Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria informatica

Data Self-Sovereignty e Decentralizzazione: Implementazione di un'architettura basata su IOTA e IPFS

Elaborato in
BLOCKCHAIN AND CRYPTOCURRENCIES

Relatore
Chiar.mo Prof.
FERRETTI STEFANO

Presentata da
PAGLIA FRANCESCO

Anno Accademico 2022-2023

Indice

Introduzione

Nell'ultimo decennio la produzione, lo scambio e l'analisi dei dati sono operazioni diventate parte integrante del processo decisionale per le politiche pubbliche, la gestione aziendale e la ricerca scientifica. Questa rivoluzione ha sancito la nascita di una società data-driven, ovvero guidata dai dati. Ciò è stato reso possibile grazie alla realizzazione di dispositivi di storage sempre più economici e capienti, all'ottimizzazione degli algoritmi di machine learning in grado di elaborare grandi quantità di dati e allo sviluppo di applicazioni che fanno largo uso delle informazioni fornite dai loro utenti. Questi ultimi ricoprono un ruolo centrale nella produzione dei dati, però sono spesso assenti nelle fasi di analisi e monetizzazione delle informazioni raccolte. La loro esclusione va a beneficio soprattutto delle aziende basate sui modelli di business DaaS (Data as a Service) o IaaS (Information as a Service), ovvero sulla vendita di dati grezzi e aggregati o di informazioni estrapolate dall'analisi dei dati. La colpa non è da attribuire solo alle aziende, ma anche agli utenti che usano le applicazioni senza interessarsi al ciclo di vita delle informazioni che forniscono.

La Distributed Ledger Technology (DLT) rappresenta la tecnologia giusta per innovare questo modus operandi, decentralizzando la gestione dei dati per accrescere il potere decisionale degli utenti produttori. L'obiettivo principale di una DLT è quello di spostare la fiducia dall'intermediario che gestisce una transazione tra due parti a un protocollo che esegue la transazione senza la necessità di una terza parte fidata. Tra le varie tipologie di DLT disponibili, negli ultimi anni si è distinto il progetto IOTA. Quest'ultimo nasce con

l'intento di rivoluzionare la tecnologia dei registri distribuiti, impiegando una struttura a grafo aciclico (Tangle) al posto della classica catena di blocchi su cui si basano le blockchain. Grazie a questo cambiamento, IOTA è riuscita a sviluppare un protocollo che consente il trasferimento di dati e di valore in maniera scalabile e gratuita. Inoltre, gli ultimi aggiornamenti hanno introdotto anche la possibilità di distribuire sulla rete gli smart contract, utili per elaborare qualsiasi tipo di calcolo distribuito.

Lo scopo di questa tesi è utilizzare la DLT per implementare un ecosistema in grado di gestire i dati degli utenti produttori e le applicazioni che li raccolgono ed elaborano. Pertanto, l'ecosistema deve essere composto principalmente da due parti: store di applicazioni e gestore delle autorizzazioni. La prima parte comprende tutte le funzionalità necessarie per pubblicare, modificare, esplorare e scaricare le applicazioni. Invece, la seconda parte è dedicata alle operazioni sui dati, eseguite dalle app, e alla gestione delle autorizzazioni che gli utenti produttori possono concedere o revocare ai consumatori. Per garantire trasparenza, ogni operazione sui dati deve essere descritta dettagliatamente e il suo codice deve essere pubblicamente accessibile. I produttori possono modificare le autorizzazioni concesse in qualsiasi momento. I dati prodotti vengono cifrati, autenticati e archiviati, in base alla loro dimensione, sul Tangle o sul file system IPFS. L'accesso è mediato dagli smart contract, i quali conservano tutti i riferimenti utili per recuperare i dati. Inoltre, gli smart contract si occupano persino dell'autenticazione degli utenti mediante le identità decentralizzate. In conclusione, l'ecosistema è pensato per rivoluzionare la raccolta e l'elaborazione dei dati prodotti dalle applicazioni, fornendo strumenti utili sia agli sviluppatori sia agli utenti che vogliono gestire in maniera più minuziosa e consapevole i propri dati.

Capitolo 1

Background

La blockchain e le applicazioni basate su un'infrastruttura a registri distribuiti propongono nuovi paradigmi che rivoluzionano ampiamente l'attuale sistema economico, modificando radicalmente i concetti di transazione, proprietà e fiducia. Inoltre, facilitano la realizzazione di un nuovo sistema di validazione degli eventi, degli scambi e dei processi, basato sul raggiungimento di un consenso distribuito fra più parti.

All'interno di questo nuovo mondo di possibilità, sono nati tantissimi progetti innovativi, tra cui IOTA. Quest'ultimo si serve della tecnologia dei registri distribuiti per agevolare lo scambio sicuro di valore e dati tra persone e macchine, senza pagare alcuna commissione. Negli ultimi anni, il progetto IOTA è cresciuto tantissimo e oggi vanta molte collaborazioni con aziende importanti e numerosi sviluppatori impegnati nella crescita continua del progetto.

1.1 Distributed Ledger Technology

La Distributed Ledger Technology (DLT) è una tecnologia rivoluzionaria che consente di archiviare, condividere e distribuire in modo sicuro i dati su più dispositivi. Inoltre, consente l'aggiornamento in tempo reale dei record digitali senza la necessità di un organo di controllo centralizzato. Infine, è

pensata per eseguire le transazioni in maniera trasparente e garantire, tramite l'impiego di funzioni crittografiche avanzate, immutabilità e sicurezza per tutti i dati che gestisce.

