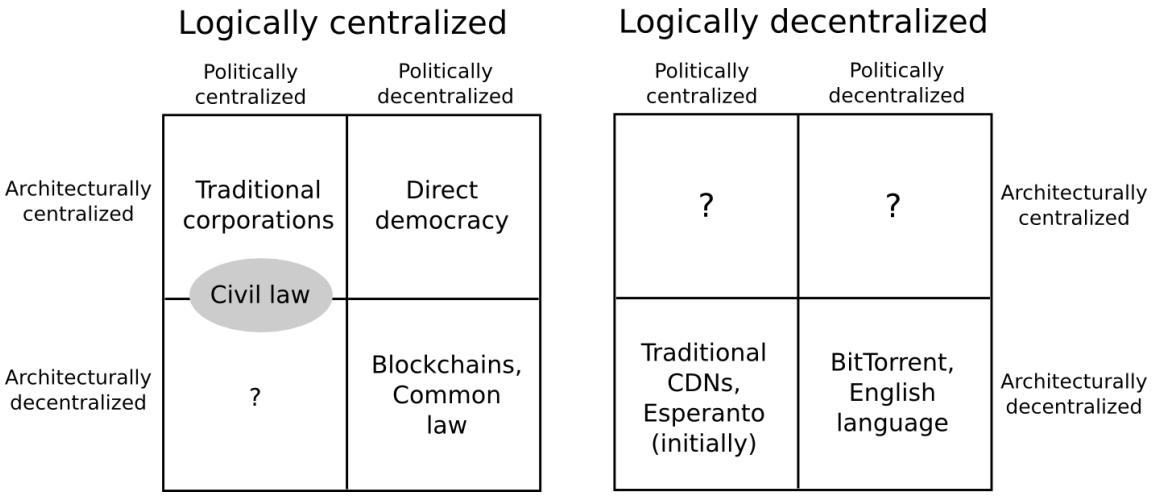

Figura 2.1: Gli assi della decentralizzazione

Secondo l'autore, come illustrato nella figura 2.1, le blockchain sono architetturalmente e politicamente decentralizzate (nessuno esegue un controllo diretto su di esse e non c'è un punto centrale di guasto) ma logicamente centralizzate in quanto sono soggette a un preciso stato condiviso al momento dell'esecuzione del sistema che si comporta come un singolo computer[6]. Quest'ultimo aspetto, di nuovo, è fortemente legato ai protocolli di consenso che dipendono principalmente dalle varie tipologie di blockchain (un aspetto discusso nel paragrafo 2.4). La decentralizzazione riguarda maggiormente l'aspetto dei permessi dei partecipanti e la *governance* sulla rete. La suddivisione descritta è valida per blockchain pubbliche (*permissionless*) sulle quali si concentra questa tesi, in cui tutti i partecipanti hanno pari diritti di agire e modificare il registro condiviso. Per quanto riguarda la tipologia con permessi (*permissioned*) la questione va riesaminata caso per caso, a seconda dell'implementazione concreta.

Uno tra tanti esempi di quest'ultima tipologia è Ripple[7], blockchain per trasferimento di fondi che nel suo sistema di validazione si avvale di un certo numero di attori imposti a

[6]Vitalik Buterin. *The Meaning of Decentralization*. URL: `https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274`. (Visitato: 20.08.2018).

[7]*Ripple: Blockchain*. URL: `https://ripple.com/`.

priori perché ritenuti fidati dall'azienda. Questa politica comporta dei vantaggi dal punto di vista della scalabilità e della velocità di funzionamento a discapito della decentralizzazione dovuta alle forme di mediazione centralizzata. Attualmente, in questo specifico caso, con l'evoluzione della tecnologia blockchain questa strategia[8] viene rivista a favore di una maggiore decentralizzazione. Gli aspetti pro e contro l'adozione di sistemi più o meno decentralizzati sono discussi nel paragrafo (2.3.3).

Finalmente, distinte queste due nozioni è possibile arrivare alla seguente conclusione: il paradigma blockchain è distribuito (i partecipanti sono in possesso del registro replicato) mentre a livello inferiore, delle implementazioni, le blockchain possono essere più o meno decentralizzate (o centralizzate) a seconda degli obiettivi e delle funzionalità che intendono raggiungere.

2.1.2 Transazioni e concatenazione di blocchi

Le blockchain utilizzano, letteralmente, una struttura pensata come una catena di blocchi. Ciascun blocco contiene al suo interno una lista di transazioni immutabili a loro volta contrassegnate da delle proprietà. Nelle attuali implementazioni le proprietà fondamentali sono:

- indirizzo mittente (input)
- indirizzo destinatario (output)
- ammontare di valuta o token contenuto nella transazione
- marcatura temporale (timestamp)

In una rete di consenso decentralizzato vengono man mano inseriti dei blocchi contenenti al loro interno un certo numero di transazioni. A livello universale, di rete, questa catena viene gestita dagli algoritmi di consenso come ad esempio: Proof of Work e Proof of Stake (descritti nel paragrafo 2.1.4) appendendo i blocchi giudicati validi dopo l'ultimo

[8] *Ripple: Strategia di decentralizzazione*. URL: `https://ripple.com/dev-blog/decentralization-strategy-update/`.

blocco valido della catena in modo da creare un registro di operazioni ordinato cronologicamente. Una volta stabilita la legittimità di un blocco e la sua appartenenza alla catena valida più lunga, l'intera struttura viene aggiornata propagando (analogamente a come avviene nel modello peer-to-peer) i nuovi elementi a tutti i partecipanti del sistema. Lo stato del registro, a questo punto, non è più soggetto a modifiche, diventa permanente e immutabile. Successivamente il procedimento riparte ricorsivamente, come descritto, ampliando la catena con le nuove operazioni.

Il primo blocco[9] [10] (programmato manualmente cioè *hardcoded*), originario di tutta la catena è chiamato *genesi*. È un blocco speciale perché non è preceduto da altri blocchi, in esso vengono programmate le proprietà dei blocchi futuri. Le specifiche cambiano a seconda dell'implementazione. Generalmente si tende a definire proprietà come la grandezza del blocco, la difficoltà di appendere nuovi blocchi e la ricompensa per la creazione (validazione) di nuovi blocchi nel sistema. Seguendo questo ragionamento, il seguente è un esempio valido di un blocco generico che eredita le sue proprietà dal blocco genesi. Di seguito, insieme alle spiegazioni di alcuni concetti si forniscono pezzi di codice (code snippets) per illustrare la relativa struttura semplificata (il codice rispetta la sintassi Javascript ECMAScript 2015[11]).

```
1 class block {
2     constructor(timestamp, transactions, prevHash ='') {
3         this.timestamp = timestamp;
4         this.transactions = transactions;
5         this.prevHash = prevHash;
6         this.hash = this.calculateHash();
7         this.nonce = 0;
8     }
```

Codice 2.1: Esempio di struttura di un blocco

[9] *Bitcoin, blocco genesis*. URL: https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f.

[10] *Ethereum, blocco genesis*. URL: https://etherscan.io/block/0f.

[11] *ECMAScript® 2015 Language Specification*. URL: http://www.ecma-international.org/ecma-262/6.0/index.html. (Visitato: 10.11.2018).

Ancora una volta, tra le proprietà importanti per il protocollo, si possono distinguere: la marcatura temporale che corrisponde alla data di creazione del blocco, la lista di transazioni incluse nel blocco, e la connessione tra i blocchi ovvero il ”legame” calcolato tramite la funzione hash tra il blocco successivo e il precedente.

Combinando insieme i blocchi si otterrà la seguente panoramica della struttura:

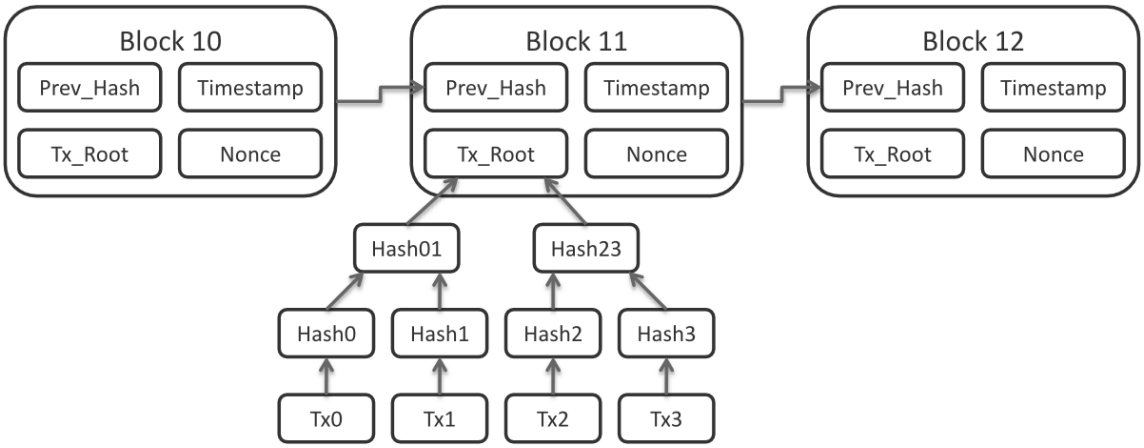

Figura 2.2: Struttura semplificata di una catena di blocchi

Tenendo in mente che l’impostazione di un blocco può contenere un numero di proprietà variabile (a seconda dell’implementazione), è possibile generalizzare una struttura come quella illustrata nella figura 2.2. Nelle sezioni successive saranno analizzati gli elementi costituenti dei blocchi e le procedure che ne determinano il funzionamento.

2.1.3 Funzioni di hash

Ogni blocco è legato al blocco precedente tramite una funzione hash calcolata in base alle proprietà del blocco. Una funzione hash è una funzione matematica unidirezionale che accetta un input di lunghezza arbitraria e ne produce un output di lunghezza prefissata.

È una delle componenti base di una struttura formata da sequenze di blocchi che a sua volta è paragonabile alla struttura di una lista concatenata (*linked list*) tramite puntatori. In entrambi i casi la struttura permette di identificare in maniera univoca la relazione precedente e successivo che intercorre tra due blocchi. Dunque, data la definizione della funzione hash, il suo risultato dipende dalle proprietà di ciascun blocco.

Senza entrare troppo in dettaglio, questo implica un insieme di proprietà ”grande” su quale vengono calcolate le funzioni hash con la caratteristica che, anche una minima variazione di ciascun componente del dominio della funzione, molto probabilmente farà cambiare radicalmente il risultato calcolato. Nel seguente esempio si calcola la funzione hash applicando l'algoritmo SHA256 (Secure Hashing Algorithm[12]), utilizzata nelle blockchain (per esempio Bitcoin) per due input apparentemente simili:

```
1 //Input 1: ciao Tizio
2 >> SHA256("ciao Tizio")
3 //Output 1:
4 >> 875647588afa538ef3645bcdf413497af42461ea09e8f79df69b7a95c229d2a5
5 //Input 2: ciao tizio
6 >> SHA256("ciao tizio")
7 //Output 2:
8 >> 961e89e74136c7094097f1a625b596f5df27cc6a71e0031bacf609e09feb2634
```

Codice 2.2: Esempio di calcolo della funzione hash SHA256

Si vede che anche la più piccola variazione in input produce (con probabilità estremamente alta) un output (righe 4,8) molto diverso.[13] Continuando con questo ragionamento, i blocchi sono formati da un insieme di proprietà che messe insieme produrranno una hash unica. Considerando il seguente esempio di calcolo della proprietà prevHash di un blocco:

```
1 >> SHA256(prevHash+timestamp+transactions+nonce)
```

Codice 2.3: Esempio di calcolo della funzione hash in base alle proprietà del blocco

È chiaro, a questo punto cosa si intende per quello che all'inizio di questa sezione veniva chiamato: ”un'insieme di proprietà grande”. Con più precisione, in qualsiasi blocco, il dominio della funzione hash è formato da (sempre in riferimento all'esempio della figura 2.2): marcatura temporale, il risultato delle continue applicazioni di hashing proprie di ciascuna transazione (tx root), il numero pseudocasuale (nonce) e il risultato della funzione hash del blocco precedente. In breve, vengono messe insieme tutte queste infor-

[12] *SHA2 - Funzioni di Hash*. URL: https://en.wikipedia.org/wiki/SHA-2. (Visitato: 10.11.2018).

[13] *Esempio del calcolo di SHA256 - Generatore Hash*. URL: https://passwordsgenerator.net/sha256-hash-generator/.

mazioni (paragonabile a una “polpetta” fatta da un composto di altri “elementi”) contenute nel blocco per produrre un unico hash. Questo non è l’unico caso in cui l’utilizzo di questa funzione è fondamentale, lo hashing delle transazioni è stato menzionato per introdurre un livello di dettaglio ulteriore. Nella prossima sezione, a proposito delle continue applicazioni delle funzioni hash, verrà introdotta la struttura, Merkle Tree (albero di hash).

Infine, disponendo della vista d’insieme dell’algoritmo di hashing, ne deriva la sicurezza dell’intero sistema. Supponendo uno o più attacchi con l’obbiettivo di alterare il contenuto dei blocchi il malintenzionato dovrà ricalcolare tutti gli hash dei blocchi successivi per validare l’intera catena e dovrà farlo prima che un nuovo blocco arrivi validato dal sistema di consenso. In questo caso la priorità è della catena più lunga quindi l’eventuale attaccante sarà costretto a ricalcolare gli hash della nuova catena dall’inizio. Quello della sicurezza è un concetto che dipende dal sistema di consenso implementato e verrà discusso in seguito con maggior dettaglio.

2.1.4 Merkle Trees

Gli alberi di hash sono una struttura dati utilizzata nelle blockchain. È una delle componenti del blocco le cui proprietà fondamentali si sono ipotizzate negli esempi delle sezioni precedenti. In particolare, nella figura 2.2 è presente la radice (tx root) di un merkle tree. La radice è il risultato di applicazioni continue di operazioni hash a partire dal livello con cardinalità più alta (ipotizzando che la cardinalità della radice sia minore di quella dei suoi figli). Il motivo per cui viene usata questa struttura è da ricondurre all’efficienza degli alberi di hash, nel memorizzare, nel caso delle blockchain, le transazioni.

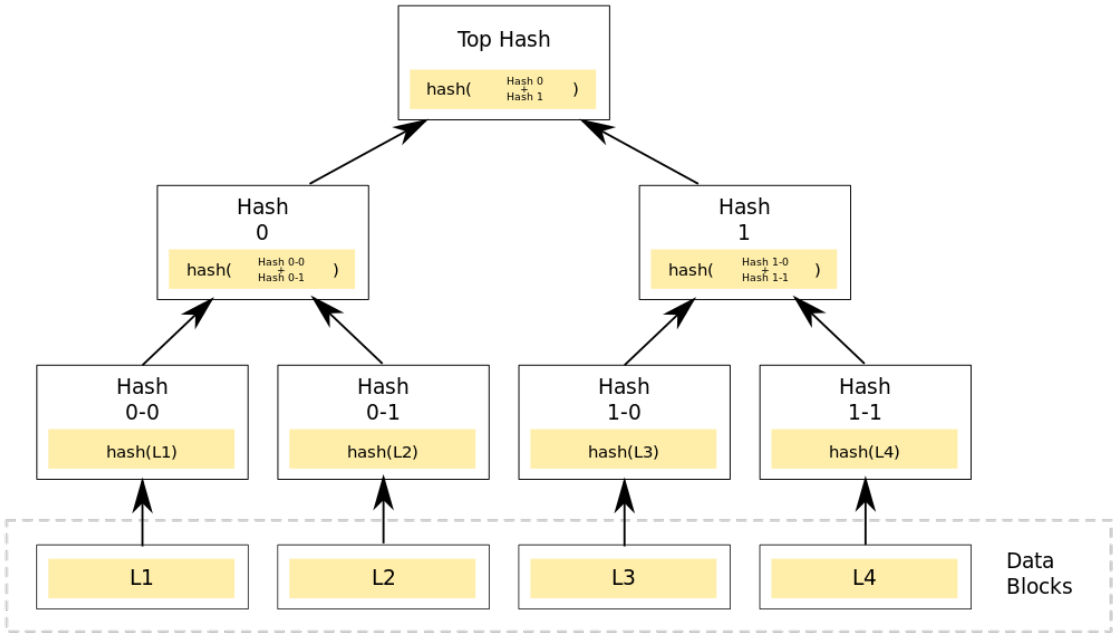

Figura 2.3: Struttura di Merkle Tree

La figura 2.3 rappresenta un albero di hash binario che può essere generalizzato per visualizzare il procedimento di hashing di un numero arbitrario di transazioni (limitate nella definizione del blocco). Nell'esempio a partire dal basso (o a livello di cardinalità più alto, nelle foglie) si trova contenuto l'insieme delle singole transazioni (L1-L4).

Il primo passo consiste nell'eseguire l'algoritmo di hashing sulle singole transazioni. L'output di queste operazioni viene successivamente accoppiato con il risultato dell'elemento (nodo) adiacente e su di essi viene di nuovo eseguito l'algoritmo di hashing. Il procedimento è ripetuto ricorsivamente fino a giungere alla radice, risultato di continue applicazioni di hashing nell'insieme di nodi, i quali si trovano a sinistra e a destra rispetto alla radice. Dunque, ogni nodo è l'hash dei suoi figli[14].

Una delle proprietà delle funzioni crittografiche di hash (come SHA-2) consiste, dal punto di vista computazionale nell'essere veloce da calcolare su qualunque tipo di dato. Inoltre la peculiarità di questa struttura è che appendere nuove foglie, equivalenti a nuove

[14] *Whitepaper Ethereum - Merkle Trees*. URL: `https://github.com/ethereum/wiki/wiki/White-Paper#merkle-trees`.

transazioni che vengono man mano aggiunte all'albero, non comporta la ricomputazione dell'intera struttura, ma solo del percorso che porta direttamente alla radice.

Il vantaggio dal punto di vista dell'efficienza dei merkle trees deriva proprio da questa impostazione della struttura gerarchica definita dagli elementi sottostanti. Si tratta dell'efficienza in termini di tempo e di spazio necessario per memorizzare la struttura. La finalità di Merkle Trees deriva dalla frammentazione dei dati al suo interno e la loro composizione nella radice, sotto un unico identificatore hash. In questo modo, per assicurare l'integrità delle transazioni del blocco, sarà sufficiente verificare la radice della struttura (tx root). Questo è possibile perché la propagazione con la messa insieme di funzioni hash avviene sequenzialmente dalle foglie alla radice.