Oggigiorno, la DLT viene impiegata in molti settori: finanza, sanità, gestione della catena di fornitura, commercio di energia e verifica dell'identità digitale. Questa tecnologia viene scelta per semplificare la creazione di un registro decentralizzato affidabile, preciso e sicuro [1].

1.1.1 Funzionamento

La DLT è a tutti gli effetti un registro distribuito su una rete di computer e database, in cui vengono salvate le transazioni processate. Tutti i partecipanti della rete hanno una copia del registro che aggiornano ogni volta che arriva un nuovo dato da memorizzare.

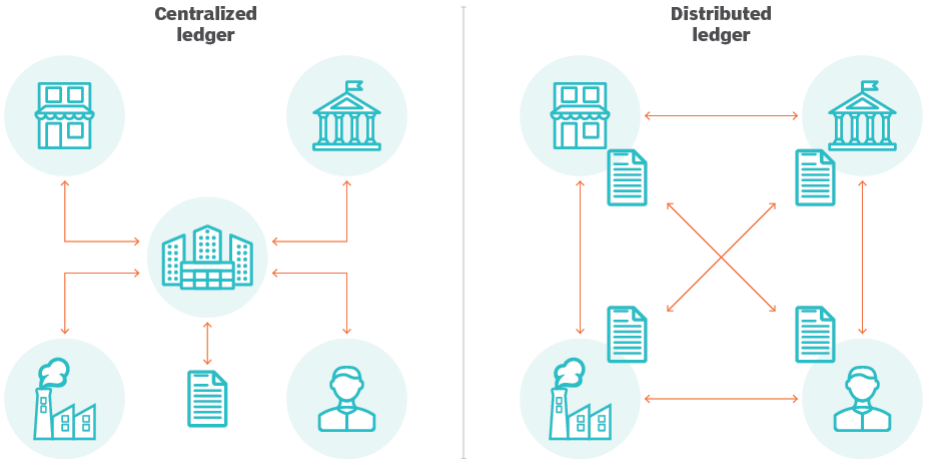

Figura 1.1: Rappresentazione della differenza tra un sistema centralizzato e un sistema DLT [2]

Non appena un nodo riceve una nuova transazione, questa viene salvata nel suo registro locale. La modifica locale deve però riflettersi anche nei registri locali degli altri nodi. Di conseguenza, sfruttando dei protocolli di comunicazione peer-to-peer, la nuova transazione viene comunicata dal nodo

di partenza verso tutti gli altri nodi della rete. Solo se tutti i nodi condividono la stessa versione del registro è possibile garantire che i dati non vengano danneggiati o manomessi localmente, perché questo tipo di modifica verrebbe subito individuata e scartata.

Quindi, la natura distribuita della DLT la rende altamente sicura e resistente agli attacchi. Questi ultimi potrebbero avere successo solo se un hacker riuscisse ad acquisire il controllo contemporaneo della maggior parte dei nodi e riuscisse a manipolare i loro dati simultaneamente: scenario praticamente impossibile.

1.1.2 Permissionless e permissioned

In base alle diverse modalità con cui è possibile accedere al registro distribuito, è possibile distinguere due differenti tipologie di Distributed Ledger:

- **Permissionless:** senza chiedere il permesso a nessuno, chiunque può partecipare al processo di validazione delle transazioni scritte sul registro e leggere e scrivere le informazioni associate.
- **Permissioned:** in questo caso esiste una terza parte fidata che è incaricata di assegnare specifici permessi a chi può leggere, modificare e validare i dati scritti sul registro distribuito. I nodi che intendono validare le transazioni vengono preselezionati e devono essere riconoscibili da tutti, tramite l'assegnazione di un'identità. In questo tipo di sistemi, la correttezza della validazione è fortemente basata sulla fiducia che si ripone nei nodi validatori.

In entrambe le tecnologie, i nodi restano connessi tra di loro in una rete peer-to-peer e gestiscono un registro distribuito governato da una logica decentralizzata.

1.1.3 Caratteristiche

Le principali caratteristiche della DLT sono:

- **Decentralizzazione:** non fa affidamento su nessuna autorità centrale o terza parte fidata. Questo significa che è resistente alla manipolazione e al controllo esterno. Tutti i partecipanti alla rete sono connessi in una rete peer-to-peer, hanno egual importanza e accedono agli stessi dati.
- **Sicurezza:** garantisce la sicurezza e l'immutabilità dei dati tramite l'impiego di algoritmi crittografici avanzati. Questo la aiuta a proteggersi da attacchi esterni, come la manipolazione dei dati nel registro.
- **Smart contract:** supporta gli smart contract, ovvero i programmi informatici auto eseguibili che aiutano ad automatizzare l'elaborazione delle transazioni e a garantire che tutte le parti coinvolte rispettino i termini e le condizioni dell'accordo.
- **Trasparenza:** consente a tutti i partecipanti della rete di visualizzare il contenuto del registro in un dato momento. Questa possibilità di controllo e tracciabilità continua favorisce la riduzione dei tentativi di frode e di corruzione.
- **Efficienza:** elimina gli intermediari e riduce i tempi di processamento. Questo velocizza l'elaborazione delle transazioni e riduce i costi.

L'adozione della DLT porta con sé diversi vantaggi: decentralizzazione, scalabilità, trasparenza, risparmio di tempo, riduzione dei costi e anonimato. Naturalmente, ci sono anche degli svantaggi: complessità, mancanza di regolamentazioni, alto consumo energetico e problemi di privacy. Nonostante ciò, la DLT ha riscosso molto successo grazie alla rivoluzione che ha portato nel modo in cui i dati vengono condivisi, archiviati e trattati.

1.2 IOTA

IOTA è stata fondata nel 2015 da David Sønstebø, Sergey Ivancheglo, Dominik Schiener e Dr. Serguei Popov. Questo progetto nasce con l'intento di creare un ecosistema completo, basato principalmente su due componenti: il Tangle e la criptovaluta IOTA.

Figura 1.2: Logo di IOTA [4]

Grazie al Tangle è possibile implementare una DLT (Distributed Ledger Technology) permissionless. Questo ci consente di controllare i nostri dati privati, eseguire programmi con cui nessuno può interferire, validare le transazioni scritte sul registro, commerciare e possedere beni digitali senza intermediari.

IOTA nasce soprattutto per gestire i dati provenienti dal mondo dei dispositivi IoT (Internet of Things), che producono periodicamente grandi quantità di informazioni che non sarebbe stato possibile controllare tramite le classiche blockchain. Infatti, IOTA è stata progettata proprio per svolgere un ruolo centrale nella prossima rivoluzione industriale, consentendo relazioni economiche tra le macchine e favorendo la cosiddetta Machine Economy. Inoltre, IOTA favorisce lo scambio di dati e di valore tra persone e macchine, rendendola perfetta persino per l'Internet of Everything [3].
Tra i principali vantaggi di IOTA troviamo:

- **Elevata scalabilità:** grazie al Tangle che consente l'aggiunta di transazioni parallelamente, a differenza della blockchain in cui vengono

aggiunte sequenzialmente.

- **Basso fabbisogno energetico:** pensato per i dispositivi, come i sensori, che consumano poca energia.
- **Transazioni gratuite:** nessuna fee (commissione) per elaborare o effettuare le transazioni.
- **Transazioni veloci:** le transazioni vengono confermate in pochi minuti.
- **Approvazione in pochi secondi:** i messaggi onesti e corretti sono approvati velocemente ed efficientemente.
- **Distribuzione:** rete distribuita globalmente, resiliente e resistente agli attacchi.

1.2.1 Tangle

Il Tangle è una struttura dati che forma un grafo aciclico diretto (Direct Acyclic Graph) composto da transazioni (blocco-DAG), dove ogni nuova transazione si collega a un insieme di transazioni passate. Le funzioni svolte dal Tangle ricordano quelle della blockchain, ma fra di loro ci sono delle importanti differenze in termini di scalabilità e performance [5].

La blockchain è rappresentata come una catena crescente di blocchi, legati fra di loro sfruttando la crittografia. Le transazioni diventano parte del registro solo quando vengono incluse in nuovi blocchi. Nel caso in cui tutte le nuove transazioni non riescano a rientrare in un singolo blocco, alcune di queste dovranno essere inserite nei prossimi blocchi. Inoltre, i minatori favoriscono l'inclusione di quelle transazioni per le quali gli utenti sono disposti a pagare delle fee maggiori. Accelerare la creazione dei blocchi e incrementare la loro dimensione non risolverebbe il problema, anzi potrebbe comprometterne la sicurezza. In altre parole, il throughput del sistema deve essere artificialmente soppresso in maniera tale da avere la garanzia che ogni

blocco appena inserito sia propagato a tutti i nodi della rete, prima di creare il successivo blocco. Questo modus operandi crea un collo di bottiglia che limita le performance e spreca tutti gli sforzi computazionali ed energetici impiegati per il PoW dei blocchi che dovranno essere scartati. Inoltre, la blockchain si trova a fronteggiare anche altri problemi relativi ai minatori e ai validatori: gare di mining, centralizzazione, decrescente profitto per i minatori (miner extractable value) e fee troppo alte per gli utenti.

Figura 1.3: Rappresentazione del collo di bottiglia della blockchain [6]

Il Tangle punta a superare le limitazioni della blockchain. Esso si basa su un nuovo protocollo che permette la validazione parallela delle transazioni tramite un consenso probabilistico senza la necessità di un ordinamento totale e senza un leader. Nel Tangle non abbiamo più il concetto di blocco che contiene più transazioni, ma ogni singola transazione viene direttamente attaccata al grafo. Ciò consente l'eliminazione di intermediari come minatori e validatori. Questi ruoli saranno svolti direttamente dal client che vorrà aggiungere una nuova transazione al Tangle, collegandola a delle transazioni passate che saranno automaticamente validate.

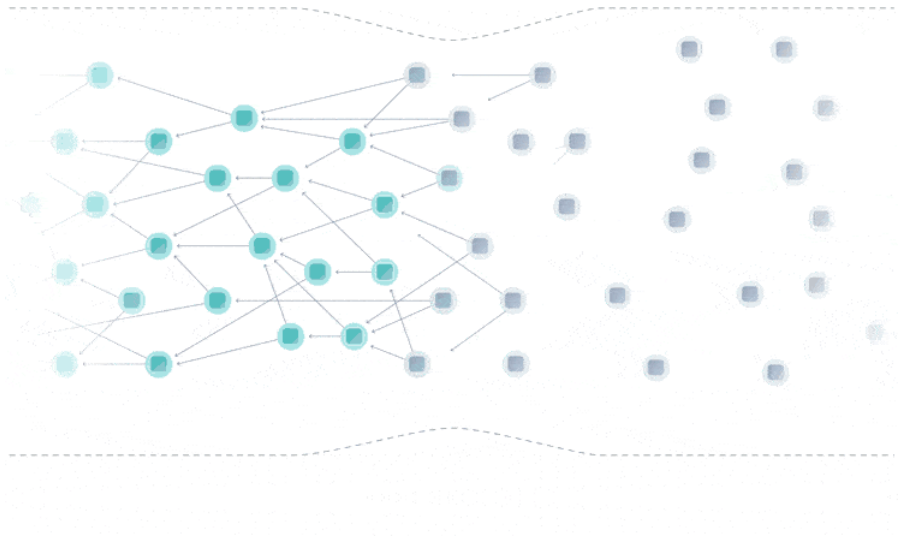

Figura 1.4: Rappresentazione del funzionamento parallelo del Tangle [7]

Tutte queste caratteristiche, unite all'elevata parallelizzazione e alla capacità di lavorare in un ambiente asincrono, consentono di ottenere una struttura dati performante, scalabile e dai consumi ridotti.