A livello della blockchain questo apre diverse possibilità di verifica dell'integrità delle singole transazioni e la possibilità di ridurre lo spazio e il tempo necessario per sincronizzare la catena dei blocchi aggiornata all'ultima versione valida. Ad esempio, per un membro della rete che non intende effettuare operazioni su singole transazioni dei blocchi sarà sufficiente verificare solo gli hash contenuti nell'intestazione (header) dei blocchi per avere una prova sufficiente della correttezza dei dati all'interno della struttura. Questa caratteristica di Merkle Tree prende il nome di *Merkle proof* (prova di correttezza) e vale per qualsiasi elemento della struttura. Dunque supponendo che questa struttura venga usata per memorizzare una grande quantità di dati, controllare che un frammento di questi dati sia consistente, verrà controllato il percorso che collega il dato elemento alla radice. Questo equivale a rieseguire calcoli delle funzioni hash su specifici frammenti di dati e controllare i risultati con quelli già calcolati da altri nodi della rete per confermare la loro correttezza e quindi la validità per la blockchain.

2.2 Il sistema di consenso

Una volta definito il funzionamento concettuale di una rete blockchain sorge il dubbio come questo proliferare di operazioni, di calcolo delle funzioni hash, dell'addizione dei nuovi blocchi alla catena e più in generale l'insieme delle operazioni di gestione dell'intera struttura siano possibili in un ecosistema decentralizzato.

Innanzitutto, le blockchain decentralizzate operano in maniera incentivata. La disintermediazione è garantita dagli algoritmi di consenso che stabiliscono le regole di funzionamento della rete. Nel caso delle blockchain pubbliche si tratta di una rete dove tutti i partecipanti sono alla pari, possono proporre nuove transazioni e tutti insieme devono mettersi d'accordo su quali transazioni effettivamente aggiungere al registro comune. Si vede delineato un contesto in cui è presente un insieme di partecipanti, ciascuno avente i propri interessi ma nonostante obiettivi spesso contrapposti l'intero sistema deve giungere a un consenso. Tipicamente l'incentivazione consiste in una ricompensa, una quantità di valuta o token, spettante al nodo o ai nodi a cui viene dato il potere di appendere un blocco alla catena. L'approfondimento di questo aspetto economico è strettamente legato al tipo di algoritmo di consenso implementato, alcuni dei quali (ad esempio, Proof of Work e Proof of Stake), verranno discussi più avanti in questa sezione.

2.2.1 Il problema dei generali bizantini

Dal punto di vista teorico, la difficoltà di consenso può essere ricondotta al problema dei generali bizantini[15]. In una delle possibili formulazioni del problema si suppone che una città venga assediata da molti eserciti con a capo un generale. Per ottenere la vittoria tutti gli eserciti devono raggiungere un consenso e attaccare simultaneamente altrimenti, se non attaccano in numero sufficiente, verranno sconfitti. Il generale manda l'ordine attraverso dei messaggeri ai comandanti degli eserciti che, a loro volta possono mentire scegliendo arbitrariamente di passare ordini diversi e di tradire il generale e gli altri comandanti.

Dunque è una situazione che si porta bene a essere visualizzata all'interno delle blockchain (propriamente decentralizzate). In riferimento al problema sopra esposto si fa a meno del generale che manda degli ordini ai suoi sottoposti. Nella pratica, si tratta di una rete dove tutti i partecipanti possono proporre nuove transazioni e tutti insieme devono mettersi d'accordo su quali transazioni effettivamente aggiungere al registro comune in mancanza di un nodo generalmente definibile come detentore della verità (*holder of truth*).

[15]Leslie Lamport, Robert Shostak e Marshall Pease. ≪The Byzantine Generals Problem≫. In: *ACM Transactions on Programming Languages and Systems* 4/3 (1982), pp. 382–401. URL: `https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/`.

Il punto chiave è che tutti i componenti del sistema devono prendere la stessa decisione (e non necessariamente la decisione ritenuta "giusta") su un piano di parità, cioè senza l'esistenza di un'entità "fidata" che possa provarne l'autenticità. Tutto questo in un contesto di presenza di potenziali nodi "traditori" che minacciano l'integrità della rete. In riferimento a questi nodi viene sollevata la questione della fiducia che deriva dall'implementazione delle regole di consenso. La struttura delle blockchain (l'aggiunta di nuovi blocchi quindi la validazione delle transazioni) è controllata da queste regole le cui implementazioni implicano anche l'esistenza di un limite che se superato permette di approvare operazioni illecite e potenzialmente corrompere l'intera rete. Questo limite, a seconda dell'algoritmo utilizzato, riguarda la potenza computazionale necessaria per superare il controllo di validazione e il numero di nodi che decidono di mettersi d'accordo per approvare operazioni al di fuori dalle regole del sistema. Un esempio concreto, nella rete Bitcoin, la potenza computazionale necessaria per compromettere la rete è maggiore del 50% totale della capacità computazionale dell'intera rete[16]. Questa soglia cresce man mano che i potenziali attaccanti vorrebbero modificare le transazioni contenute nei blocchi precedenti[17].

In questo modo, grazie al consenso distribuito, le blockchain riescono a rimediare al problema dei generali bizantini (BFT, *Byzantine Fault Tolerance*) in un ecosistema distribuito. Questo, come già accennato, è garantito dagli algoritmi di consenso implementati nella rete, di cui i principali Proof of Work e Proof of Stake. Nell'insieme BFT contribuisce alla sicurezza della rete e previene il fenomeno della doppia spesa[18].

2.2.2 Proof of Work

Proof of Work, cronologicamente, è il primo sistema di consenso usato nelle blockchain implementato per la prima volta da Bitcoin con lo scopo di raggiungere un consenso tra in nodi presenti sulla rete circa lo stato dell'intero sistema. La sua finalità consiste nella

[16] *Bitcoin: Attacco della maggiornaza (majority attack).* URL: `https://en.bitcoin.it/wiki/Majority_attack`. (Visitato: 10.11.2018).

[17] *Bitcoin: Tentativi di modifica della storia della rete.* URL: `https://en.bitcoin.it/wiki/Weaknesses#Attacker_has_a_lot_of_computing_power`. (Visitato: 10.11.2018).

[18] *Bitcoin: Attacchi e prevenzione della doppia spesa.* URL: `https://en.bitcoin.it/wiki/Irreversible_Transactions`. (Visitato: 10.11.2018).

gestione e nell'aggiornamento della catena dei blocchi con i relativi elementi in modo che i tutti i partecipanti siano "in linea" con la versione della catena valida cioè con la sequenza dei blocchi più lunga. Il legame tra i blocchi, una volta validato dall'algoritmo comporta la creazione di una struttura di contenuti immutabile comprendente tutta la storia delle operazioni intercorse fin dalla creazione del blocco genesi.

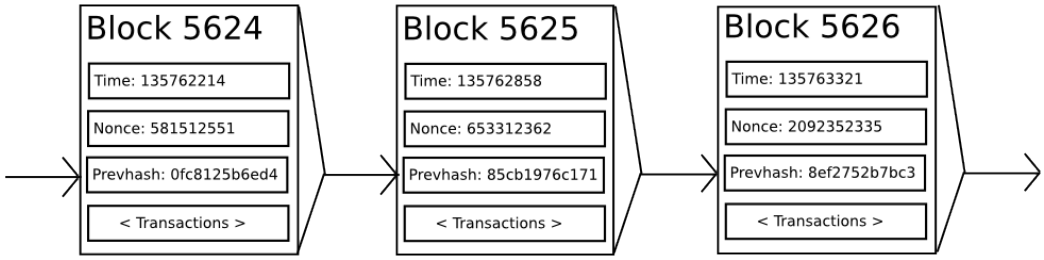

Figura 2.4: Mining: Proof of Work

Nella figura 2.4 è illustrato il processo di creazione di nuovi blocchi collegati mediante il riferimento all'indirizzo hash del blocco precedente. Da notare: il tempo di creazione del blocco strettamente crescente dovuto al funzionamento di Proof of Work che sequenzialmente, a intervalli regolari, permette l'inserimento nella rete di un blocco valido. Messe insieme: la quantità di computazione richiesta e la creazione di blocchi a intervalli regolari diventa possibile rimediare ai problemi di congestione della rete, dovuti alla creazione di una quantità di transazioni troppo grande da essere processata in tempi ragionevoli (così come avviene negli attacchi di tipo DDOS).

Senza un punto centrale di calcolo il procedimento di validazione è incentivato per permettere il funzionamento affidabile e costante in un sistema distribuito. Nel caso di Proof of Work, i membri della rete blockchain possono partecipare a una competizione con l'obbiettivo di appendere un blocco alla catena e ricevere in cambio una ricompensa. Si tratta di dimostrare di aver compiuto una quantità di calcoli computazionali per mezzo di cosiddetti "puzzle crittografici". Queste dimostrazioni possono, ad esempio sfruttare la caratteristica della distribuzione uniforme delle funzioni crittografiche di hash per cui il calcolo di una funzione hash comporta la generazione di un risultato, il quale, come dimo-

strazione, dovrebbe appartenere a un insieme ristretto del dominio dei possibili risultati. Relativamente alla crescita e all'incremento della potenza computazionale dei calcolatori è consigliabile che questa dimostrazione sia controllata da una difficoltà variabile, anch'essa crescente in relazione all'aumento di *hashrate* (la frequenza di calcolo della funzione hash). Ad esempio una dimostrazione Proof of Work potrebbe richiedere che un numero arbitrario di cifre del risultato generato dalla funzione hash sia uguale a zero. Successivamente l'incremento della difficoltà consiste nell'incremento di questo numero di cifre. Di conseguenza, l'aumento di *hashrate* è direttamente proporzionale alla probabilità di venire selezionati come creatori del nuovo blocco e quindi ricevere la ricompensa.

Per ribadire quanto è stato appena detto si suppone che la dimostrazione richieda che le prime tre cifre della funzione hash calcolata siano pari a zero. A questo scopo si riprende l'esempio del codice 2.3: Esempio di calcolo della funzione hash, in base alle proprietà del blocco.

```
1  function calculateHash(difficulty) {
2      nonce = nonce + 1;
3      return SHA256(prevHash+timestamp+transactions+nonce);
4  }
5  ...
6  //Output per nonce = 10:
7  3747cd1b31b876bda0bf8540ad4eeb115239c2d69835a74cc5835d9d3b3f3b27
8  //Output per nonce = 11:
9  aac295677f4761d411bc4fdb2953ac3ce2a2b8202207928a8db1cdac45aa8885
10 ...
11 //Output per nonce = 10390:
12 0000b280f48a4640ea18e2b6ee999fc60bfc85b690e990c0bdc4f9ae2b1e13c6
```

Codice 2.4: Esempio di calcolo della funzione hash in base alle proprietà del blocco

A ogni iterazione viene generato il risultato della riga 3 e successivamente confrontato con la difficoltà che in questo esempio supponiamo sia pari a quattro zeri iniziali. Concretamente il numero di questi tentativi fatti in un secondo prende il nome di *hashrate*. Se il risultato non soddisfa la dimostrazione allora viene incrementata di uno la proprietà

nonce, numero arbitrario che può essere usato una volta sola. Siccome anche una minima variazione in input probabilmente farà cambiare radicalmente il risultato, ad un certo punto verrà generato un risultato che soddisfa i requisiti prestabiliti nella proprietà della difficoltà. Nella simulazione effettuata utilizzando una versione estesa di calculateHash l'output della riga 12 è stato trovato, dopo pochi secondi, al 10.390 tentativo. Per ottenere il risultato con cinque cifre iniziali uguali a zero ci è voluto circa un minuto e 426.203 tentativi.

```
nonce 426201 hash 51cee854f7224aaa54266e21fd9ccdaf34e671de53b4c84f5b6373b00493b6e7
nonce 426202 hash 45e492b2e955045f5e943211df4305d2ff2689ff2cb9f2653a97c0670c633302
nonce 426203 hash 0000027efc4c1fa4966cb72a08bd039b2402c171bccf719d188ac428cb72bf34
Block mined: 0000027efc4c1fa4966cb72a08bd039b2402c171bccf719d188ac428cb72bf34
```

Figura 2.5: Mining: Risultato simulazione

In conseguenza, nelle applicazioni reali, succede che questa dimostrazione di calcoli computazionali porta con sé il consumo di una quantità enorme di energia elettrica. Evidentemente questo può essere considerato come uno spreco con un impatto negativo nei vari contesti generali come l'ambiente, il favoreggiamento di utenti con infrastruttura più efficiente (ad esempio, *mining pools*[19]) ecc. Si ipotizza che, attualmente, il consumo di energia annuo di Bitcoin sia di circa 2.55 GW, paragonabile al consumo annuo dei paesi come l'Irlanda (circa 3.1 GW) e in questa forma è destinato a crescere[20].

2.2.3 Proof of Stake

Per contrastare le conseguenze negative derivanti dall'utilizzo di Proof of Work sono stati sviluppati oppure sono in produzione altri sistemi di consenso. Il perno della costruzione di questi sistemi riguarda proprio la fase di dimostrazione che contribuisce alla sicurezza

[19]Le mining pool ragruppano utenti chiamati miner che, decidono di condividere la propria potenza computazionale per collaborare al processo di creazione dei blocchi. In questo modo la ricompensa sarà equamente distribuita a tutti i partecipanti di un certo gruppo (pool) di miner

[20]Alex de Vries. ≪Bitcoin's Growing Energy Problem≫. In: (5.2018). DOI: https://doi.org/10.1016/j.joule.2018.04.0165. URL: https://www.cell.com/joule/fulltext/S2542-4351(18)30177-6. (Visitato: 10.11.2018).

e integrità della rete, così come descritta in Proof of Work. Tra i sistemi più promettenti è stato definito Proof of Stake, il quale riesce a risolvere il problema di consumo incrementale di energia dovuto alla quantità di calcoli computazionali.

Questo algoritmo si basa su un processo di elezione. All'interno della rete blockchain, gli utenti possono scegliere di entrare a far parte di un gruppo di utenti, aperto a tutti, chiamati validatori. Per poter partecipare a questo processo è sufficiente depositare una quantità di valuta o token che verrà conservata nella rete come deposito (*stake*). Analogamente a Proof of Work a ogni partecipante, nel caso di Proof of Stake chiamato validatore al posto di miner, viene concesso il potere di appendere un blocco alla blockchain ricevendo in cambio una ricompensa (di solito una parte delle commissioni incluse nell'insieme delle transazioni del blocco). Il nodo validatore può essere scelto seguendo un processo di selezione casuale e tenendo conto della quantità del deposito messo a disposizione. I vantaggi di questo approccio sono evidenti dal punto di vista dell'efficienza computazionale. Inoltre, con la crescita della rete e la quantità di criptovaluta circolante al suo interno diventa teoricamente impraticabile effettuare un attacco della maggioranza[21] che, tra le altre cose può essere ulteriormente contrastato introducendo una certa variabile di casualità al processo di selezione del validatore. La scelta di particolari soluzioni dipende dalle diverse implementazioni dell'algoritmo, relativamente a Proof of Stake è opportuno menzionare che il deposito di validatori disonesti, cioè di coloro che cercano di approvare transazioni fraudolente, non sarà restituito e in tal caso questi perderebbero dei soldi. Lo stesso deposito sarà mantenuto dalla rete per un certo periodo di tempo necessario per provare che non siano state accettate transazioni fraudolente da parte del validatore.

2.3 Funzionamento dell'ecosistema blockchain

L'insieme dei concetti presentati, contribuisce al funzionamento delle blockchain in un contesto distribuito. In questo capitolo sarà approfondito il legame tra queste componenti per capire meglio il funzionamento dell'intera rete. In molti casi, l'adozione delle block-

[21] *Bitcoin: Attacco della maggiornaza (majority attack).* URL: `https://en.bitcoin.it/wiki/Majority_attack`. (Visitato: 10.11.2018).

chain porta con sé vantaggi, ma anche ripercussioni circa questioni come la scalabilità e i costi di funzionamento.

Saranno introdotte alcune ulteriori caratteristiche della tecnologia che derivano dai concetti presentati nel primo capitolo e che, per motivi di chiarezza dell'esposizione non erano adatte ad essere inserite nelle loro parti introduttive. In molti casi le diverse componenti collaborano alla creazione di caratteristiche e di soluzioni e pertanto è lecito parlarne solo dopo che sono state introdotte nel loro insieme.

2.3.1 Transazioni e prevenzione della doppia spesa

Tra le componenti fondamentali delle blockchain un ruolo di primo piano spetta alle transazioni. Dopo una prima definizione nel capitolo 2.1 in relazione alla struttura della rete, queste necessitano di ulteriori approfondimenti.

I primi due elementi essenziali per poter effettuare una transazione sono: l'indirizzo mittente e destinatario. Questi indirizzi corrispondono agli account degli utenti che effettuano e ricevono le transazioni. Così come avviene nelle applicazioni web tradizionali, gli account vengono identificati da un username e una password, nelle blockchain vengono semplicemente usati indirizzi (*address*) univoci. Questo non solo è sufficiente ai fini dell'autenticazione, ma contribuisce alla sicurezza grazie all'uso della crittografia asimmetrica[22] e al sistema di firme digitali[23]. Questa estensione delle transazioni si presta bene ad essere visualizzata attraverso un semplice esempio di trasferimento di token da un indirizzo ad un altro.

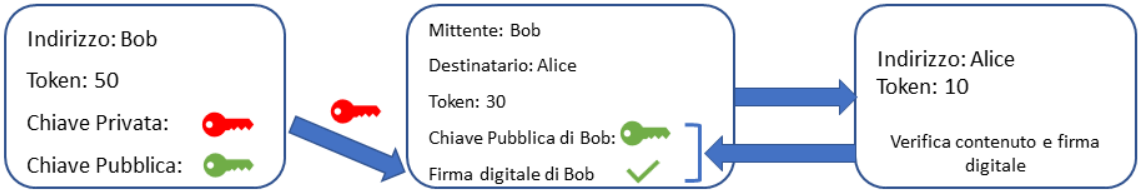

Figura 2.6: Transazione: Trasferimento Token

[22] *Crittografia Asimmetrica - Asymmetric Encryption*. URL: `https://en.wikipedia.org/wiki/Public-key_cryptography`.

[23] *Firme digitali - Digital signature*. URL: `https://en.wikipedia.org/wiki/Digital_signature`.