1.2.2 Protocollo IOTA

Il protocollo IOTA è ancora in fase di ricerca. Al momento ha due reti pubbliche: la rete principale, che è stabile e gestisce tutti i token IOTA; e la rete di staging, utile per testare gli ultimi aggiornamenti del protocollo. La rete principale prende il nome di IOTA Mainnet; mentre quella di staging si chiama Shimmer.

Attualmente, sulla rete principale è attiva la versione 1.5 del protocollo IOTA, meglio conosciuta come Chrysalis. Nel frattempo, su Shimmer è in fase di test la successiva versione del protocollo chiamata Stardust [8].

Reti	Mainnet	Shimmer
Protocollo	Chrysalis	Stardust
Stabilità	Eccellente	Buona
Fee per transazioni	Nulle	Nulle
Token nativo	IOTA	SMR
Struttura dati	DAG (Tangle)	DAG (Tangle)
Tokenizzazione	No	NFT, token nativi e output alias
Supporto Smart Contract	No	In fase alpha

Tutti i miglioramenti che si susseguono nelle varie versioni del protocollo IOTA vengono conservati in un archivio pubblico ed etichettati con il prefisso TIP (Tangle Improvement Proposal). L'obiettivo finale della Fondazione IOTA è quello di puntare alla versione 2.0 del protocollo, ovvero a Coordicide che supporta pienamente la decentralizzazione della rete [9].

1.2.2.1 Chrysalis

Per IOTA, Chrysalis rappresenta la fase intermedia della rete principale prima del completamento di Coordicide. Lo scopo di Chrysalis è migliorare l'usabilità della precedente IOTA Mainnet, soprattutto per gli sviluppatori [10].
Tra i miglioramenti proposti da questa versione troviamo:

- **Ordinamento White-flag (TIP 0002):** forza l'ordinamento deterministico del Tangle per far sì che le milestone confermino solamente i messaggi che non sono in conflitto fra di loro. Ignorando i conflitti, è possibile incrementare la velocità e l'efficienza della selezione delle tip.

Inoltre, è possibile persino evitare diversi attacchi informatici, tra cui il noto attacco di conflict spamming [11].

- **Nuovo algoritmo di selezione delle milestone (TIP 0004):** il Coordinatore potrà confermare più transazioni al secondo (Confirmed Transaction Per Second - CTPS). Il tempo di conferma scende da 2 minuti a 10 secondi. Questo miglioramento comporta una maggiore efficienza computazionale [12].

- **Nuovo algoritmo per la selezione delle tip (TIP 0003):** prende il nome di Uniform Random Tip Selection (URTS) e verrà approfondito in seguito [13].

- **Schema di firma Ed25519 (TIPS 0014):** sostituisce il precedente schema di firma, Winternitz One Time Signature (W-OTS). L'utilizzo di uno schema di firma EdDSA consente al protocollo e ai client che lo usano di funzionare più efficientemente su hardware meno prestanti. A differenza di W-OTS, Ed25519 supporta anche il riutilizzo delle chiavi private, consentendo il riutilizzo degli indirizzi nel protocollo. Questa modifica riduce persino drasticamente le dimensioni della transazione, risparmiando larghezza di banda e riducendo i tempi di elaborazione [14].

- **Atomicità delle transazioni (TIP 0006):** con l'introduzione delle transazioni atomiche, è possibile gestire transazioni semplici senza l'utilizzo dei vecchi bundle. Ciò si traduce in un codice meno complesso, più adattabile e gestibile per il software di base. Inoltre, le transazioni atomiche riducono l'overhead della rete, il carico di convalida delle transazioni e di verifica delle firme. Di conseguenza, diventa meno oneroso fronteggiare gli spam e controllare le congestioni [15].

- **Passaggio al Modello UTXO (TIP 0007):** ogni moneta su un indirizzo diventa univocamente identificabile e in ogni spesa diventa possibile riconoscere le monete che si vogliono scambiare. Ciò consente

una gestione dei conflitti più rapida e precisa, migliorando la resilienza e la sicurezza del protocollo. Inoltre, il passaggio a UTXO pone le fondamenta per funzionalità future come, ad esempio, i token colorati [16].

- **Rappresentazione binaria interna della transazione ternaria (TIP 0005):** questo consente di lavorare su dati binari per la convalida, l'IO e altre elaborazioni, senza affidarsi sulle conversioni binario-ternario che avvenivano sui nodi prima di Chrysalis. Il passaggio alle transazioni binarie riduce ulteriormente le dimensione delle transazioni, risparmiando larghezza di banda e riducendo i tempi di elaborazione [17].

Chrysalis è disponibile su due reti:

- **Mainnet:** rete IOTA che opera sui token IOTA scambiati tramite gli exchange. Questa rete è più stabile.
- **Devnet:** rete IOTA pensata per gli sviluppatori, dove i token IOTA sono gratuiti. Questa rete è meno stabile.

Per interagire con queste due reti pubbliche, la Fondazione IOTA mette a disposizione degli endpoint che svolgono anche il ruolo di load-balancer tra i vari nodi facenti parte delle reti.