Nella transazione, in figura 2.6, vengono inviati 20 token (o qualunque criptovaluta implementata nella blockchain) verso un altro utente della rete. L'uso della crittografia asimmetrica permette di condividere la propria chiave pubblica all'intera rete in modo da poter essere identificati, in questo caso, come mittente della transazione. Dall'altra parte la chiave privata (segreta) corrisponde per certe caratteristiche alle tradizionali password in quanto prova il possesso della chiave pubblica e pertanto consente di identificare in maniera sicura l'utente ma in più permette anche di firmare le transazioni. Una transazione firmata, cioè dotata di firma digitale, costruita combinando la chiave privata del mittente e il contenuto della transazione, assicura l'integrità e la provenienza della transazione. Un algoritmo verificherà la firma digitale determinando se il contenuto corrisponde a quello originale e quindi se non è stato modificato.

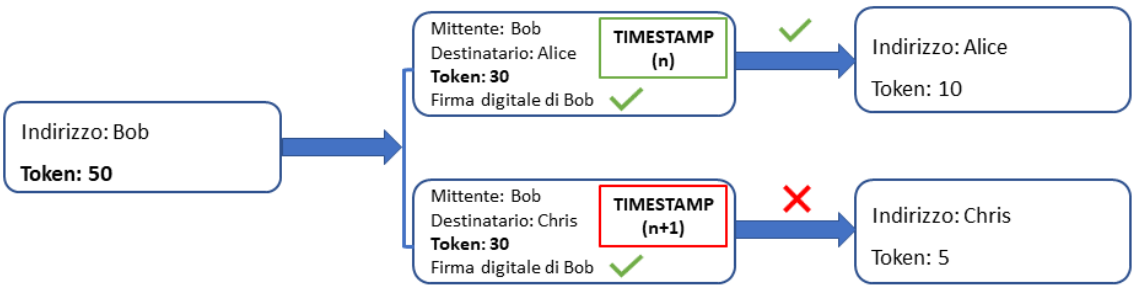

Figura 2.7: Transazione: Doppia Spesa

Un altro problema fondamentale circa le transazioni è il problema della doppia spesa illustrato nella figura 2.7. Consiste essenzialmente nell'abilità di spendere più di una volta la stessa quantità di valuta. La prima applicazione della blockchain aveva come scopo la creazione di un sistema monetario digitale dunque era fondamentale risolvere la questione. Effettivamente, combinando le caratteristiche delle blockchain, il sistema di consenso e la crittografia, il problema della doppia spesa può essere risolto in un ecosistema decentralizzato[24].

[24] *Bitcoin: Attacchi e prevenzione della doppia spesa.* URL: `https://en.bitcoin.it/wiki/Irreversible_Transactions`. (Visitato: 10.11.2018).

In dettaglio le transazioni contengono una propria marcatura temporale e durante il loro processo di validazione seguono il criterio della sequenzialità. Il tutto avviene in un contesto trasparente con a disposizione tutta la storia delle transazioni avvenute nel sistema (equivalente a una base di dati dotata di storia completa). In questo modo il contenuto delle transazioni è interamente tracciabile e pertanto può essere esplicitamente validata la loro effettiva disponibilità, in questo caso, di token. Seguendo questo ragionamento in realtà non sarebbe necessario tenere un bilancio esplicito di token totali, ma analizzando lo storico complessivo dei trasferimenti di valori specifici, è possibile risalire all'importo in questione da cui deriva la trasparenza della rete blockchain.

2.3.2 Teoria dei giochi e governabilità

Una nozione importante per il funzionamento del paradigma blockchain è relativa al sistema di consenso incentivato. Riprendendo il principio delle blockchain pubbliche ogni utente dovrebbe, in qualunque momento, essere in grado di partecipare ed effettuare modifiche al registro comune affidandosi alle regole della rete che garantiscono la terminazione delle operazioni effettuate. Le blockchain, così come tutti gli altri servizi web, devono essere in grado di offrire un servizio consistente producendo risultati deterministici con la peculiarità del contesto distribuito e decentralizzato.

Nell'assenza di un'entità centrale l'obiettivo viene raggiunto applicando un sistema di ricompense che segue i principi della teoria dei giochi[25]. Un modello ragionevole, dal punto di vista economico, cerca di mantenere un equilibrio tra ricompense previste per utenti che seguono le regole e punizioni per chi le trasgredisce. In precedenza (nel capitolo 2.2) si è visto che a intervalli di tempo regolari viene avviata una competizione, al vincitore della quale spetta il potere di appendere un blocco alla catena e ricevere la ricompensa prevista. Detto questo i dettagli sono strettamente legati alle singole implementazioni della tecnologia. Generalmente si cerca di promuovere la cooperazione a beneficio della rete e quindi di tutti gli utenti. L'obiettivo finale è quello di costruire la fiducia in una rete non supervisionata, creando condizioni in cui i procedimenti di verifica o validazione siano ricompensati e quindi benefici per l'intero sistema.

[25] *Teoria dei giochi - Game theory*. URL: `https://en.wikipedia.org/wiki/Game_theory`.

Un contesto decentralizzato implica anche l'adozione di regole di *governance* precise. Queste regole e procedure riguardano proprio la gestione e il controllo di questo fenomeno collettivo, messo in pratica, attraverso una rete decentralizzata. La governance riguarda tradizionalmente: la fiducia nelle istituzioni, enti centralizzati, governi ecc. Per l'oggetto di questa tesi la governance si fonda sulle tecnologie di regolamentazione basate sulla trasparenza, disponibilità di dati immutabili e permanenti, sulla esecuzione automatica di contratti (smart contracts di cui si parlerà nel capitolo 3.2) e su una gestione comunitaria dell'intera rete.

Secondo gli analisti della STOA (Science and Technology Options Assessment), gruppo di ricerca del parlamento europeo, le blockchain potrebbero portare alla diffusione di *organizzazioni autonome decentralizzate* (*DAO*) governate dalla collettività. I potenziali sviluppi di un nuovo sistema di governance possono favorire la nascita di sistemi organizzati, non più gerarchicamente attraverso il controllo dall'alto verso il basso, ma più democratici ed efficienti, dotati di utilità sociale che rispecchia direttamente i valori perseguiti dalla comunità[26].

2.3.3 Trilemma scalability

L'adozione di tutti questi concetti potenzialmente rivoluzionari porta con sé anche dei compromessi. Per poter presentare l'estensione del problema un esempio significante riguarda la velocità del funzionamento. In relazione ai sistemi in uso come il protocollo finanziario *Visa*, le blockchain pubbliche sono molto più lente. Il sistema Visa, in media, è in grado di processare circa 1.700 transazioni al secondo (con potenziali capacità di scalabilità di oltre 24.000 transazioni al secondo[27]), mentre allo stato dell'arte attuale, il throughput (inteso come capacità di processare le operazioni) per le blockchain più popolari Bitcoin ed Ethereum è pari rispettivamente a circa 7 transazioni al secondo e 15

[26]Philip Boucher, Susana Nascimento, Mihalis Kritikos. ≪How blockchain technology could change our lives≫. In: (2018). DOI: 10.2861/926645. URL: https://publications.europa.eu/en/publication-detail/-/publication/9964fbfd-6141-11e7-8dc1-01aa75ed71a1.

[27]*Throughput - Visa*. URL: https://usa.visa.com/run-your-business/small-business-tools/retail.html.

transazioni al secondo nel caso di Ethereum. Si può notare che le transazioni Visa sono puramente finanziarie mentre il vantaggio delle blockchain deriva dalla possibilità di poter processare qualunque tipo di operazione di scambio. Tuttavia il problema persiste (e non è correlato alla tipologia delle transazioni), ed è difficilmente accettabile che, al crescere della rete, gli utenti siano disposti ad aspettare tempi sempre più lunghi dovuti al processo di validazione (creazione di nuovi blocchi di transazioni controllata dal sistema di consenso). Un discorso simile può essere fatto per la scalabilità in termini di spazio, i registri distribuiti dotati di storia completa devono essere sincronizzati per potervi partecipare in maniera sicura. Questo comporta uno spazio in continua crescita inadatto, sia per i tempi necessari per una prima sincronizzazione (dal blocco genesis al blocco attuale), sia per la disponibilità di memoria nei dispositivi da cui è possibile accedere (ad esempio, scaricare una copia di centinaia di gigabyte su un cellulare non è praticabile).

Introdotto questo problema della scalabilità lo stato dell'arte della tecnologia blockchain è in grado di garantire simultaneamente due su tre proprietà fondamentali, come illustrato nella figura 2.8.

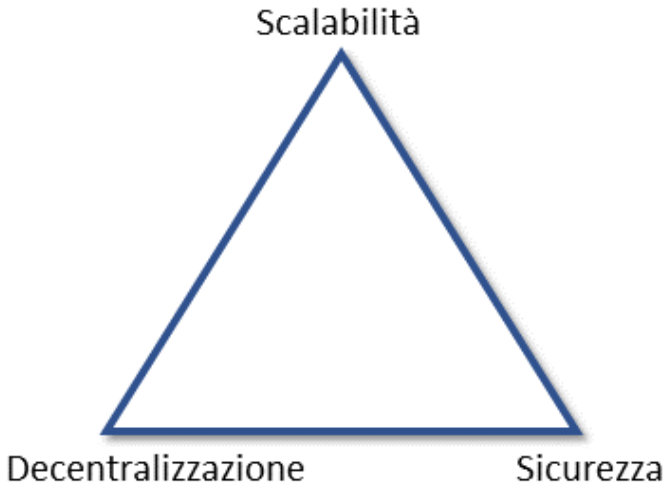

Figura 2.8: Trilemma Scalability

La combinazione delle proprietà individuate: la decentralizzazione, la scalabilità e la sicurezza, della figura 2.8 prende il nome di *trilemma scalability*[28]. La difficoltà di mantenere un equilibrio, un compromesso tra queste tre proprietà è uno dei problemi principali

[28] *Trilemma scalability*. URL: `https://github.com/ethereum/wiki/wiki/Sharding-FAQs#this-`

del paradigma blockchain. Attualmente è in corso la ricerca sui metodi e sulle soluzioni per ovviare alla necessità di sacrificare almeno una di queste tre proprietà. Un'analisi più approfondita esula dallo scopo di questa tesi, ma è opportuno menzionare un certo numero di direzioni verso quali, al momento dello scrivere, tende la ricerca. Tra questo è lecito considerare come più promettenti le seguenti soluzioni: *Sharding*, *Sidechains*, *Plasma* e molte altre.

Infine per certe applicazioni il trilemma scalability potrebbe non costituire affatto un problema. Nella prossima sezione si parlerà dell'adozione di una tipologia di blockchain con permessi (*permissioned*) che, per ragioni pratiche, non necessitano l'applicazione simultanea delle tre proprietà appena descritte.

2.4 Classificazione di sistemi distribuiti

In funzione dei principi che concorrono alla formazione delle blockchain è ragionevole pensare che, a seconda delle applicazioni che vogliono raggiungere degli obiettivi concreti, sia importante trovare un compromesso tra i vantaggi e gli svantaggi inerenti al paradigma in questione. Infatti è possibile scegliere tra i concetti (descritti nel capitolo 2) per creare un sistema blockchain adatto alle proprie esigenze. Per questo scopo è possibile estrapolare quattro caratteristiche principali del paradigma blockchain:

- Decentralizzazione
- Trasparenza
- Immutabilità
- Permanenza

Insieme questi concetti concorrono alla sicurezza dell'intera rete ma, come è stato già detto, ne derivano problemi relativi a scalability trilemma. Rinunciando a delle proprietà è possibile creare dei sistemi più efficienti, ma che si avvicinano di più ai sistemi informatici

`sounds-like-theres-some-kind-of-scalability-trilemma-at-play-what-is-this-trilemma-and-can-we-break-through-it`.

attualmente in uso. Pertanto è importante considerare attentamente i requisti del progetto a cui si vorrebbe applicare questa tecnologia per poter valutare in maniera ottimale i vantaggi e gli svantaggi, rispetto ai tradizionali sistemi centralizzati.

2.4.1 Unpermissioned Ledgers

Le blockchain pubbliche o senza permessi sono la tipologia che implementa tutti i concetti descritti fino a questo punto. Sono aperte, non hanno un proprietario e consentono indistintamente a tutti di contribuire e interagire con le risorse della rete. Come affermato in precedenza permettono a tutti di entrare in possesso di un'identica copia del registro distribuito e universalmente validato dagli utenti della blockchain.

Questa tipologia di blockchain costituisce l'oggetto principale di questa tesi, è il paradigma che ha come presupposto la decentralizzazione del potere e dei dati, rendendo quasi impossibili eventuali tentativi di censura[29]. Gli esempi di implementazioni blockchain: Bitcoin ed Ethereum sono delle blockchain pubbliche. È una tipologia promettente ma sulla quale sono necessari altri miglioramenti, in particolare nel settore della scalabilità. In questo modo potrebbero costituire una valida alternativa ai sistemi attuali e competere in un numero di ambiti applicativi maggiore.

2.4.2 Permissioned Ledgers

Le blockchain con permessi (*permissioned*) ammettono di essere più strettamente controllate. Solitamente incorporano un'entità centrale con il compito di gestire le regole della rete, stabilire i permessi degli utenti, la visibilità dei dati e così via, proprio come avviene nei sistemi centralizzati. A seconda delle necessità, in una blockchain di questo tipo, si potrebbe rinunciare a una parte del potere centralizzato o viceversa a una parte delle caratteristiche delle blockchain pubbliche, per creare applicazioni che meglio si adattano alle necessità di enti, imprese e istituzioni interessate. Ad esempio tra le tante possibilità il sistema di consenso potrebbe essere controllato da utenti imposti a priori dal

[29] *Censura nelle blockchain*. URL: `https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/`.

proprietario/i della rete, perché ritenuti fidati. Un'altra opzione utile è relativa ai diritti di leggere o di interagire con i registri, questi permessi potrebbero essere assegnati a certe categorie di utenti. La conseguenza di questo modello è collegata alla diminuzione della decentralizzazione e della trasparenza rispetto alla tipologia senza permessi.

In questo modo è possibile usufruire di alcuni concetti delle blockchain, come la struttura basata sulla crittografia, l'implementazione di smart contracts ecc. allo stesso tempo favorendo la scalabilità della rete stessa. Un esempio rilevante, già accennato in precedenza, riguarda Ripple e il suo sistema di pagamenti fondato sulla blockchain con permessi che, riesce a raggiungere un throughput simile a quello del sistema Visa. Analizzando il funzionamento di attuali sistemi bancari è facile pensare a come le transazioni, in particolare tra banche e nazioni diverse, richiedano tempo e costi connessi alle commissioni. Una potenziale applicazione di *permissioned blockchain*, come Ripple[30], potrebbe favorire la creazione di standard internazionali con un incremento della velocità di trasferimenti e la relativa diminuzione di costi di mantenimento dovuta ai dati sparsi nei diversi registri bancari.

2.5 Blockchain negli archivi digitali

Tra i tanti settori interessati alle applicazioni della tecnologia blockchain, l'area relativa ai beni culturali e archivi digitali sembra particolarmente promettente. I vantaggi derivati dalla digitalizzazione dei dati in un contesto distribuito potrebbero fornire la risposta ai problemi presenti nelle attuali basi di dati.

Questa sezione si ricollega direttamente al progetto svolto per questa tesi, i cui requisiti dettagliati sono esposti nel capitolo 4.1. Il progetto svolto ha come obiettivo l'inserimento, la consultazione e la verifica di opere d'arte minori. Si vuole dimostrare che i concetti presentati fino a questo punto si prestano bene all'analisi nell'ambito applicativo in questione.

[30] *Whitepaper Ripple - sistema di consenso*. URL: `https://ripple.com/files/ripple_consensus_whitepaper.pdf`.

Innanzitutto intuitivamente è possibile dare una prima definizione di opere d'arte minori: opere meno famose e per qualsiasi motivo giudicate meno importanti dei capolavori artistici. Questo ovviamente non diminuisce la loro importanza a livello culturale e del patrimonio umano, specie in un contesto locale. Anzi, la loro salvaguardia e valorizzazione potrebbero sotto certi aspetti rivelarsi più difficili dei loro corrispettivi "famosi". Questo perché, spesso questi oggetti sono più vulnerabili ai furti e alle contraffazioni. Una prima azione prioritaria consisterebbe proprio nella registrazione di questi oggetti in un sistema blockchain.

Tenendo in considerazione lo stato dell'arte degli archivi digitali generalmente controllati da un ente centrale con il compito di mantenere e aggiornare il relativo database, è possibile analizzare caratteristiche potenzialmente innovative del paradigma blockchain e il loro impatto in rapporto ai loro corrispettivi tradizionali:

- La decentralizzazione permette di fare a meno del controllo centrale trasferendo il compito di gestione dell'archivio alla comunità. Riguardo alla disponibilità e al mantenimento di questi sistemi a lungo termine, si toglie la responsabilità dei rischi connessi alle attività di organi centrali (ad esempio: cambiamenti di proprietà, mancanza di fondi, gestione della sicurezza del sistema ecc.), garantendo il possesso di una copia dell'archivio a tutti gli utenti.

- Grazie alla permanenza dei dati situati in registri immutabili è possibile garantire la disponibilità degli oggetti digitalizzati a chiunque e in qualunque periodo di tempo.

- La trasparenza della rete blockchain favorisce una completa tracciabilità delle opere e i relativi aggiornamenti del loro stato, come i trasferimenti di proprietà e gli spostamenti degli oggetti in questione.

- L'immutabilità, applicata attraverso regole di crittografia, contribuisce all'integrità e alla sicurezza del sistema, prevenendo fenomeni come modifiche non autorizzate e tentativi di cancellazione dei dati.

Questo breve elenco può risultare utile come punto di partenza per una riflessione finalizzata al miglioramento di sistemi esistenti.

3. Ethereum: Sviluppo di applicazioni decentralizzate

Seguendo la divisione in tre strati di concetti, implementazioni e istanze, definita fin dall'inizio del lavoro, in questo capitolo verrà affrontato l'aspetto implementativo, nel caso concreto della piattaforma Ethereum. I concetti spiegati fino a questo punto possono costituire una base decisionale, un punto di partenza per scegliere le implementazioni che meglio si adattano alle proprie necessità. Possono essere visti come una sorta di contenitore aperto da cui è possibile "pescare" le componenti concettuali da implementare con i relativi pro e contro, a seconda degli obiettivi delle applicazioni finali.