In conclusione, possiamo affermare che Chrysalis è stata la serie più promettente di aggiornamenti mai apportati a IOTA. È stato un passo importante per la preparazione alla produzione, con un aumento del throughput delle transazioni, stabilità della rete, usabilità migliore, nuove funzionalità e casi d'uso abilitati. Con il rilascio di Chrysalis, IOTA è sulla buona strada per diventare la tecnologia più consona per l'IoT e per tante altre applicazioni.

1.2.2.2 Stardust

Il protocollo Chrysalis è ottimizzato per una singola applicazione: scambiare denaro digitale [18]. L'imminente aggiornamento Stardust introduce nuove importanti feature:

- Rendere IOTA un'infrastruttura che supporta un secondo layer su cui eseguire gli IOTA Smart Contracts (ISC).
- Trasformare IOTA in un registro multi-asset in grado di gestire token personalizzati e definiti dagli utenti tramite il nuovo framework di tokenizzazione.

Gli IOTA Smart Contracts possono essere implementati sul Layer 2 (L2), a differenza di tanti altri progetti che offrono gli smart contract solo nel Layer 1 (L1). Questa scelta implementativa porta con sé dei vantaggi ma anche degli svantaggi:

- **Vantaggi**
  - L1 non congestionato dagli smart contract.
  - Il numero di transazioni elaborate al secondo nel L1 non è limitato dalla velocità della VM su cui esegue lo smart contract.
  - Integrazione nativa del generatore casuale di numeri (Random Number Generator), che sarà disponibile alle dApp senza la necessità di sfruttare i RNG di terze parti.
  - Progettazione di un sistema per far interagire le catene al L2 con il L1. Dunque, le risorse potranno essere trasferite tra i due layer senza la necessità di ponti o collegamenti simili.
  - ISC scalabili orizzontalmente tramite l'aggiunta di nuove catene in grado di gestire il carico crescente di richieste.
  - Le catene al L2 potranno essere pubbliche o private, mantenendo in entrambi i casi l'interazione con il L1.

- **Svantaggi**
  - La sicurezza di ciascuna catena L2 dipende dai suoi nodi validatori (nodi Wasp), che sono indipendenti dai nodi della IOTA Mainnet. Ciò significa che IOTA può garantire la loro sicurezza solo se si dovesse trovare un modo per condividere la sicurezza dei nodi del L1 con i nodi del L2.
  - Come con qualsiasi soluzione di interoperabilità, se la maggior parte delle attività economiche passa al L2, il valore di L1 potrebbe essere compromesso.

Per quanto riguarda la seconda feature introdotta da Stardust, il framework di tokenizzazione renderà disponibili nuove funzionalità, tra cui la possibilità di coniare nuovi token nativi e NFT (Non-Fungible Token). IOTA utilizza un approccio diverso da quello delle altre piattaforme, le quali richiedono sempre l'utilizzo di smart contract per gestire i token. Questo nuovo approccio ha dei pro e dei contro:

- **Vantaggi**
  - La tokenizzazione è gratuita e consuma meno energia.
  - Il minting (pubblicazione token) è veloce e facile.
  - Gestione di condizioni personalizzate per l'utilizzo dei token.
  - Salvataggio degli NFT direttamente on-chain, senza la necessità di utilizzare cluster IPFS o soluzioni web2.
- **Svantaggi**
  - Nessuna programmabilità a causa della mancanza di smart contract nel Layer 1.
  - Nessuna compatibilità con gli standard ERC, sempre a causa della mancanza di smart contract.

In questo momento Stardust è ancora in fase di sviluppo. Difatti, si trova sulla rete Shimmer e solo nei prossimi anni, dopo un lungo processo di test e validazione, potrà sostituire Chrysalis sulla Mainnet.

1.2.3 Nodi

I nodi sono i contabili, i decisori e i validatori di tutte le informazioni presenti nella rete IOTA. Ogni nodo è in grado di gestire lo stato del registro (ledger) grazie alla conoscenza, in un dato momento, dei valori associati a tutti gli indirizzi esistenti [19].
Nello specifico, i nodi hanno il compito di:

- **Attaccare nuove transazioni al Tangle:** quando i nodi ricevono una nuova transazione, la attaccano al Tangle aggiungendola al loro database locale. Pertanto, in qualsiasi momento tutti i nodi potrebbero avere transazioni diverse nei rispettivi database locali. Queste transazioni costituiscono la vista personale che un nodo ha del Tangle. Per distribuire le transazioni sul resto della rete, i nodi sincronizzano i loro database locali con quelli vicini.

- **Sincronizzazione della rete:** come ogni sistema distribuito, i nodi della rete sincronizzano i loro database locali con i nodi vicini per formare un'unica versione condivisa del registro. Indipendentemente dalla sua posizione geografica, quando un nodo riceve una nuova transazione, proverà a comunicarla (mediante il protocollo gossip) con tutti i suoi vicini. Così facendo, alla fine tutti i nodi si sincronizzeranno e memorizzeranno le stesse transazioni all'interno dei loro database locali.
  Per sincronizzarsi, i nodi nella rete IOTA utilizzano le milestone. Quando una milestone referenzia un insieme di transazioni all'interno del database locale di un nodo, vuol dire che quelle transazioni sono state confermate dal Coordinatore. Pertanto, i nodi puntano a sincronizzarsi tenendo in considerazione le milestone, in maniera tale da avere la sicurezza di sincronizzarsi su una cronologia di transazioni che risulta essere

solida. Questo tipo di sincronizzazione è semplificata dal fatto che le milestone sono indicizzate, quindi facilmente identificabili. Quando un nodo è sincronizzato, dispone delle informazioni sufficienti per decidere quali nuove transazioni confermare o meno.