Importante sottolineare che lo stato dell'arte delle blockchain limita l'utilità di descrivere in dettaglio un certo numero di implementazioni concrete. Questa affermazione deriva dal contesto attuale: le blockchain nonostante non siano più definibili come una tecnologia fra le più recenti in confronto alle tempistiche del settore informatico, generalmente non hanno raggiunto una stabilità sufficiente. Usando la terminologia presente nelle analisi e predizioni annuali condotte dalla Gartner[1], la blockchain si trova attualmente sulla via di uscita dal cosiddetto "*hype cycle*"[2] [3]. Una metodologia usata per rappresentare la maturità, l'adozione e l'applicazione di nuove tecnologie.

[1] Gartner è una società di consulenza nel campo dell'Information Technology. Effettua analisi e ricerche al fine di produrre previsioni attendibili per i vari processi decisionali e di investimento.

[2] *Gartner - Hype Cycle for Emerging Technologies, 2016.* URL: `https://www.gartner.com/newsroom/id/3412017`.

[3] *Gartner - Hype Cycle for Emerging Technologies, 2018.* URL: `https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/`.

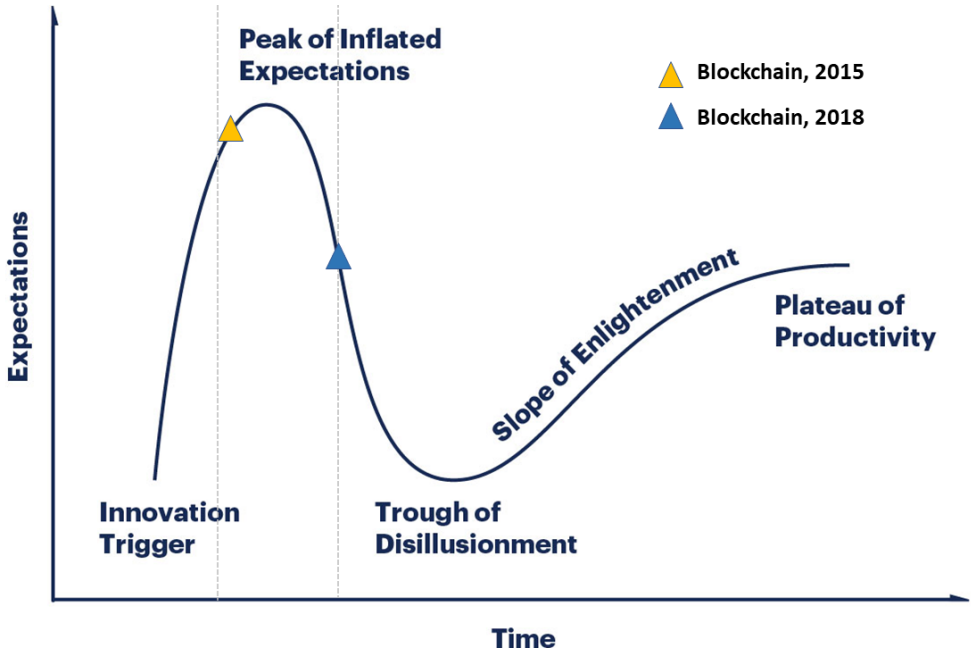

Figura 3.1: Tecnologia blockchain: Hype Cycle

La situazione in figura 3.1, può essere vista come un contesto variabile, di continuo sviluppo con l'emergere di nuove implementazioni, piattaforme e applicazioni. Si tratta di una ricerca spesso finalizzata al miglioramento e all'adozione di nuove tecniche per contrastare le limitazioni attuali del paradigma blockchain, in una situazione, qual è quella presente, in continuo mutamento, caratterizzata dall'instabilità e in alcuni casi da cambiamenti radicali. Tuttavia questo non pregiudica l'utilità delle cose dette finora in quanto si tratta di concetti fondamentali che, nel caso peggiore, possono servire anche per comprendere le scelte e gli sviluppi futuri della tecnologia.

La scelta di Ethereum come sistema da presentare, collegato alla successiva costruzione di un'applicazione, è stata fatta seguendo, tra le altre, queste motivazioni di stabilità, di sviluppo e di innovazione. Viene perseguita l'idea legata alla correlazione tra il successo e la popolarità di un sistema, misurata tenendo conto del numero di partecipanti e di sviluppatori coinvolti, degli strumenti di sviluppo messi a disposizione e dal loro grado di maturità. Si è tenuto conto anche della frequenza di aggiornamenti e dell'interesse della comunità, degli enti pubblici e privati con i relativi rischi connessi. Un insieme di fattori che le nuove implementazioni dovranno cercare di raggiungere per attrarre nuovi utenti e

sviluppatori ai loro sistemi. Al momento dello scrivere la piattaforma Ethereum è il sistema più sviluppato sotto questi aspetti anche grazie all'integrazione di strumenti per la costruzione di applicazioni decentralizzate.

3.1 Implementazione Ethereum

Ethereum è una piattaforma di sviluppo delle applicazioni decentralizzate basata sulla tecnologia blockchain, pubblica e open-source.

La novità principale è data dalla presenza di un linguaggio di programmazione Turing completo. Si tratta dunque di una blockchain programmabile con un linguaggio chiamato Solidity, attraverso cui è possibile esprimere qualsiasi algoritmo esprimibile con altri linguaggi di programmazione universali. Questi programmi scritti e compilati all'interno della blockchain prendono il nome di "*Smart Contracts*" e si comportano come dei contratti veri e propri. In rapporto ai loro corrispettivi tradizionali i termini contrattuali non sono definiti usando un linguaggio legale, ma direttamente il codice dei programmi (chiamati appunto contratti). Gli smart contracts permettono l'implementazione della logica delle transazioni e la disintermediazione tra le parti interessate, grazie ai concetti che guidano le blockchain.

Riassumendo, la visione del progetto Ethereum è quella di creare un computer decentralizzato permanente e autosufficiente, in grado di resistere ai tentativi di censura. Dal punto di vista della programmazione, nella figura 3.2 è rappresentato il flusso relativo all'architettura delle applicazioni blockchain.

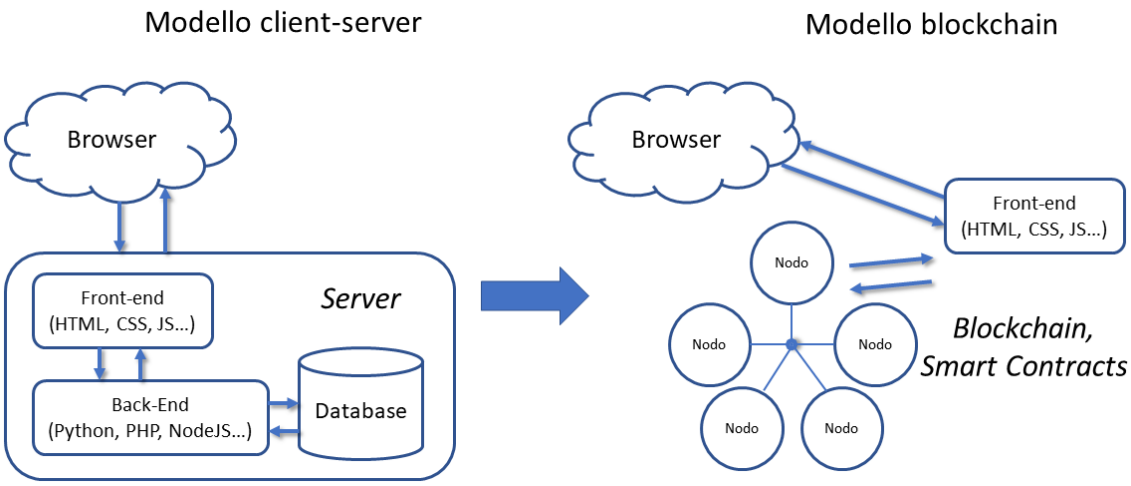

Figura 3.2: Modello architetturale: Client-server vs. Blockchain

Differentemente dal classico modello client-server, nel modello decentralizzato lo strato back-end è direttamente implementato nella blockchain attraverso gli smart contracts. L'esecuzione della logica dei contratti avviene grazie alla macchina virtuale Ethereum (EVM) eseguita da ciascun nodo della rete, estendendo così il concetto di database distribuito con la nozione di calcolo distribuito (distributed computing[4]). Ethereum prevede un meccanismo di incentivi incorporato nel sistema di consenso della rete attraverso l'uso di una criptovaluta chiamata Ether. La logica transazionale (estesa con gli smart contracts) in relazione alla complessità di calcoli computazionali, è regolata da un costo espresso in unità di Gas corrispondente a una frazione di Ether. Si tratta di un meccanismo grazie al quale è possibile prevenire lo spam dovuto ad esempio all'esecuzione di programmi che non terminano in un tempo finito.

In questo capitolo verranno analizzate queste modifiche ed espansioni della blockchain Ethereum.

3.1.1 Transazioni

Le transazioni in Ethereum sono estese con tre nuove proprietà:

- Un campo di dati opzionale

[4] *Calcolo distribuito - distributed computing*. URL: `https://en.wikipedia.org/wiki/Distributed_computing`. (Visitato: 10.11.2018).

- Un valore Start Gas
- Un valore Gas Price

In riferimento alle transazioni del capitolo precedente (come ad esempio quella in figura 2.6), queste proprietà vengono aggiunte ai campi già descritti precedentemente: mittente, destinatario e valore da trasferire. Le nuove proprietà o campi sono strettamente legati al funzionamento della EVM, che può accedere ai campi della transazione.

In particolare il campo di dati (opzionale) può essere usato per identificare specifiche operazioni sulla blockchain come ad esempio la registrazione di domini. Prima di andare avanti è opportuno spiegare che Ethereum prevede due tipi di account, quelli appartenenti agli utenti e quelli associati ai contratti. Questi account posseggono le stesse caratteristiche. Sono identificati da un indirizzo, mantengono il bilancio di token disponibili e funzionano alla stessa maniera, con l'eccezione che i primi (account utente) sono controllati da chiavi private (quindi utenti veri e propri) e gli altri dal codice eseguito in maniera automatica. Proprio per la necessità di eseguire programmi equivalenti al codice dei contratti, sono state create le proprietà Start Gas e Gas Price. Start Gas rappresenta il numero massimo di passi (*step*) computazionali che la transazione a cui si riferisce può eseguire. Gas Price, rappresenta il costo che il mittente è disposto a pagare per l'esecuzione dei programmi. Di solito l'incremento di quest'ultima proprietà permette di conquistare una certa priorità nel processo di validazione delle transazioni[5].

Come già introdotto, i valori Start Gas e Gas Price permettono di rimediare ai problemi relativi all'esecuzione di programmi infiniti e alla conseguente congestione della rete dovuta a un numero eccessivo di calcoli computazionali costosi.

3.1.2 Patricia Trees

La struttura dati usata in Ethereum prende il nome di Patricia Tree (o trie) e rappresenta un'estensione di Merkle Tree (per la definizione si rimanda al capitolo 2.3). Se per lo strato concettuale i Merkle Tree servivano fondamentalmente per la memorizzazione e

[5] *Whitepaper Ethereum - transazioni*. URL: `https://github.com/ethereum/wiki/wiki/White-Paper#messages-and-transactions`.

la verifica delle transazioni attraverso procedimenti di hashing ricorsivi, in Ethereum sono usati anche per memorizzare lo stato della rete e il risultato delle transazioni. In particolare in un blocco della rete al posto di una radice di Merkle Tree (tx root), sono incluse le seguenti tre radici di Patricia Tree:

- Radice dello stato
- Radice delle transazioni
- Radice delle ricevute (delle transazioni)

Analogamente a quanto detto per i Merkle Trees, in Patricia Trees grazie all'applicazione ricorsiva di funzioni hash crittografiche (in Ethereum viene usato lo standard SHA-3 e in particolare l'algoritmo Keccak256[6]) la radice diventa una sorta di impronta digitale per l'intera struttura dati sottostante. Ogni modifica a qualsiasi livello della struttura farà cambiare il risultato dell'applicazione di funzioni hash sul ramo interessato dalla modifica fino alla radice stessa.

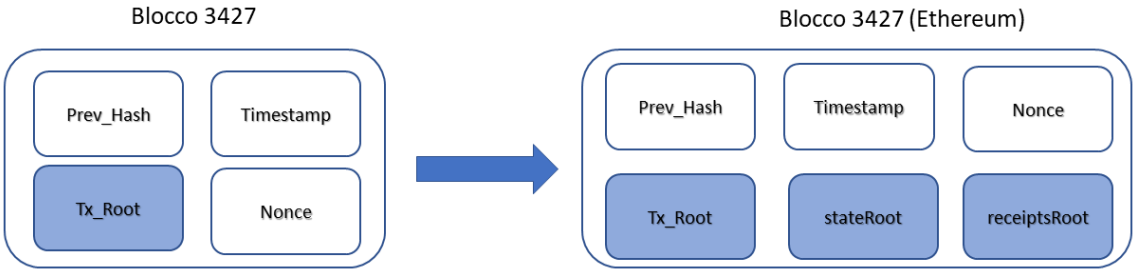

Figura 3.3: Ethereum: Semplificazione blocco

Dunque, in relazione al concetto di blocco illustrato nel capitolo 2.1, la figura 3.3 rappresenta una semplificazione del blocco Ethereum (per la completa lista delle proprietà si rimanda alla sezione 4.3. "The Block" del Yellowpaper Ethereum[7]).

Un aumento della complessità delle transazioni ha come conseguenza diretta l'espansione dell'intero stato della rete Ethereum. In questa implementazione, per lo stato della rete si intende essenzialmente lo stato di tutti gli account della blockchain (inclusi account

[6] *SHA3 - Funzioni di Hash*. URL: `https://en.wikipedia.org/wiki/SHA-3`. (Visitato: 10.11.2018).
[7] *Yellowpaper Ethereum*. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

esterni e contratti). Lo stato della rete è mutabile a causa della presenza degli smart contracts e deve essere memorizzato perché può cambiare in qualunque momento in seguito all'esecuzione automatica dei contratti. Se la struttura Merkle Tree una volta creata veniva mantenuta in maniera immutabile, in Ethereum è neccessario tenere conto di modifiche frequenti dello stato.

L'altra novità del blocco Ethereum (in figura 3.3) è la radice delle ricevute. Ogni blocco contiene ricevute permanenti delle transazioni relative al blocco che le contengono.

Patricia Tree facilita l'inserzione, l'eliminazione e l'aggiornamento di dati, obiettivi importanti per questioni di efficienza in una situazione dove lo stato della rete necessita di aggiornamenti frequenti. Infine, nelle applicazioni reali, permette agli utenti di fare ricerche come, ad esempio, trovare una data istanza di un evento "X" (ad esempio: inserimento di un oggetto nella blockchain) fatta da un utente "Y" con un certo indirizzo negli ultimi "Z" giorni. Per una descrizione formale di questa struttura dati si rimanda all'appendice D, "Modified Merkle Patricia Tree" del Yellowpaper Ethereum[8].

3.2 Smart Contracts

Una delle potenzialità più promettenti di Ethereum deriva dall'implementazione di un linguaggio di programmazione Turing completo, il quale permette di scrivere contratti (intelligenti) chiamati per l'appunto Smart Contracts. Il termine è stato coniato da Nick Szabo nel 1994, che lo definisce come un "protocollo di transazione computerizzato che esegue i termini di un contratto[9]". Una definizione sempre attuale che, nel caso del paradigma soggetto di questa tesi, può essere espressa come contratto o programma immutabile situato direttamente sulla blockchain. Come già detto nella sezione precedente, i contratti fanno parte dello stato della rete e quindi hanno un vero e proprio indirizzo con delle proprietà. Quando vengono soddisfatte certe condizioni descritte nel codice, automaticamente verranno avviate specifiche azioni, anch'esse definite nel codice del contratto.

[8] *Yellowpaper Ethereum*. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[9] Nick Szabo. «Smart Contracts». In: (1994). URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

I contratti, una volta creati e quindi situati sulla blockchain, ereditano le proprietà del paradigma. Sono immutabili, cioè una volta creati non possono essere più modificati, il loro codice è trasparente e quindi consultabile da qualsiasi utente e infine sono distribuiti, il risultato delle azioni compiute nei contratti può essere validato o controllato da tutti.

Queste caratteristiche favoriscono la disintermediazione e potenzialmente la nascita di una situazione dove il codice definito nei contratti diventa l'espressione dei vincoli specificati nelle normative legali. Un esempio significativo dei vantaggi derivati dall'utilizzo di questi contratti può essere fatto analizzando il servizio di finanziamento collettivo Kickstarter[10], basato sull'architettura tradizionale client-server.

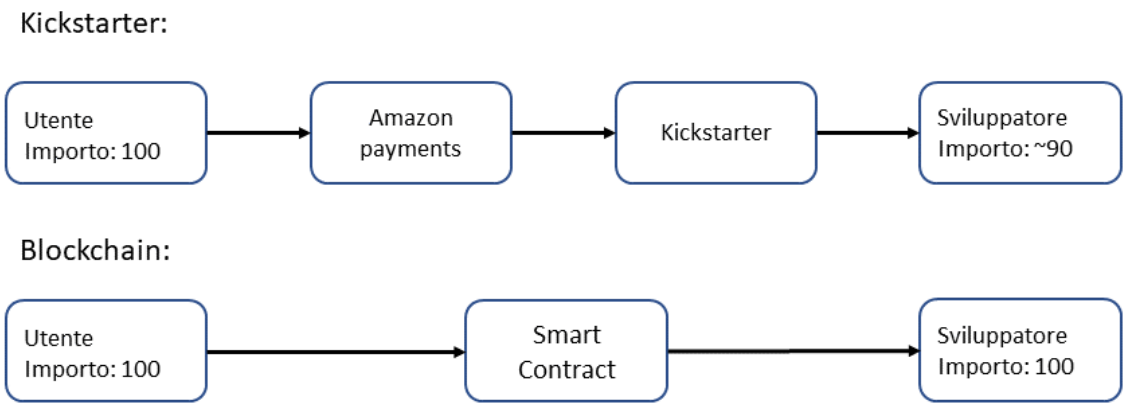

Figura 3.4: Ethereum: Esempio Finanziamento Collettivo

Il procedimento è esemplificato in figura 3.4. Sostenere un progetto con Kickstarter necessita di almeno due intermediari: il sistema di pagamenti Amazon, il quale addebita circa 3-5 percento della somma totale, e lo stesso servizio Kickstarter che guadagna il cinque per cento dei fondi raccolti. In un'applicazione analoga costruita su Ethereum è possibile fare a meno delle commissioni dovute agli intermediari mantenendo un elevato livello di sicurezza. In entrambi i casi se la raccolta non viene completata entro il termine specificato gli importi verranno restituiti agli utenti. I contratti sostituiscono anche la parte

[10] *Kickstarter - servizio di finanziamento collettivo*. URL: `https://en.wikipedia.org/wiki/Kickstarter`. (Visitato: 10.11.2018).

back-end gestita dall'intermediario garantendo che, dopo una certa scadenza definita nel contratto, gli importi saranno restituiti agli indirizzi che hanno contributo alla raccolta.

Importante notare che la scrittura di contratti immutabili deve essere preceduta da un'accurata fase di test automatizzati e simulazioni in modo da individuare eventuali errori e comportamenti non previsti dei programmi che potrebbero avere gravi conseguenze una volta pubblicati sulla rete.

3.2.1 Solidity

Solidity è uno dei principali linguaggi di programmazione orientato alla scrittura dei contratti sulla blockchain[11]. Implementato da Ethereum, è un linguaggio ad alto livello, fortemente tipato, supporta il meccanismo dell'ereditarietà e ha una sintassi ispirata a quella di C++, Python e Javascript. Un contratto Solidity è dunque una collezione di codice incapsulato nelle funzioni e di dati che formano lo stato del contratto (permanentemente memorizzato sulla blockchain) e che risiedono sotto uno specifico indirizzo della blockchain Ethereum.

```
1  pragma solidity ^0.4.25;
2
3  contract storeNum {
4    uint256 number;
5    address owner;
6
7    modifier onlyOwner {
8      require(msg.sender == owner);
9      _;
10   }
11
12   constructor(uint256 _number) public {
13     number = _number;
14     owner = msg.sender;
```

[11] *Solidity - linguaggio di programmazione di smart contracts.* URL: https://solidity.readthedocs.io.

```
15    }
16
17    function changeNum(uint256 _newNumber) public onlyOwner {
18      number = _newNumber;
19    }
20  }
```

Codice 3.1: Esempio contratto Solidity

Questo semplice esempio di contratto “storeNum” permette al suo creatore di memorizzare e modificare un numero naturale. Analizzare questo esempio può essere utile per introdurre alcune novità del linguaggio.

- *pragma* (riga 1) definisce la versione del compilatore Solidity da usare per il contratto.
- *contract storeNum* (righe 3, 5) inizializza il contratto “storeNum” con due variabili di stato: number, può contenere un numero positivo di 256bit e owner che potrà contenere un indirizzo Ethereum di 20byte.
- *modifier onlyOwner* (righe 7, 10) definisce un insieme di condizioni da applicare a una o più funzioni. Spesso viene usato per definire una guardia di controllo dell’accesso alle specifiche funzioni. In questo caso controlla che l’indirizzo che chiama il contratto sia equivalente a quello di chi lo ha creato.
    - *require* è un’istruzione che può creare un’eccezione con il conseguente ripristino dello stato del contratto a quello immediatamente antecedente alla sua chiamata. Require è fondamente per la gestione degli errori in Solidity, è buona regola utilizzarlo il prima possibile per non intercorrere nei costi dovuti all’esecuzione di istruzioni che devono soddisfare certi requisiti oppure possono produrre errori.
    - *msg.sender* è un’istruzione predefinita di Solidity, restituisce l’indirizzo che, al momento della chiamata, sta interagendo con il contratto.

    - _; è un'istruzione specifica dei modifier, stabilisce il punto nel quale viene inserito il corpo della funzione da essa controllata. Dunque, se il flusso di esecuzione del programma non viene interrotto fino a questo punto allora solo adesso verrà eseguito il codice della funzione.

- *constructor* (righe 12, 15) funziona esattamente come in altri linguaggi, al momento della creazione il contratto viene inizializzato con valori specificati nel costruttore.

- *changeNum* (righe 17, 19) questa funzione permette di modificare il valore memorizzato nella variabile di stato number. È controllata dal modifier onlyOwner e quindi verrà eseguita solo se viene chiamata dall'indirizzo che ha creato il contratto altrimenti nessun cambiamento di stato verrà effettuato nel contratto. Questa modifica non pregiudica l'immutabilità dei dati, il valore storico della variabile number sarà sempre contenuto nei blocchi precedenti.

Un'ultima considerazione sull'esempio del contratto riguarda la visibilità e l'accesso alle funzioni dichiarate. In Solidity le funzioni sono di default pubbliche (*public*) e quindi possono essere chiamate ed eseguite da tutti, pertanto, è raccomandabile specificare questo parametro nella dichiarazione della funzione stessa. La documentazione già citata prevede parametri tra cui "private" (solo le funzioni all'interno del contratto in cui sono contenute potranno chiamare questa funzione), "view" (per funzioni che non cambiano lo stato del contratto) e molte altre. Un modo immediato per provare ed eseguire i contratti (come l'esempio del contratto storeNum di questa sezione) senza la necessità di operare sulla blockchain è quello di usare il compilatore ufficiale Remix[12] di Ethereum.

Supponendo che il contratto sia corretto e termini in un tempo finito, il passo successivo consiste nella compilazione del codice Solidity in byte-code eseguibile da EVM in maniera simile a come avviene con JAVA e la componente JVM[13].

[12] *Remix - Solidity IDE*. URL: `https://remix.ethereum.org/`.

[13] *Java Virtual Machine*. URL: `https://en.wikipedia.org/wiki/Java_virtual_machine`. (Visitato: 10.11.2018).

3.2.2 Ethereum Virtual Machine

Ethereum Virtual Machine o EVM, è una macchina virtuale che viene eseguita da ciascun partecipante della rete Ethereum. I contratti scritti in un linguaggio ad alto livello come ad esempio Solidity sono compilati in byte-code EVM che, nell'insieme rappresenta una sequenza di istruzioni di basso livello chiamate opcode[14], eseguibili dalla macchina virtuale. L'esecuzione avviene in modalità protetta (*sandbox*[15]) durante la quale il codice eseguito è isolato, l'EVM (durante il *runtime*) non ha accesso alla rete, ai file locali di sistema o ad altri processi in esecuzione. A livello della rete tutti gli utenti eseguono in maniera indipendente i calcoli sulla EVM, definibile quindi come un computer globale condiviso o *world computer*.

Formalmente la EVM spesso è definita come una macchina quasi Turing completa. Questa affermazione deriva da un ulteriore vincolo legato al costo computazionale delle istruzioni eseguite al suo interno, misurato in gas, un parametro che limita la quantità di computazione che può essere compiuta durante l'esecuzione[16]. Ricapitolando, a livello dell'implementazione, la EVM assume un ruolo centrale per l'applicazione dei concetti descritti nelle sezioni precedenti. Concetti relativi alle transazioni, al calcolo e al consenso distribuito nella blockchain Ethereum.

3.2.3 Gas

Il costo della computazione è rappresentato attraverso Gas, un'unità di misura corrispondente a una frazione di Ether. Gas può essere considerato come una specie di carburante per la EVM, ogni istruzione eseguita al suo interno consumerà una quantità predefinita di Gas. A livello di transazioni l'utente può impostare l'ammontare di Gas che è disposto a spendere per eseguire una certa transazione. In particolare Gas price indica il prezzo misurato in *gwei*, dove 10^9 gwei corrispondono a 1 ether cioè 1 ether equivale a 1,000,000,000

[14] *EVM - Lista opcodes*. URL: `https://ethervm.io/#opcodes`.

[15] *Sandbox - Sicurezza informatica*. URL: `https://en.wikipedia.org/wiki/Sandbox_(computer_security)`. (Visitato: 10.11.2018).

[16] *Yellowpaper Ethereum*. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

gwei[17]. Il parametro Gas start (in questo contesto chiamato anche Gas limit) impone un limite alla quantità di Gas per la quale l'utente è disposto a pagare il prezzo stabilito. Infine il prodotto tra questi due valori, Gas price e Gas start, indica il limite superiore del costo di esecuzione che l'utente vorrebbe spendere, così come nel seguente esempio in figura 3.5.

Figura 3.5: Gas: Esempio Calcolo Commissioni

Le transazioni verranno eseguite dalla EVM finché il programma termina oppure termina la quantità di Gas disponibile per la transazione. Nel caso di una transazione avvenuta con successo al mittente sarà rimborsato il costo di Gas non utilizzato, altrimenti il Gas verrà consumato e la EVM produrrà un'eccezione *out of gas*. Se la quantità di carburante risulta insufficiente per l'esecuzione di una data transazione tutti i cambiamenti effettuati fino a quel punto saranno scartati (ritorno allo stato prima della transazione), mentre il Gas sarà consumato e quindi non verrà restituito al mittente[18].

Attenendosi alle condizioni della rete e riguardo i prezzi delle transazioni concluse con successo è possibile fare stime affidabili circa il costo e il tempo di una transazione[19]. Il concetto di Gas è puramente riferito ai calcoli, non è possibile mandare o ricevere unità di Gas, è una misura intesa come una costante con un proprio costo indipendente dalle variazioni di valore di Ether.

[17] *Ether conversioni*. URL: `http://ethdocs.org/en/latest/ether.html`.

[18] *Yellowpaper Ethereum*. URL: `https://ethereum.github.io/yellowpaper/paper.pdf`.

[19] ETH Gas Station (https://ethgasstation.info/) è uno dei siti principali che effettua il monitoraggio dei costi nella rete Ethereum

3.3 Data storage

Una delle operazioni più costose nelle blockchain è la memorizzazione[20]. Memorizzare in maniera permanente file di grandi dimensioni, che dovranno essere replicati e sincronizzati da tutti i partecipanti della rete, implica un costo elevato espresso in termini di Gas. Le blockchain pubbliche, nonostante possano essere definibili come basi di dati distribuite, sono inadeguate per applicazioni che richiedono un uso significativo in termini di spazio di memoria. La regola di base è quella di includere direttamente nelle blockchain soltanto le informazioni, come ad esempio i termini di un contratto, che dovranno godere dei benefici della rete (immutabilità, permanenza, trasparenza ecc.) e memorizzare il resto attraverso servizi esterni.

Tuttavia è evidente che l'uso di servizi di Cloud Storage tradizionali come AmazonS3, Microsoft Azure o altri sistemi centralizzati limiterebbe i vantaggi delle applicazioni propriamente decentralizzate. Per questo la nascita della tecnologia blockchain è accompagnata dallo sviluppo di sistemi di memorizzazione basati in tutto o in parte sugli stessi principi del paradigma blockchain. Attualmente sono in via di sviluppo molti sistemi di Storage decentralizzato, tra questi i principali Storj[21], Sia[22], BigchanDB[23], Ethereum Swarm[24] e IPFS[25].

Questi servizi promettono di usufruire delle novità delle blockchain favorendo la disintermediazione con la conseguente riduzione dei costi dovuti alla memorizzazione di dati e un maggiore controllo su di essi da parte degli utenti finali. In particolare si è scelto IPFS per lo strato di memorizzazione dell'applicazione decentralizzata che accompagna questo lavoro.

[20] Indicata dall'opcode "SSTORE" contenuto nell'appendice G, "Fee Schedule" di Ethereum Yellowpaper

[21] *Storj - Decentralized Cloud Storage*. URL: `https://storj.io/`.

[22] *Sia - Decentralized Cloud Storage*. URL: `https://sia.tech/`.

[23] *BigchainDB - Blockchain Database*. URL: `https://www.bigchaindb.com/`.

[24] *Swarm - Distributed Storage Platform*. URL: `https://swarm-guide.readthedocs.io/en/latest/introduction.html`.

[25] *IPFS - Distributed Web*. URL: `https://ipfs.io/`.

3.3.1 IPFS

IPFS (InterPlanetary File System) è un progetto open-source i cui obiettivi vanno ben oltre l'offerta di un servizio di storage distribuito. Lo scopo principale di IPFS è quello di costruire un file system globale distribuito basato su peer-to-peer in modo da creare una nuova architettura Internet permanente e distribuita. I protocolli alla base di Internet attuale si sono sviluppati secondo livelli di astrazione aggiungendo negli anni via via nuove funzionalità, IPFS costituisce un tentativo di sostituzione e di miglioramento di questi protocolli.

In particolare, riguardo al protocollo di trasmissione HTTP, la differenza sostanziale consiste nel passaggio da un architettura client-server basata su *location based addressing* (indirizzamento in base alla posizione) a un architettura peer-to-peer basata sul *content based addressing* (indirizzamento in base al contenuto). Questo implica una caratteristica importante che può giustificare la scelta di questa particolare tecnologia e la sua integrazione nelle applicazioni blockchain. La caratteristica menzionata riguarda l'indirizzamento delle informazioni in base al contenuto reso possibile grazie all'utilizzo dalle funzioni crittografiche di hash. Memorizzare un dato su IPFS genererà, a partire dal contenuto, un indirizzo hash che garantisce l'immutabilità del contenuto memorizzato. A livello dell'architettura l'insieme degli indirizzi IPFS è archiviato attraverso la tabella di hash distribuita, DHT (*Distributed Hash Table*[26]).

[26] *DHT - Distributed Hash Table*. URL: `https://en.wikipedia.org/wiki/Distributed_hash_table`. (Visitato: 10.11.2018).

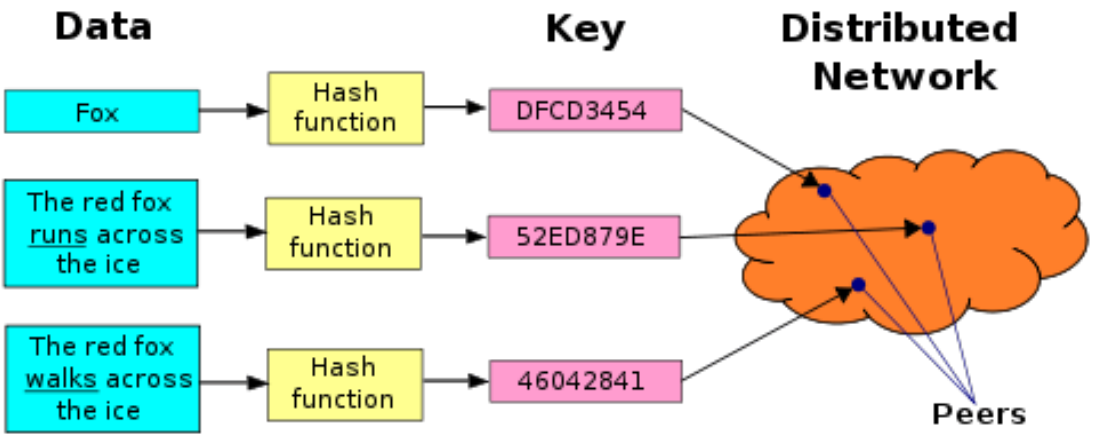

Figura 3.6: IPFS: Indirizzi Hash e DHT

Nella figura 3.6 si può vedere come la chiave (*key*) generata a partire dal contenuto (*data*) entra a far parte della rete distribuita (*DHT*) di cui costituisce un indirizzo univoco.

Le tecnologie alla base di IPFS sono numerose e la loro descrizione esula dagli scopi di questa tesi, per approfondimenti si rimanda al Whitepaper IPFS[27], che costituisce un punto di riferimento per lo studio della tecnologia.

La memorizzazione dei dati in maniera distribuita necessita, così come nelle blockchain, di un sistema di incentivi. IPFS allo stato dell'arte non ha ancora implementato un meccanismo del genere e dunque non permette di memorizzare dati in maniera permanente. Lo sviluppo della tecnologia mira alla costruzione di un'infrastruttura che, una volta raggiunta una fase stabile, permetterà di implementare questo tipo di meccanismi. Filecoin[28] costituisce un esempio di servizio di storage implementato sulla blockchain, che potrà essere integrato su IPFS.

[27] *IPFS - Whitepaper (Draft 3).* URL: `https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf`.

[28] *Filecoin - Decentralized Storage Network.* URL: `https://filecoin.io/`.

4. Progetto di Archivi Digitali

Questo capitolo contiene la documentazione del progetto realizzato utilizzando le tecnologie descritte nei capitoli precedenti. Si tratta di un'applicazione decentralizzata costruita attraverso la blockchain pubblica Ethereum e il protocollo IPFS. Lo scopo principale di questo progetto è la descrizione della metodologia di sviluppo in un ecosistema distribuito. Se per molti programmatori imparare un nuovo linguaggio di programmazione o framework può essere una situazione frequente, lo sviluppo in un paradigma differente richiede una descrizione più approfondita della metodologia e delle pratiche migliori da adottare durante la fase di preparazione e di sviluppo.

4.1 Definizione di obiettivi e requisiti dell'applicazione

L'obiettivo consiste nello sviluppo di un'applicazione per la registrazione di opere d'arte definibili come *minori* che, potranno essere memorizzate in maniera trasparente, permanente e immutabile in una blockchain pubblica.

In dettaglio l'applicazione deve soddisfare i seguenti requisiti e funzionalità:

- Inserimento di oggetti nell'archivio, descritti attraverso l'utilizzo di sistemi di metadati opportuni.
- Memorizzazione e visualizzazione degli oggetti inseriti.
- Possibilità di modificare l'oggetto inserito da parte del suo autore mantenendo la storia completa delle modifiche effettuate.

Gli utenti previsti per l'applicazione sono di due tipi:

- Qualsiasi utente dotato di un indirizzo sulla rete, quindi in grado di interagire con l'applicazione, può, senza restrizioni, visualizzare tutti gli oggetti inseriti nell'archivio. Inoltre finché l'utente dispone di fondi sufficienti potrà inserire nuovi oggetti e modificare la loro descrizione.

- Utenti dotati di permessi di verifica e approvazione dei dati inseriti nell'archivio attraverso un sistema di votazione.

Tenendo in mente questi requisiti e le considerazioni fatte nel capitolo 2.5: "Blockchain negli archivi digitali" è possibile procedere con l'implementazione basata sulla blockchain Ethereum.

4.1.1 Sistemi di metadati

Prima di proseguire con la fase dello sviluppo dell'applicazione, una considerazione importante riguarda la descrizione delle risorse attraverso sistemi di metadati. Secondo una descrizione generica: "i metadati sono informazioni strutturate relative ai dati, interpretabili da parte di un computer"[1]. Inserire un oggetto dentro un archivio digitale che gestisce opere d'arte minori implica l'implementazione di un sistema di regole adatto a esprimere le diverse tipologie delle risorse in questione. Ad esempio, le risorse da rappresentare possono essere dei manoscritti, delle fotografie, dei disegni, degli oggetti di oreficeria e così via.