- **Confermare le transazioni:** tutte le transazioni restano in sospeso finché il nodo non è sicuro della loro validità. Innanzitutto, appena si riceve una transazione, bisogna controllare la sua formattazione. Nel caso in cui dovesse essere strutturata correttamente, si analizza il suo contenuto per verificarne la validità. Tuttavia, anche quando una transazione è valida, i nodi potrebbero non essere immediatamente in grado di confermarla a causa di possibili conflitti, come nel caso del double spending. Quando i nodi rilevano transazioni in conflitto, devono decidere quale transazione considerare confermata e quale scartare. Questa scelta viene presa sfruttando le regole di consenso distribuito integrate nel software dei nodi.

- **Memorizzare il saldo degli indirizzi:** tutti i nodi tengono traccia dei saldi degli indirizzi. Queste informazioni sono importantissime quando bisogna verificare la validità delle transazioni monetarie, controllando che un client non trasferisca più token di quelli che effettivamente sono disponibili sul suo conto. Dopo ogni transazione confermata, i nodi aggiornano il saldo degli utenti.

- **Mettere a disposizione dei client le API:** i nodi rappresentano per i client dei punti di ingresso nella rete. Difatti, ogni nodo si pone in ascolto su una determinata porta per ricevere i messaggi degli utenti ed elaborarli. Inoltre, i nodi mettono a disposizione dei client due tipologie di API:

  - **API HTTP:** consente ai client di interagire con il Tangle e chiedere di ricevere i tip, attaccare una nuova transazione al Tangle, effettuare il PoW e leggere le transazioni già attaccate.

  - **API Eventi:** pensata per interrogare i nodi in merito a degli eventi che possono verificarsi sul Tangle, come l'aggiunta di nuove transazioni. Questo può essere utile per monitorare il Tangle.

Attualmente esistono principalmente due tipologie di software per i nodi:

- **Hornet:** è un potente software, scritto in Go, creato e supportato dalla community di IOTA. È facile da installare e funziona persino su dispositivi meno potenti come il Raspberry Pi 4. Hornet supporta tutte le funzionalità degli ultimi aggiornamenti e il suo sviluppo continuerà anche in futuro per integrarlo nella prossima rete decentralizzata IOTA 2.0 [20].
- **Chronicle:** fornisce gli strumenti per gestire e accedere alle soluzioni basate su permanode e che sfruttano il framework backstage [21].

Scaricando gli appositi software, è possibile eseguire il proprio nodo localmente e diventare degli operatori della rete IOTA.

1.2.4 Client

Un client è un software che crea, invia, riceve ed elabora transazioni IOTA per conto di un utente (sia esso umano, macchina o dispositivo). Per eseguire questa tipologia di operazioni, un client generalmente utilizza delle librerie apposite. Queste sono state progettare per semplificare la connessione e l'interazione con i nodi della rete, esponendo delle funzioni di alto livello facili da utilizzare per gli sviluppatori. Dietro queste funzioni si celano delle operazioni atomiche più complesse che vengono eseguite in parte dal client e in parte dal nodo. Quest'ultimo espone un'API REST, attraverso la quale riceve ed elabora le richieste fatte da parte dei client [22].
Le librerie client più importanti sono le seguenti [23]:

- **Iota:** è stata progettata per semplificare la connessione e l'interazione con i nodi della rete IOTA. È possibile usarla per generare indirizzi,

firmare e inviare transazioni, leggere le transazioni sul Tangle e molto altro ancora.

- **Wallet:** contiene tutte le specifiche per creare un portafoglio di token IOTA e gestire, in maniera sicura, gli scambi monetari. Questa libreria utilizza la libreria Stronghold che implementa tutte le best practice di sicurezza per tenere al sicuro i dati sensibili.

- **Identity:** implementa gli standard e i modelli più comuni per gestire l'identità decentralizzata sia in modo agnostico rispetto al DLT sia rispetto agli altri metodi di IOTA. È stata progettata per identificare persone, organizzazioni, dispositivi e oggetti e per creare un rapporto di reciproca fiducia tra questi.

- **Streams:** nasce come sostituto della precedente libreria MAM. Streams implementa dei canali dati basati sul meccanismo di produttore (pubblica i dati) e consumatore (legge i dati). Ogni dato è cifrato e firmato per garantire la sua massima protezione e autenticità.

- **Stronghold:** basata su un database sicuro usato per contenere tutti i segreti come chiavi private e seed degli utenti. La sicurezza è assicurata grazie all'uso della crittografia e di tutte le migliori best practice in campo di sicurezza dell'informazione.

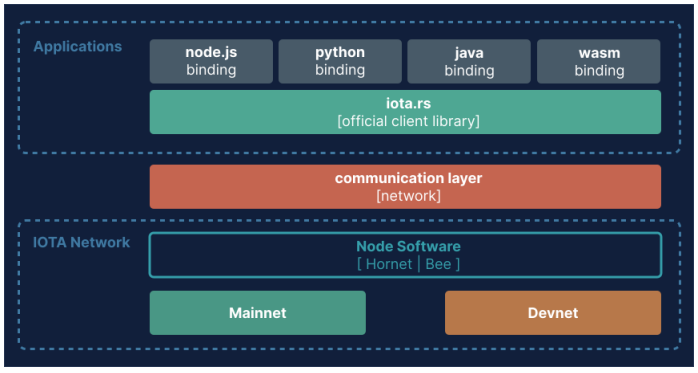

Figura 1.5: Rappresentazione della comunicazione tra client e nodo [24]

Le librerie sono tutte scritte in Rust ma è possibile effettuare dei binding con altri linguaggi di programmazione (JavaScript, Pyhton, Java e Wasm) per poterle integrare nella maggior parte delle applicazioni odierne.