Esistono molti standard più o meno formali per la descrizione di particolari categorie di materiale, per questo progetto si è scelto di implementare tre modalità di rappresentazione delle risorse, le prime due appartengono allo schema di metadati Dublin Core[2], mentre la terza è un esempio del modello ontologico CIDOC Conceptual Reference Model (CRM)[3].

La scelta dei metadati è stata fatta tenendo conto di diversi fattori, innanzitutto si è cercato di scegliere gli schemi in grado di rappresentare il vasto insieme di tipologie a cui possono appartenere gli oggetti dell'archivio. Inoltre, nello scenario d'uso della salvaguardia dei beni artistici, potrebbe essere importante adottare un sistema utile alle forze dell'ordine (vedi il paragrafo successivo). Infine un'altra considerazione può essere fatta relativamente alla ricerca di un compromesso tra la complessità della descrizione, la facilità di utilizzo per un utente non esperto in materia e il grado di precisione del siste-

[1] Pasquale Savino. *Corso Biblioteche Digitali*. URL: `http://www.nmis.isti.cnr.it/savino/Dispensa/metadati.pdf`.

[2] *Dublin Core Metadata Initiative*. URL: `http://dublincore.org/`.

[3] *CIDOC CRM - Conceptual Reference Model*. URL: `http://www.cidoc-crm.org/`.

ma utilizzato. Per questo l'utente può scegliere tra tre modalità di inserimento, dal più semplice al più complesso. Dublin Core fornisce uno schema di meta informazioni ideato per assegnare etichette ragionevolmente interessanti per qualunque materiale digitale. È uno schema flessibile, semplice ed estendibile che si presta bene alla costruzione di form adatte per la maggior parte dei casi di utilizzo previsti.

In figura 4.1 è rappresentata una parte della finestra contenente il modulo di inserimento, implementato nel progetto, con le scelte relative ai sistemi di descrizione da utilizzare.

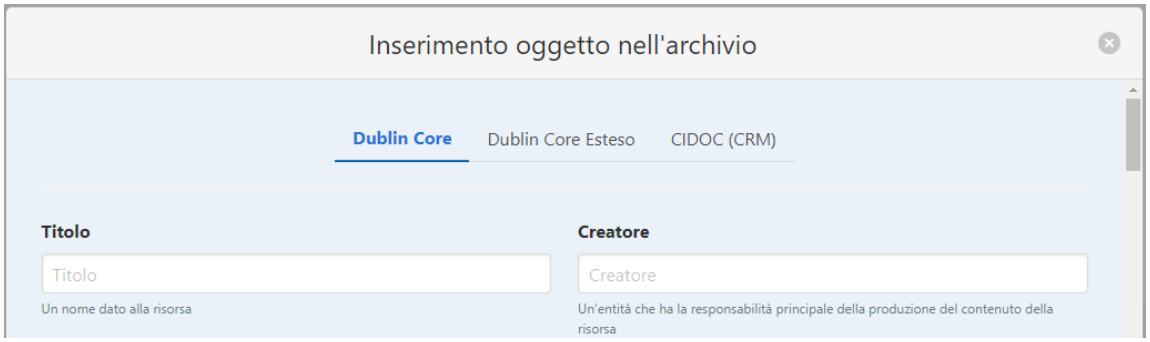

Figura 4.1: Progetto: Modulo inserimento nell'archivio

La prima opzione, "Dublin Core", fornisce quindici etichette base per la descrizione delle risorse. La seconda opzione, "Dublin Core Esteso", permette una descrizione più approfondita della risorsa attraverso l'uso di qualificatori (o sottoclassi) che permettono un raffinamento dello schema con l'aggiunta di significati più precisi sui termini base. Infine la terza opzione, "CIDOC", è rivolta agli *esperti* in quanto fornisce un modello ontologico formale incentrato sulla rappresentazione semantica delle risorse.

Un ultima considerazione pratica riguarda le linee guida per la descrizione di beni culturali, fornite dal nucleo carabinieri per la tutela del patrimonio culturale in figura 4.2.

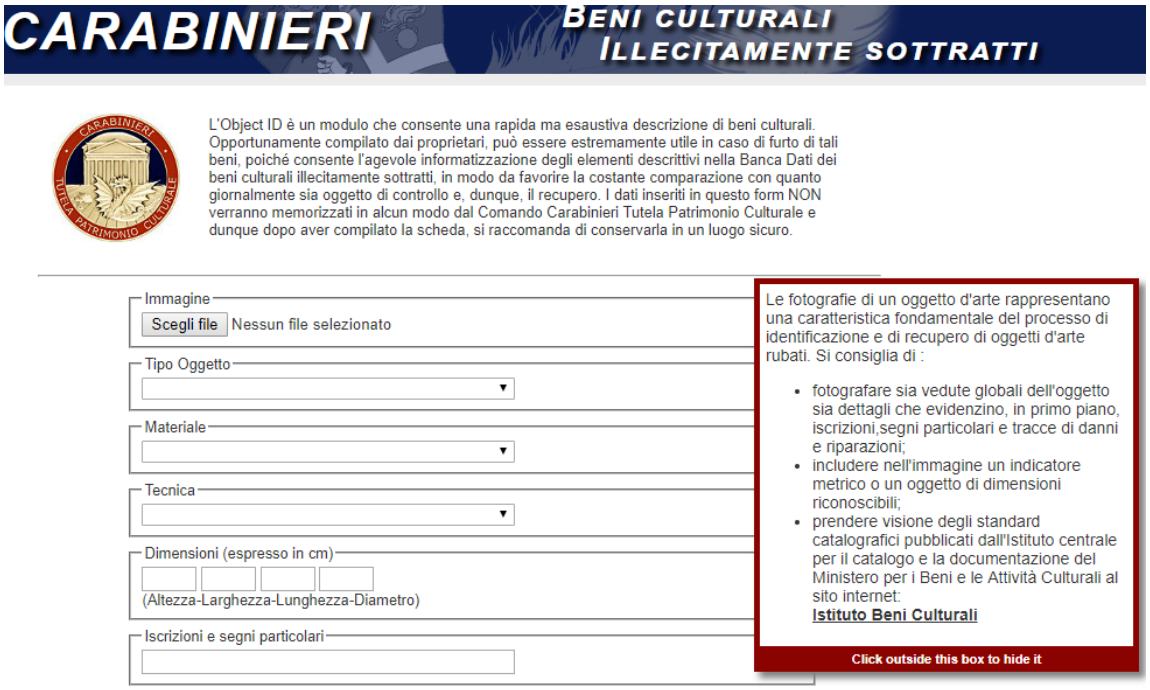

Figura 4.2: Beni culturali illecitamente sottratti: Modulo descrittivo

Il modulo Object ID[4], adottato dalle forze dell'ordine, consente la descrizione di beni culturali ed è utile per le operazioni di recupero in caso di furto di tali beni. Quando la tecnologia sarà sufficientemente matura e testata, l'approvazione degli oggetti inseriti tramite un'applicazione basata sulla blockchain, potrebbe essere fatta dai membri autorizzati del Comando Carabinieri. Chiaramente il fatto di poter contare su una base di dati immutabile, permanente e dotata di storia completa fornirebbe dei vantaggi rilevanti rispetto ai sistemi attualmente in uso.

Con questo progetto si vuole dimostrare che è possibile adottare qualunque schema di descrizione in maniera relativamente semplice e quindi fornire una prova attendibile del possesso di tali beni. Mentre queste considerazioni sono state fatte tenendo in mente l'obiettivo consistente nella dimostrazione della metodologia, nel caso di un'applicazione pronta ad essere adottata al pubblico, questa parte assumerebbe un ruolo di primo piano per l'intero processo di sviluppo.

[4] *Modulo Object ID - Comando Carabinieri Tutela Patrimonio Culturale*. URL: `http://tpcweb.carabinieri.it/SitoPubblico/objectId`.

4.2 Preparazione dell'ambiente di sviluppo

Lo sviluppo del progetto sulla blockchain Ethereum è accompagnato dalla presenza di numerosi strumenti, in questa sezione sarà fornita la lista dei principali strumenti e delle dipendenze utilizzate durante la programmazione.

- NPM (Node Package Manager) per la gestione e l'installazione dei moduli/librerie Javascript. NPM richiede la presenza di Node.js[5].

- Truffle framework (versione 4.1.14)[6] un ambiente di sviluppo composto da un insieme di strumenti che facilitano la creazione delle applicazioni decentralizzate (di seguito dApps) sulla piattaforma Ethereum. Truffle gestisce il ciclo di migrazione (sulla blockchain) degli smart contracts, la loro integrazione ed esecuzione con la possibilità di eseguire test automatizzati.

  Il framework include Ganache[7] (versione 1.2.2), una blockchain personale, utile per lo sviluppo locale delle dApps. Ganache è un emulatore di nodi Ethereum con un'interfaccia grafica intuitiva (in figura 4.3), mette a disposizione un certo numero di account precaricati con Ether. Si tratta di una blockchain con un funzionamento semplificato che permette di analizzare lo stato della blockchain in qualunque momento dell'esecuzione.

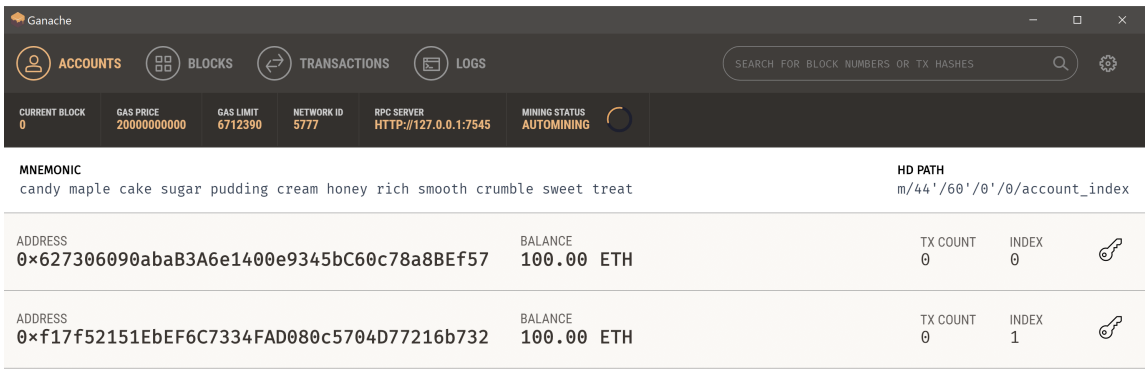

Figura 4.3: Ganache: Blockchain locale

[5] In questo progetto si è utilizzato Node.js nella versione 10.8.0 e NPM versione 6.2.0

[6] *Truffle Suite*. URL: `https://truffleframework.com/`.

[7] *Truffle Suite - Ganache*. URL: `https://truffleframework.com/ganache`.

Truffle fornisce anche il codice *boilerplate* raggruppato in diversi pacchetti chiamati "boxes", utili per uno sviluppo accelerato[8].

- Geth[9] è l'implementazione ufficiale di un nodo completo Ethereum che permette la sincronizzazione e la partecipazione alla blockchain Ethereum.

- Mocha e Chai sono usati per la creazione e l'esecuzione di test automatizzati. Mocha[10] è un test framework per Javascript basato su Node.js, mentre Chai[11] è una libreria di asserzioni che può essere implementata sul framework. Insieme permettono l'implementazione di test automatizzati complessivi secondo un approccio TDD (Test Driven Development) e BDD (Behaviour Driven Development).

- Metamask[12] è un'estensione, disponibile per i maggiori browser (Firefox, Chrome,Brave, ecc.). Necessario per utilizzare le dApps, fornisce un portafoglio (*wallet*) che consente di identificare l'utente (attraverso la chiave pubblica) e di gestire le transazioni con la chiave privata.

- Web3.js[13] (versione 0.20) è una collezione di librerie che, attraverso un API, permette l'interazione con il codice dei contratti situati sulla blockchain. In particolare, permette di sviluppare la parte client dell'applicazione con le chiamate di lettura e di scrittura ai contratti.

Questi sono gli elementi più importanti, al momento più usati per interagire con le dApps e per sviluppare le proprie applicazioni. Altri elementi scelti per lo sviluppo di questa specifica applicazione, nella maggior parte opzionali, verranno introdotti in seguito.

[8] Come base di questo progetto si è scelto il pacchetto chiamato pet-shop disponibile su https://truffleframework.com/boxes/pet-shop

[9] *Geth - Ethereum implementation*. URL: `https://geth.ethereum.org/`.

[10] *Mocha - test framework*. URL: `https://mochajs.org/`.

[11] *Chai - assertion library*. URL: `https://www.chaijs.com/`.

[12] *Metamask*. URL: `https://metamask.io/`.

[13] *Web3.js*. URL: `https://github.com/ethereum/web3.js/`.

4.3 Sviluppo dell'applicazione

Lo sviluppo di questo progetto può essere suddiviso in due fasi distinte:

- La prima fase si pone come obiettivo la creazione di un *Minimal Viable Product* (MVP) cioè un prototipo che implementata le funzionalità base. Consiste principalmente nello sviluppo dei contratti, l'esecuzione di test e costruzione di un'interfaccia di prova (GUI) che permette l'interazione con i contratti in un contesto di sviluppo locale.
- Nella seconda fase si estende l'applicazione in modo da implementare nuove funzionalità e costruire la versione finale. Questa fase consiste principalmente nell'integrazione con il servizio di storage IPFS, lo sviluppo della parte front-end finale e il trasferimento dei contratti su una blockchain Ethereum di sviluppo (*testnet*).

4.3.1 Scrittura dei Contratti

Gli smart contracts scritti in Solidity costituiscono la parte back-end e contengono la logica dell'applicazione. Nel contratto principale, *Archives.sol*, tra le prime operazioni si definisce la struttura degli oggetti da inserire sulla blockchain.

```
1 struct Artwork {
2     uint id; //identificatore univoco
3     address author; //indirizzo dell'autore
4     string name; //nome o titolo dell'oggetto
5     string descriptionHash; //descrizione completa dell'oggetto
6     string mainPreviewHash; //immagine rappresentativa dell'oggetto
7     bool validation; //stato dell'approvazione (true o false)
8     uint votesNum; //numero delle approvazioni (o voti)
9 }
```

Codice 4.1: Struttura degli oggetti

Le caratteristiche degli oggetti sono definite tramite una variabile "Artwork" di tipo *struct* cioè un tipo complesso definito dall'utente contenente al suo interno un insieme di

proprietà che definiscono l'oggetto. Il passo successivo consiste nella costruzione di una sequenza di oggetti da memorizzare sulla blockchain. In Solidity è possibile costruire una sequenza composta da coppie chiave-valore (come in un array associativo) usando il tipo *mapping*.

```
1 mapping (uint => Artwork) public artworks;
2 uint artworkCounter; //contatore degli oggetti inseriti
```

Codice 4.2: Associazione chiave-valore

In questo modo il mapping crea una specie[14] di tabella hash (inizialmente vuota[15]) composta da un indice univoco (uint) che si riferisce a un oggetto di tipo Artwork. Ogni volta che verrà eseguito un assegnamento di chiave-valore, il risultato sarà memorizzato sulla blockchain. Le tabelle hash sono una struttura dinamica e pertanto è necessario tenere una variabile di stato contatore in modo da tenere traccia degli oggetti effettivamente inseriti.

A questo punto analizzando il codice del contratto Archives.sol[16] sono state implementate le funzionalità descritte nella fase di definizione dei requisiti del progetto.

```
1 function publishArtwork(string _name, string _descriptionHash,
2                         string _dataHash) public {...}
3 function approveArtwork(uint _id) public onlyIfWhitelisted(
4                         msg.sender) checkValidation(_id) {...}
5 function modifyArtworkDescription(uint _id,
6                         string _newDescriptionHash) public {...}
```

Codice 4.3: Funzioni principali del contratto Archives.sol

[14] *Solidity - mapping*. URL: https://solidity.readthedocs.io/en/v0.4.21/types.html#mappings.

[15] In realtà è impossibile determinare a priori la dimensione del mapping o eseguire direttamente delle iterazioni sugli elementi. Questo perché tutte le chiavi restituiranno un valore di default di tipo Artwork.

[16] https://github.com/lukasd2/Digital-Archives-dApp/blob/master/contracts/Archives.sol

- La funzione ”publishArtwork” gestisce l’inserimento di un nuovo oggetto nell’archivio.
- La funzione ”approveArtwork” gestisce l’approvazione degli oggetti inserti. Permette agli utenti autorizzati di votare per un dato oggetto e di controllare se il numero di voti registrati è sufficiente alla sua approvazione.
- La funzione ”modifyArtworkDescription” permette all’utente che inserisce l’oggetto di modificare la sua descrizione.

Successivamente al contratto Archives.sol sono stati aggiunti altri contratti che espandono le funzionalità e i controlli richiesti dall’applicazione. Per la costruzione della struttura dei contratti si è sfruttato il meccanismo dell’ereditarietà di Solidity, con la gerarchia rappresentata in figura 4.4. Archives.sol si trova al livello più alto della catena e dunque può usufruire di tutte le funzioni definite nei contratti situati a livello più basso. In particolare i contratti Whitelist.sol, RBAC.sol e Roles.sol sono usati rispettivamente per aggiungere indirizzi alla lista di utenti dotati di permessi di verifica e per gestire i permessi degli utenti che acquistano lo status di ”artwork checker” con i rispettivi controlli. Il contratto Ownable.sol implementa il meccanismo di autorizzazione dell’autore del contratto, cioè all’indirizzo che pubblica il contratto sulla rete. Nella versione attuale al ”proprietario” vengono dati i permessi di aggiungere nuovi utenti alla whitelist.

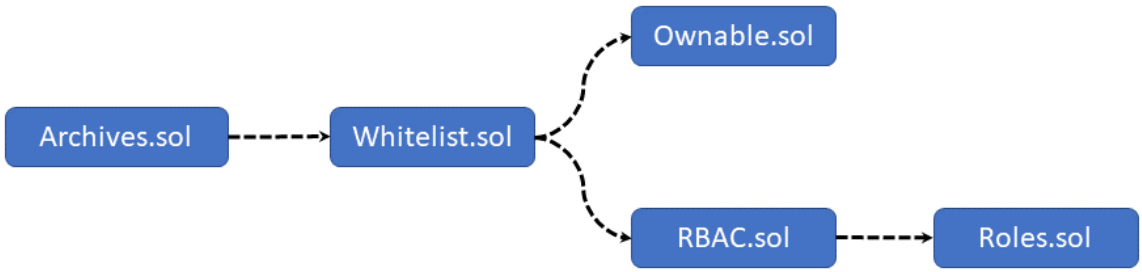

Figura 4.4: Progetto: Struttura dei contratti

Questi contratti (Whitelist.sol, Ownable.sol, RBAC.sol e Roles.sol) sono stati presi dalla libreria OpenZeppelin[17] contenente implementazioni delle funzionalità ricorrenti

[17] *Open Zeppelin*. URL: `https://openzeppelin.org/`.

nelle dApps. Sono considerati sicuri perché ampiamente testati e aggiornati dalla comunità, questi contratti sono stati opportunamente modificati e integrati in questo progetto[18].