1.2.5 Uniform Random Tip Selection

L'algoritmo Weighted Uniform Random Tip Selection è stato pensato per velocizzare i nodi nella fase di selezione dei tip. Questi ultimi sono dei messaggi già presenti nel Tangle, non ancora approvati, che puntano a un insieme di messaggi validi e che possono essere referenziati dai nuovi messaggi. L'obiettivo di questo algoritmo è quello di incrementare il throughput dei messaggi e favorire la selezione dei tip non-lazy, massimizzando il tasso di conferma.

Per essere definito non-lazy, il tip deve essere un messaggio che non è attaccato a dei messaggi troppo datati perché questi ultimi sicuramente saranno già stati opportunamente confermati tempo a dietro. Invece, selezionando dei tip non-lazy ed emettendo la prossima milestone, il nuovo messaggio favorirà la conferma dei messaggi che referenzia (non ancora confermati), incrementando così il tasso di conferme.

A ogni tip viene associato un punteggio per indicare la sua tipologia: 0 se è lazy, 1 se è semi-lazy e 2 se è non-lazy. Il punteggio da assegnare ai tip viene calcolato tramite un apposito algoritmo che tiene in considerazione le milestone emesse e l'insieme di transazioni già confermate.

Ogni nodo conserva un insieme di tip non-lazy, da cui seleziona casualmente dei tip tutte le volte che un client ne fa richiesta. Un nodo non deve eseguire la selezione dei tip se non è sincronizzato perché non è in grado di valutare correttamente se un tip è lazy o meno.

Un tip non viene immediatamente rimosso dall'insieme quando viene selezionato e inviato al client. Questo perché è possibile che venga nuovamente selezionato più avanti, favorendo l'ampliamento del Tangle e migliorando la velocità di sincronizzazione. Generalmente, un tip viene rimosso dall'insieme solo quando raggiunge un numero X di approvatori o trascorre un certo lasso

di tempo T. Le soglie X e T sono configurabili, ma solitamente si sceglie X=2 e T=3.

Per quanto riguarda i tip semi-lazy, questi non possono essere selezionati dai nodi e vengono lasciati indietro. Sarà poi il Coordinatore a selezionarli, tramite un apposito algoritmo, per confermarli tramite delle milestone.

Il precedente algoritmo di selezione dei tip era stato scritto in conformità al white paper originale di IOTA ed era parte del meccanismo di consenso. Tuttavia, divenne presto evidente che il calcolo dei pesi cumulativi era troppo pesante per uno scenario che puntava a un alto throughput.

Grazie all'algoritmo di conferma White-flag, introdotto con Chrysalis, non è più necessario eseguire la selezione dei tip tenendo in considerazione le mutazioni del registro mentre si esegue l'esplorazione. Pertanto, per la selezione dei tip è possibile impiegare un algoritmo più semplice e con prestazioni migliori [25].

1.2.6 Coordinatore e consenso

In questa versione del protocollo IOTA, i nodi raggiungono il consenso sullo stato del registro grazie all'aiuto dei blocchi milestone che vengono attaccati al Tangle da un nodo centrale chiamato Coordinatore e gestito direttamente dalla Fondazione IOTA.

Il Coordinatore è un client che invia messaggi firmati chiamati milestone di cui i nodi si fidano e che utilizzano per confermare i messaggi. I messaggi nel Tangle vengono considerati per la conferma solo quando fanno riferimento direttamente o indirettamente a una milestone che i nodi hanno convalidato.

Per riconoscere le milestone valide, tutti i nodi IOTA sulla stessa rete devono essere configurati in maniera tale da riconoscere le milestone firmate da uno specifico Coordinatore. Grazie a questo sistema di firme, i nodi riescono a controllare la firma delle milestone e verificano se sono state emesse o meno dal loro Coordinatore di fiducia.

Per garantire la conferma dei nuovi messaggi attaccati al Tangle, il Coordinatore invia regolarmente (ogni 10 secondi) delle milestone indicizzate e

firmate. Così facendo, i nodi potranno sincronizzarsi su uno stato comune del registro, facendo tutti riferimento a uno specifico indice (milestone indicizzata), ovvero a un determinato instante di tempo in cui tutti i messaggi precedenti alla milestone di riferimento sono stati correttamente confermati. Così facendo, i nodi potranno sincronizzarsi e raggiungere il consenso sullo stato del registro fino a uno specifico momento temporale.

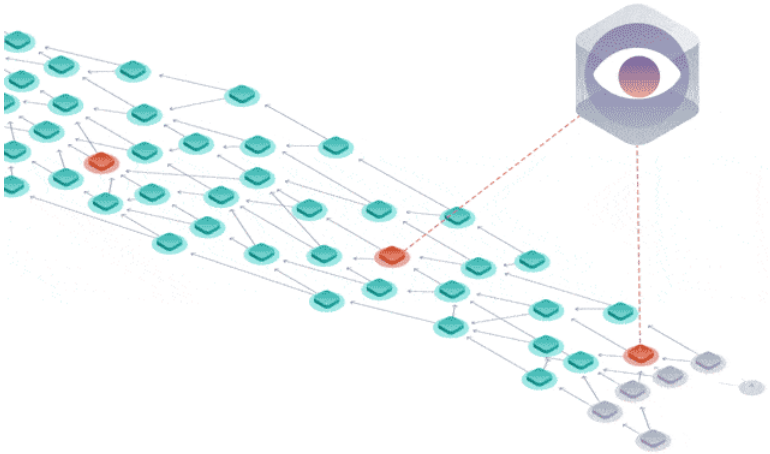

Figura 1.6: Rappresentazione del lavoro svolto dal Coordinatore, che produce le milestone (in rosso) per confermare i messaggi deli utenti (in verde) [26]

Al momento, il Coordinatore rappresenta una soluzione centralizzata ma temporanea. Infatti, sarà eliminato nelle prossime versioni del protocollo IOTA (Coordicide), dove si raggiungerà la totale decentralizzazione [25].