4.3.2 Test dei Contratti

Una volta implementata la logica dell'applicazione è particolarmente importante assicurarsi che i contratti siano sicuri e privi di errori eseguendo dei test automatici. Gli smart contracts una volta pubblicati su Ethereum diventano immutabili, ai contratti verrà assegnato un certo indirizzo sulla blockchain e non sarà più possibile effettuare delle modifiche al loro codice a meno di caricarne altri sotto nuovi indirizzi.

Un altro aspetto importante riguarda i costi legati alla loro pubblicazione (*deploy*), le nuove versioni dell'applicazione possono essere pubblicate sotto nuovi indirizzi e ogni volta sarà necessario pagare, ex novo, il prezzo (Gas) calcolato per la loro pubblicazione. Un discorso analogo può essere fatto per i costi legati alle singole transazioni del contratto, l'obiettivo ideale è quello di creare una versione dell'applicazione ottimizzata e affidabile per i casi d'uso previsti dal programma e allo stesso tempo testata per prevenire comportamenti indesiderati.

In questa applicazione i programmi di testing sono stati divisi in tre parti (DigitalArchives_initialState.js, DigitalArchives_events.js e DigitalArchives_exceptions.js[19]). I programmi fanno uso della libreria di asserzioni Chai, nel primo gruppo viene controllato lo stato iniziale del contratto simulando una lo stato del contratto al momento della pubblicazione sulla blockchain. Il secondo gruppo verifica l'esecuzione degli eventi e i comportamenti previsti al cambiamento dello stato in seguito alle transazioni. Infine il terzo gruppo riunisce un certo numero di scenari non permessi (*edge cases*) che l'applicazione dovrebbe gestire lanciando delle eccezioni.

Di seguito nella figura 4.5 è mostrata l'esecuzione di questi test con framework Truffle.

```
1 >> truffle test
```

[18] Codice di tutti i contratti implementati, https://github.com/lukasd2/Digital-Archives-dApp/tree/master/contracts

[19] Codice dei test automatizzati, https://github.com/lukasd2/Digital-Archives-dApp/tree/master/test

Codice 4.4: Truffle esecuzione di test

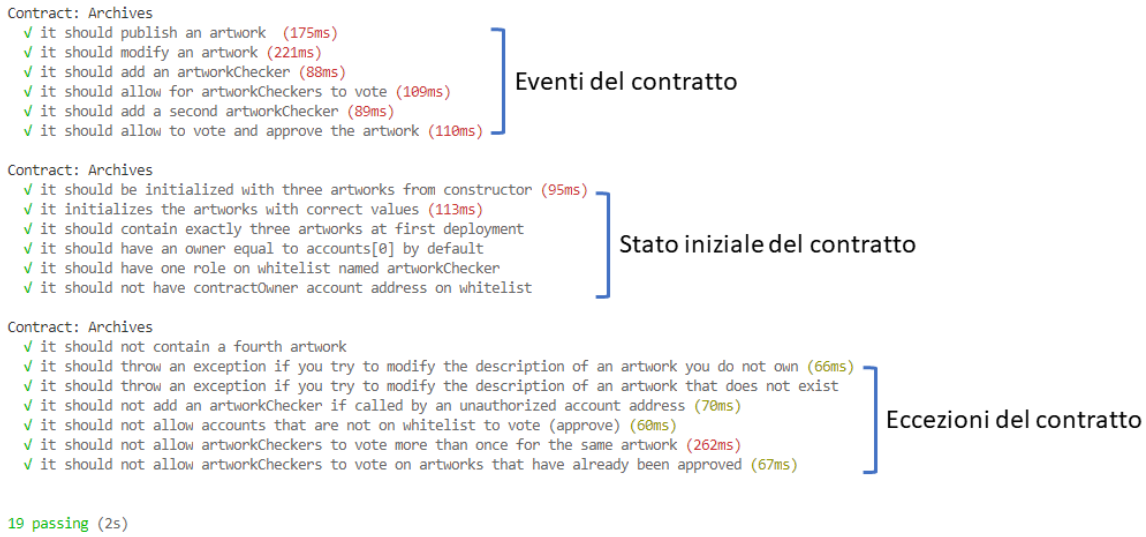

Figura 4.5: Progetto: Risultato esecuzione dei test

4.3.3 Applicazione client-side

Ultimo passaggio della prima fase di sviluppo[20] consiste nella programmazione di un'interfaccia client-side da collegare con i contratti testati.

Web3.js è una collezione di librerie che permettono di effettuare operazioni di lettura e scrittura sulla blockchain, così come avviene ad esempio con *Ajax* e i web server tradizionali. Fondamentalmente, a parte l'utilizzo di web3, il lato della programmazione front-end non subisce altri cambiamenti. Per le istruzioni di chiamata alla blockchain bisogna tenere conto dell'utilizzo delle funzioni asincrone e quindi gestire l'interfaccia e il flusso delle funzioni in modo adeguato per l'utente finale e per la gestione dei risultati in arrivo.

Per questo progetto si è scelto di costruire un'interfaccia minima con Bootstrap[21], in figura 4.6 si vede la parte relativa all'inserimento di un oggetto nella blockchain.

[20] Il processo di sviluppo del prototipo può essere consultato in questa repository, https://github.com/lukasd2/DigitalArchivesPrototype

[21] *Bootstrap Toolkit*. URL: `https://getbootstrap.com/`.

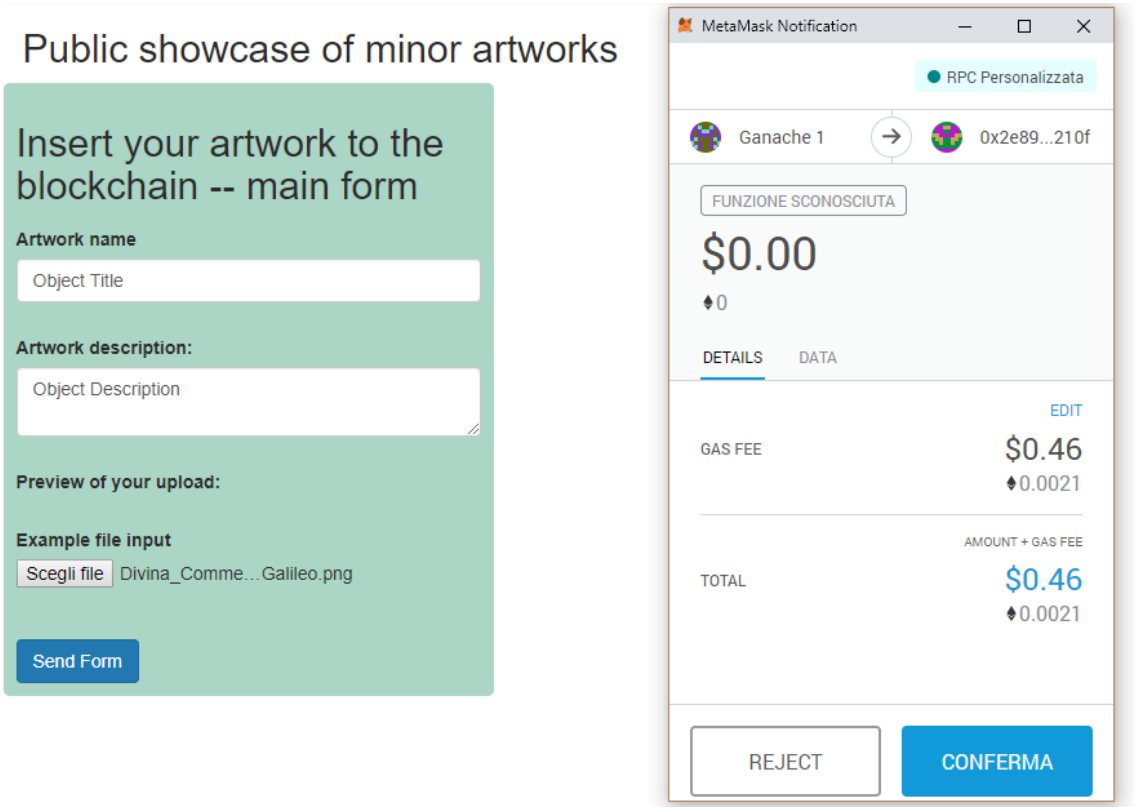

Figura 4.6: Progetto: Prototipazione inserimento oggetto sulla blockchain

A questo punto è possibile testare il funzionamento della dApp attraverso l'interfaccia grafica. Per quanto riguarda le transazioni, ovvero le chiamate ai contratti che richiedono lo svolgimento di calcoli computazionali da parte della EVM, esse saranno intercettate da Metamask. In questa fase potrebbe essere utile passare da una blockchain con un funzionamento semplificato, impostando un nodo blockchain (privato) vero e proprio con Geth per accertarsi del funzionamento corretto della dApp.

4.3.4 Integrazione con IPFS

Nelle dApp che richiedono la memorizzazione di una quantità rilevante di dati, che nelle blockchain è tra le operazioni più costose, diventa indispensabile integrare un servizio di storage esterno. In questo progetto si utilizza IPFS per memorizzare e richiamare i dati troppo grandi per essere mantenuti direttamente su Ethereum. La figura 4.7 rappresenta l'impostazione finale della logica dell'applicazione.

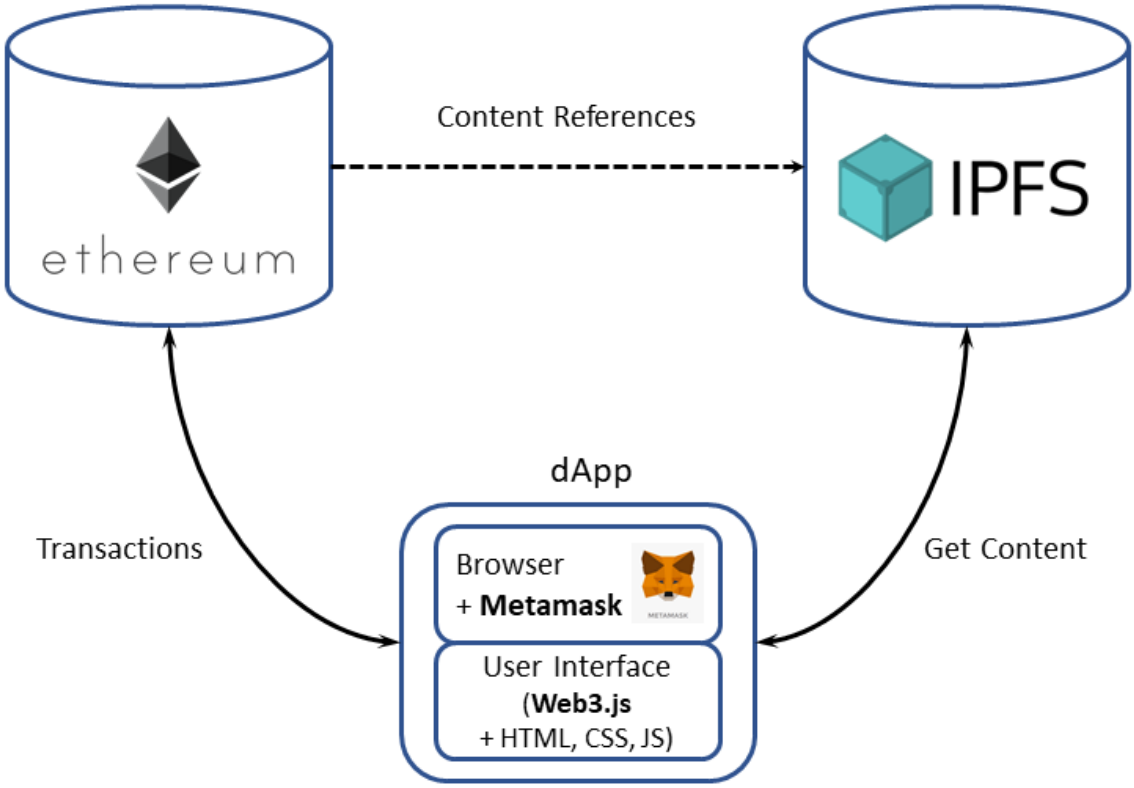

Figura 4.7: Progetto: Logica dell'applicazione finale

Seguendo questa impostazione la compilazione del modulo per l'inserimento di un nuovo oggetto include la spedizione delle risorse su IPFS. Prima di poter spedire la transazione sulla blockchain, la richiesta viene processata e rimane in attesa di upload sul servizio di storage. Se il caricamento avviene con successo, verrà restituito un indirizzo hash contenente i file caricati. A questo punto sarà possibile memorizzare il riferimento all'indirizzo IPFS sulla blockchain con una transazione. Ricapitolando, l'oggetto, la cui struttura è definita nel contratto, conterrà un puntatore all'indirizzo IPFS contenente la sua descrizione.

```
1 struct Artwork {
2     ...
3     string descriptionHash; //hash della descrizione dell'oggetto
4 }
```

Codice 4.5: Riferimento all'indirizzo hash di IPFS

In questo modo si sfrutta la caratteristica di IPFS dell'indirizzamento basato sul contenuto, per cui è matematicamente certo che sotto un indirizzo hash sarà possibile trovare un dato dal contenuto immutabile, e con il progredire della tecnologia, permanente. Riguardo alla permanenza, il sistema IPFS allo stato dell'arte non ha implementato un meccanismo di incentivi, per cui, in questa applicazione, la memorizzazione viene effettuata attraverso Infura[22], una API che agisce da intermediario, in modo da mantenere il contenuto accessibile senza bisogno di mantenere un nodo personale (IPFS) costantemente attivo. Dunque una chiamata generica di caricamento del contenuto su IPFS avrà la seguente struttura:

```
1 ipfs.files.add(data, [options], [callback])
```

Codice 4.6: Aggiunta di file con IPFS Javascript API

Una delle principali funzionalità dell'applicazione, l'inserimento dell'oggetto sulla blockchain, è composta dalla seguente catena di operazioni:

1. L'utente compila il modulo descrittivo dell'oggetto e preme il pulsante "Inserisci Oggetto"

2. Il programma cattura i file (in questa implementazione immagini) rappresentative dell'oggetto e attraverso un Buffer[23] li trasforma in una sequenza di dati binari

3. La sequenza di bit viene caricata su IPFS

4. IPFS ritorna un indirizzo hash per ciascuno dei file caricati che vengono memorizzati nella sessione in esecuzione del programma

5. Il programma costruisce una descrizione completa dell'oggetto utilizzando la versione del sistema di metadati scelta dall'utente e include gli indirizzi hash dei file caricati. Con un'altra operazione di caricamento su IPFS il tutto viene raggruppato sotto un unico indirizzo hash che viene restituito al programma in esecuzione

[22] *Infura API*. URL: `https://infura.io/`.

[23] https://www.npmjs.com/package/buffer

6. Il riferimento hash adesso può essere spedito e memorizzato sulla blockchain con una transazione effettuata e confermata dall'utente attraverso Metamask

7. La descrizione completa dell'oggetto viene richiamata utilizzando il riferimento memorizzato sulla blockchain che punta alla posizione contenuta su IPFS

4.3.5 Costruzione dell'interfaccia finale

Una volta implementata la logica completa dell'applicazione, seguita da una fase di test per verificarne la correttezza, è stata costruita l'interfaccia grafica finale. Per questa fase di programmazione *front-end* si è scelto di utilizzare Bulma[24] e Sass[25]. Bulma è un framework CSS basato su *flexbox* con un certo numero di elementi grafici predefiniti e un sistema di griglia che facilita la costruzione di layout flessibili, mentre Sass è un preprocessore CSS, serve principalmente per velocizzare e rendere più efficiente lo sviluppo dei fogli di stile[26]. In figura 4.8 è possibile vedere la pagina iniziale dell'applicazione.

Figura 4.8: Progetto: Pagina iniziale dell'applicazione

[24] *Bulma CSS framework*. URL: `https://bulma.io/`.

[25] *Sass - Syntactically awesome style sheets*. URL: `https://sass-lang.com/`.

[26] Sass estende i fogli di stile con diverse funzionalità tra cui la possibilità di utilizzare variabili, funzioni modulari (mixins), operazioni matematiche ecc.

Questa pagina serve a dare informazioni essenziali sulle funzionalità del progetto e i vantaggi derivanti dall'utilizzo della blockchain. Inoltre serve da guida per l'utilizzo delle dApps e si ricollega direttamente ai contenuti di questo elaborato.

La parte principale dell'applicazione, contente l'archivio, è illustrata in figura 4.9.

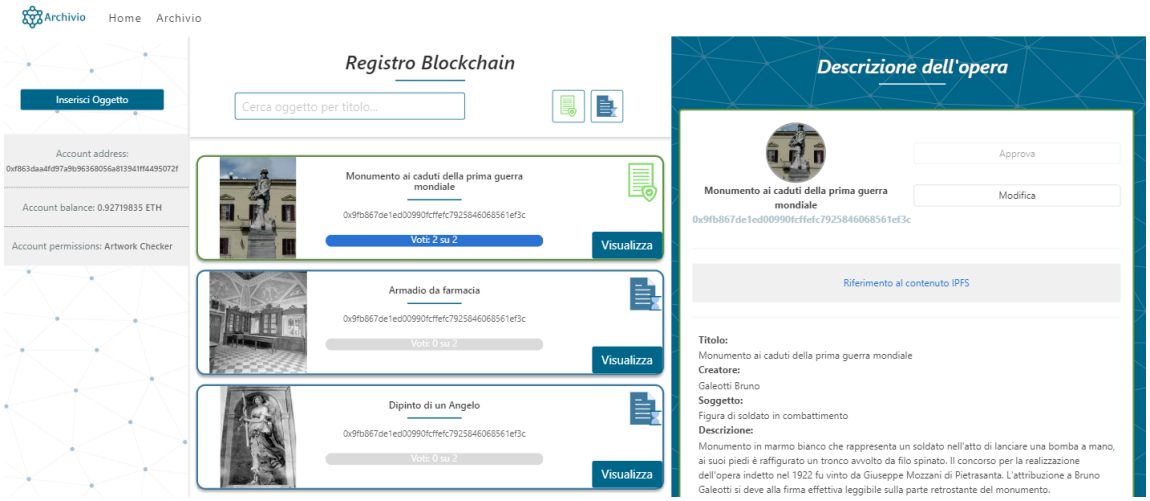

Figura 4.9: Progetto: Pagina contente l'archivio blockchain

Con questa impostazione dell'interfaccia finale l'applicazione è divisa logicamente, da sinistra verso destra, in tre parti. Nella parte sinistra è possibile attivare il modulo per l'inserimento di un oggetto nel registro e consultare le informazioni principali relative al proprio account (l'indirizzo utente, il bilancio di token e i permessi).

La parte centrale contiene l'archivio di tutte le opere registrate sulla blockchain e le informazioni principali relative a ciascun oggetto come illustrato di seguito nella figura 4.10. Inoltre permette di "filtrare" gli oggetti in base al loro stato di approvazione e di effettuare una ricerca per titolo tramite un input situato in alto a sinistra.

La parte a destra contiene la descrizione completa di ciascun oggetto inserito nell'archivio. In particolare i metadati e le immagini descrittive vengono richiamate da IPFS attraverso l'indirizzo hash memorizzato direttamente sulla blockchain. Una volta selezionato l'oggetto dell'archivio tramite il pulsante "visualizza", queste informazioni saranno raggruppate e compariranno in questa sezione del sito.

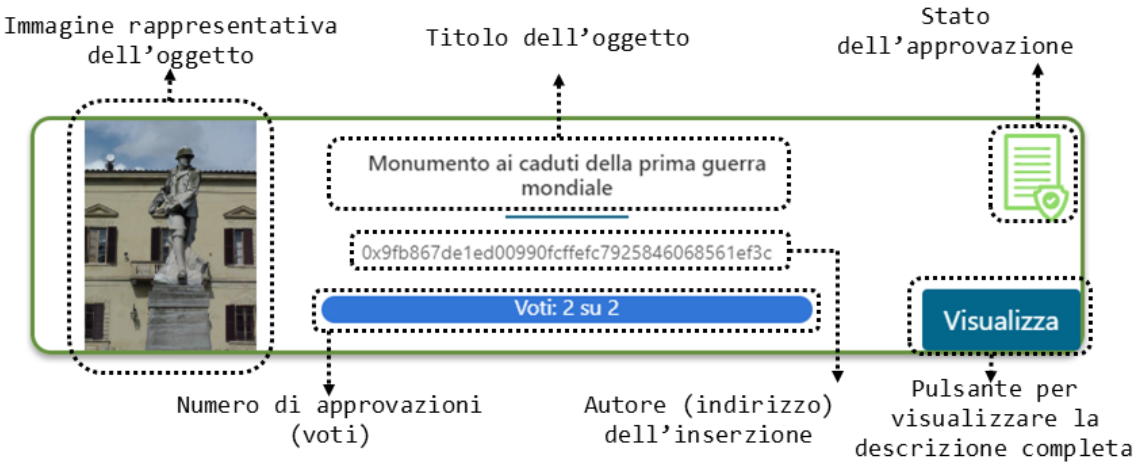

Figura 4.10: Progetto: Archivio la scheda di un oggetto

4.3.6 Ethereum Testnet

Il trasferimento dei contratti su una blockchain Ethereum di prova (*testnet*) conclude lo sviluppo di questo progetto. Ethereum ha diverse reti di sviluppo, comunemente chiamate testnet, utili allo scopo di provare il funzionamento dei contratti prima di farne il deploy sulla rete principale.

La differenza fondamentale tra queste reti e la rete principale (*mainnet*) deriva dall'utilizzo di token che non hanno un valore reale al di fuori dell'ambiente di prova, dunque è possibile creare le proprie applicazioni e pubblicarle eseguendo dei test sulla determinata blockchain di sviluppo evitando di dover sostenere spese legate alle transazioni.

In questo progetto si è scelto di pubblicare i contratti su Ropsten[27], una blockchain di prova che, al momento dello scrivere, utilizza lo stesso sistema di consenso (Proof of Work) della rete principale. È una rete in grado di riprodurre con un buon grado di precisione il funzionamento di mainnet, questo è particolarmente utile per verificare le stime relative ai costi delle transazioni e tutti gli aspetti legati al funzionamento della dApp stessa. Questa fase risulta fondamentale nel processo di sviluppo di un'applicazione decentralizzata ed è raccomandabile effettuarla con un numero di utenti il più ampio possibile per raccogliere i dati (*feedback*) circa il funzionamento di smart contracts e delle dApps

[27] *Ropsten - Ethereum Testnet*. URL: `https://ropsten.etherscan.io/`.

in modo da trovare eventuali errori prima di pubblicare l'applicazione sulla blockchain principale.

5. Conclusioni

Il presente elaborato ha illustrato un possibile percorso pensato come un'introduzione allo studio del paradigma blockchain seguito da una parte pratica relativa allo sviluppo di un progetto basato sulla tecnologia in questione.

Uno degli obiettivi principali di questo lavoro era quello di fornire un'organizzazione logica adatta per l'esposizione di un argomento che, generalmente, potrebbe essere giudicato come complesso. La divisione in tre strati - ciascuno dei quali corrispondente a un capitolo - ha permesso di dividere gli argomenti in sottoinsiemi di elementi costituenti. In definitiva, questi elementi costituiscono un percorso ordinato, introdotto con un approccio dal generale al particolare, adottato per massimizzare la chiarezza dell'esposizione. Una delle sfide nella stesura di questo elaborato è stata la ricerca di equilibrio tra il livello di dettaglio nella descrizione degli argomenti trattati e l'accessibilità di questi ultimi da parte di un lettore non esperto.

La realizzazione di questo elaborato è stata possibile anche grazie al tirocinio formativo effettuato presso l'Istituto di Informatica e Telematica del CNR di Pisa. Grazie a questa esperienza, che ha fornito una preparazione adeguata per poter affrontare l'argomento, è stato possibile proseguire con un'ulteriore ricerca sulla tecnologia finalizzata con la creazione di un progetto concreto.

In merito all'aspetto, prevalentemente teorico, relativo alla descrizione delle blockchain, si è dimostrato quali sono i principali vantaggi e gli svantaggi derivanti dall'utilizzo di tali soluzioni. Da una parte il paradigma offre la possibilità concreta di utilizzare la struttura di "*registri distribuiti*" (DLT) con i possibili vantaggi relativi alla decentralizzazione, alla trasparenza, all'immutabilità e alla permanenza dei dati. Dall'altra parte, queste caratteristiche, sono strettamente connesse con i costi inerenti alle transazioni e, rispetto ai sistemi tradizionali, con i tempi relativamente lunghi dovuti al funzionamento delle blockchain. Questi costi sono dovuti principalmente all'adozione dei sistemi di consenso e alla propagazione di una copia aggiornata del registro a tutti i partecipanti alla rete.

Durante l'analisi dello stato dell'arte è emerso anche l'aspetto dei problemi relativi

alla scalabilità. Nell'insieme, l'analisi degli aspetti potenzialmente innovativi correlati ai costi e alle problematiche delle attuali soluzioni implementative, dovrebbe permettere una valutazione più consapevole, in merito all'adozione della tecnologia, per i diversi casi d'uso. Tutto questo tenendo conto del fatto che si tratta di soluzioni tecnologiche in via di sviluppo che, una volta raggiunta una stabilità sufficiente, potranno essere applicate per la creazione di applicazioni reali potenzialmente innovative.

Uno degli aspetti innovativi su cui ci si è concentrati in questo lavoro deriva dall'espansione delle blockchain con i cosiddetti contratti intelligenti o *smart contracts*. L'unione di questi elementi dà luogo a un nuovo tipo di applicazioni, le cosiddette applicazioni decentralizzate. In particolare per la parte implementativa delle blockchain si è deciso di usare Ethereum, la piattaforma, al momento, più frequentemente usata per la costruzione di questo tipo di applicazioni.

Per quanto riguarda la creazione del progetto che accompagna questo lavoro, l'obiettivo principale era quello di dimostrare la metodologia, insieme alle *best practices*, con cui è possibile costruire le applicazioni decentralizzate. Il risultato è un'applicazione basata sulla blockchain, per la parte *back-end*, e su IPFS per la parte relativa alla memorizzazione dei dati.

Grazie a questo progetto è stato possibile applicare concretamente la tecnologia allo stato dell'arte, all'ambito delle scienze umane. Si tratta dunque di un impiego possibile di tale soluzione tecnologica nell'ottica della salvaguardia dei beni culturali. In particolare si è scelto di sviluppare una soluzione informatica che permette di catalogare le opere d'arte definibili come minori. Con questo esempio è stato possibile verificare il funzionamento della blockchain come un'alternativa valida alla risoluzione dei problemi più comuni presenti nei sistemi centralizzati.

Il progetto ha illustrato la metodologia, un'eventuale applicazione reale di questo tipo, al momento è limitata sopratutto dallo stato dell'arte della tecnologia. Si tratta comunque di una soluzione pubblicata su una blockchain di prova, il passo successivo potrebbe consistere nell'aggiornamento dell'applicazione con l'evolversi della tecnologia e nella raccolta di *feedback* da parte di utenti per poter espandere l'applicazione con nuove funzionalità e per verificare ulteriormente la correttezza delle soluzioni attualmente adottate.

Inoltre potrebbe essere importante considerare l'argomento di questa tesi fondamentalmente per qualsiasi applicazione di tipo informatico, ma soprattutto per la figura dell'informatico umanista, per i numerosi campi applicativi legati al trattamento di contenuti culturali. Lo studio di tale tecnologia potrebbe costituire un vantaggio importante per il mondo del lavoro e per affrontare lo studio delle soluzioni di molti problemi presenti nella società dell'informazione attuale.

In conclusione, il presente elaborato non esaurisce l'argomento ma potrebbe essere una buona base di partenza per lo studio più approfondito dei diversi elementi che ne fanno parte. Le blockchain sono una tecnologia promettente, in via di sviluppo, sempre più da tenere a mente nella costruzione di nuove soluzioni informatiche.

Bibliografia

[1] UK Government Chief Scientific Adviser. *Distributed Ledger Technology: beyond block chain*. Government Office for Science, 2016. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

[2] Alex de Vries. «Bitcoin's Growing Energy Problem». In: (5.2018). DOI: https://doi.org/10.1016/j.joule.2018.04.0165. URL: https://www.cell.com/joule/fulltext/S2542-4351(18)30177-6. (Visitato: 10.11.2018).

[3] *BigchainDB - Blockchain Database*. URL: https://www.bigchaindb.com/.

[4] *Bitcoin: Attacchi e prevenzione della doppia spesa*. URL: https://en.bitcoin.it/wiki/Irreversible_Transactions. (Visitato: 10.11.2018).

[5] *Bitcoin: Attacco della maggiornaza (majority attack)*. URL: https://en.bitcoin.it/wiki/Majority_attack. (Visitato: 10.11.2018).

[6] *Bitcoin, blocco genesis*. URL: https://blockexplorer.com/block/000000000019d6689c085

[7] *Bitcoin: Tentativi di modifica della storia della rete*. URL: https://en.bitcoin.it/wiki/Weaknesses#Attacker_has_a_lot_of_computing_power. (Visitato: 10.11.2018).

[8] *Bootstrap Toolkit*. URL: https://getbootstrap.com/.

[9] *Bulma CSS framework*. URL: https://bulma.io/.

[10] Vitalik Buterin. *The Meaning of Decentralization*. URL: https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274. (Visitato: 20.08.2018).

[11] *Calcolo distribuito - distributed computing*. URL: https://en.wikipedia.org/wiki/Distributed_computing. (Visitato: 10.11.2018).

[12] *Censura nelle blockchain*. URL: https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/.

[13] *Chai - assertion library*. URL: https://www.chaijs.com/.

[14] *CIDOC CRM - Conceptual Reference Model*. URL: http://www.cidoc-crm.org/.

[15] *Cloud Computing - Servizi*. URL: https://en.wikipedia.org/wiki/Cloud_computing#Service_models. (Visitato: 10.11.2018).

[16] *Crittografia Asimmetrica - Asymmetric Encryption*. URL: https://en.wikipedia.org/wiki/Public-key_cryptography.

[17] *DHT - Distributed Hash Table*. URL: https://en.wikipedia.org/wiki/Distributed_hash_table. (Visitato: 10.11.2018).

[18] *Dublin Core Metadata Initiative*. URL: http://dublincore.org/.

[19] *ECMAScript® 2015 Language Specification*. URL: http://www.ecma-international.org/ecma-262/6.0/index.html. (Visitato: 10.11.2018).

[20] *Esempio del calcolo di SHA256 - Generatore Hash*. URL: https://passwordsgenerator.net/sha256-hash-generator/.

[21] *Ether conversioni*. URL: http://ethdocs.org/en/latest/ether.html.

[22] *Ethereum, blocco genesis*. URL: https://etherscan.io/block/0f.

[23] *Ethereum Blockchain App Platform*. URL: https://www.ethereum.org/.

[24] *EVM - Lista opcodes*. URL: https://ethervm.io/#opcodes.

[25] *Filecoin - Decentralized Storage Network*. URL: https://filecoin.io/.

[26] *Firme digitali - Digital signature*. URL: https://en.wikipedia.org/wiki/Digital_signature.

[27] *Gartner - Hype Cycle for Emerging Technologies, 2016*. URL: https://www.gartner.com/newsroom/id/3412017.

[28] *Gartner - Hype Cycle for Emerging Technologies, 2018*. URL: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/.

[29] *Geth - Ethereum implementation*. URL: https://geth.ethereum.org/.

[30] Stuart Haber e W. Scott Stornetta. ≪How to Time-stamp a Digital Document≫. In: *Journal of Cryptology* 3 (1991), pp. 99–111.

[31] *Infura API*. URL: https://infura.io/.

[32] *IPFS - Distributed Web*. URL: https://ipfs.io/.

[33] *IPFS - Whitepaper (Draft 3)*. URL: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQ
ipfs.draft3.pdf.

[34] *Java Virtual Machine*. URL: https://en.wikipedia.org/wiki/Java_virtual_machine. (Visitato: 10.11.2018).

[35] *Kickstarter - servizio di finanziamento collettivo*. URL: https://en.wikipedia.org/wiki/Kickstarter. (Visitato: 10.11.2018).

[36] Leslie Lamport, Robert Shostak e Marshall Pease. ≪The Byzantine Generals Problem≫. In: *ACM Transactions on Programming Languages and Systems* 4/3 (1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[37] *Metamask*. URL: https://metamask.io/.

[38] *Mocha - test framework*. URL: https://mochajs.org/.

[39] *Modulo Object ID - Comando Carabinieri Tutela Patrimonio Culturale*. URL: http://tpcweb.carabinieri.it/SitoPubblico/objectId.

[40] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf*. 2008.

[41] *Open Zeppelin*. URL: https://openzeppelin.org/.

[42] Philip Boucher, Susana Nascimento, Mihalis Kritikos. «How blockchain technology could change our lives». In: (2018). DOI: 10.2861/926645. URL: https://publications.europa.eu/en/publication-detail/-/publication/9964fbfd-6141-11e7-8dc1-01aa75ed71a1.

[43] *Remix - Solidity IDE*. URL: https://remix.ethereum.org/.

[44] *Ripple: Blockchain*. URL: https://ripple.com/.

[45] *Ripple: Strategia di decentralizzazione*. URL: https://ripple.com/dev-blog/decentralization-strategy-update/.

[46] *Ropsten - Ethereum Testnet*. URL: https://ropsten.etherscan.io/.

[47] *Sandbox - Sicurezza informatica*. URL: https://en.wikipedia.org/wiki/Sandbox_(computer_security). (Visitato: 10.11.2018).

[48] *Sass - Syntactically awesome style sheets*. URL: https://sass-lang.com/.

[49] Pasquale Savino. *Corso Biblioteche Digitali*. URL: http://www.nmis.isti.cnr.it/savino/Dispensa/metadati.pdf.

[50] *SHA2 - Funzioni di Hash*. URL: https://en.wikipedia.org/wiki/SHA-2. (Visitato: 10.11.2018).

[51] *SHA3 - Funzioni di Hash*. URL: https://en.wikipedia.org/wiki/SHA-3. (Visitato: 10.11.2018).

[52] *Sia - Decentralized Cloud Storage*. URL: https://sia.tech/.

[53] *Solidity - linguaggio di programmazione di smart contracts*. URL: https://solidity.readthedocs.io.

[54] *Solidity - mapping*. URL: https://solidity.readthedocs.io/en/v0.4.21/types.html#mappings.

[55] *Storj - Decentralized Cloud Storage*. URL: https://storj.io/.

[56] *Swarm - Distributed Storage Platform*. URL: https://swarm-guide.readthedocs.io/en/latest/introduction.html.

[57] Nick Szabo. «Contracts with Bearer». In: (1997). URL: https://nakamotoinstitute.org/contracts-with-bearer/.

[58] Nick Szabo. «Smart Contracts». In: (1994). URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[59] *Teoria dei giochi - Game theory*. URL: https://en.wikipedia.org/wiki/Game_theory.

[60] *Throughput - Visa*. URL: https://usa.visa.com/run-your-business/small-business-tools/retail.html.

[61] *Trilemma scalability*. URL: https://github.com/ethereum/wiki/wiki/Sharding-FAQs#this-sounds-like-theres-some-kind-of-scalability-trilemma-at-play-what-is-this-trilemma-and-can-we-break-through-it.

[62] *Truffle Suite*. URL: https://truffleframework.com/.

[63] *Truffle Suite - Ganache*. URL: https://truffleframework.com/ganache.

[64] *Web3.js*. URL: https://github.com/ethereum/web3.js/.

[65] *Whitepaper Ethereum - Merkle Trees*. URL: https://github.com/ethereum/wiki/wiki/White-Paper#merkle-trees.

[66] *Whitepaper Ethereum - transazioni*. URL: https://github.com/ethereum/wiki/wiki/White-Paper#messages-and-transactions.

[67] *Whitepaper Ripple - sistema di consenso*. URL: https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[68] *Yellowpaper Ethereum*. URL: https://ethereum.github.io/yellowpaper/paper.pdf.