1.2.7 Trasferimento dati

Il trasferimento dei dati è veloce, immutabile, non falsificabile, gratuito, sicuro ed è una delle caratteristiche principali di IOTA. Questa funzionalità apre a una vasta gamma di casi d'uso che la maggior parte delle altre criptovalute non possono servire come fa IOTA.

I client, che possono essere dei wallet o delle applicazioni, inviano e ricevono messaggi attraverso i nodi di IOTA. I nodi, quindi, sono i punti di ingresso e di uscita per questi messaggi e consentono la comunicazione tra i client connessi.

Esistono diversi tipi di messaggi in IOTA. Alcuni messaggi trasferiscono valore (token IOTA o risorse digitali), altri trasferiscono solo dati puri e altri ancora possono persino contenere sia valore che dati. La struttura flessibile dei messaggi consente il trasporto decentralizzato di dati e valore garantendo il massimo grado di sicurezza e la totale assenza di commissioni [27].

1.2.7.1 Messaggio IOTA

Un messaggio IOTA è un oggetto contenente delle informazioni che si vogliono trasmettere sul Tangle. Ogni applicazione che utilizza il protocollo IOTA può inviare questi oggetti a un nodo. Quest'ultimo ha il compito di controllare i messaggi in arrivo e, qualora dovessero risultare validi e seguire le specifiche del protocollo, di trasmetterli attraverso la rete.

Se un nodo decide che un messaggio è valido, lo invierà ai suoi vicini diretti utilizzando il protocollo gossip. Ogni vicino che riceverà il messaggio lo ritrasmetterà ai suoi vicini e così via. Molto rapidamente, ogni altro nodo della rete vedrà il messaggio e avrà le stesse informazioni e la stessa conoscenza sullo stato della rete in un dato momento. Naturalmente, seguirà poi la sincronizzazione tra i nodi per confermare definitivamente i vari messaggi tramite le milestone emesse dal Coordinatore.

La struttura del messaggio può adattarsi per contenere diverse tipologie di informazioni. In base al payload specificato, il protocollo IOTA classifica questi pacchetti di dati e li elabora differentemente. La struttura del messaggio dovrà sempre essere corretta, altrimenti il nodo scarterà il messaggio.

1.2.7.2 Struttura del messaggio

Affinché un messaggio risulti sintatticamente corretto e possa essere elaborato da un nodo, il messaggio dovrebbe essere strutturato in maniera tale

da contenere [15]:

- **ID del messaggio:** hash crittografico univoco ottenuto a partire dai byte del messaggio. Identifica il messaggio e viene creato dal client che emette il messaggio.
- **ID della rete:** identifica la rete di cui il messaggio fa parte (Mainnet, Devnet, rete privata). I nodi accettano solo i messaggi che appartengono alla stessa rete del nodo.
- **Numero e ID dei genitori:** quantità e identificativi di messaggi passati referenziati dal messaggio. Ogni nuovo messaggio può referenziare da un minino di 1 fino a un massimo di 8 messaggi già attaccati al Tangle. Le informazioni sui messaggi referenziabili sono gestite dai nodi e inviate al client per poterle includere nel nuovo messaggio.
- **Lunghezza del payload:** la lunghezza del payload che il nodo deve conoscere a priori. Una lunghezza pari a 0 significa che il messaggio sarà senza payload.
- **Payload:** conterrà il tipo e i dati del payload.
- **Nonce:** necessario per far sì che il messaggio soddisfi il requisito di Proof-of-Work. Quest'ultimo viene calcolato localmente sul dispositivo che invia il messaggio e rappresenta anche una forma di protezione dagli spam. In alcuni casi, il nodo può occuparsi del PoW al posto del client. Questa è una funzione utile soprattutto per i dispositivi IoT che potranno pubblicare i messaggi nonostante non dispongano di sufficiente potenza computazionale per eseguire il PoW localmente. Questa scelta implementativa rende IOTA la soluzione ideale per i dispositivi mobili che lavorano su molti dati.

1.2.7.3 Validazione del messaggio

Un messaggio è considerato valido se vengono rispettate le seguenti regole sintattiche:

- La dimensione del messaggio non deve superare i 32 KiB (32 * 1024 byte).
- L'analisi sintattica della struttura del messaggio (parsing) non lascia nessun bit sconosciuto: ciò significa che tutte le informazioni del messaggio sono completamente leggibili dal nodo. Le informazioni illeggibili devono essere scartate perché potrebbero contenere codice malevolo.
- Il tipo di payload deve essere noto al nodo.
- Il PoW del messaggio soddisfa i requisiti minimi di PoW richiesti dalla rete.
- Il numero di messaggi referenziati deve essere compreso tra 1 e 8.

Il messaggio verrà accettato per l'elaborazione solo se i requisiti soprastanti saranno soddisfatti, altrimenti verrà scartato.

1.2.7.4 Payload

Un messaggio potrebbe contenere il payload. Nella Mainnet sono accettate solo tre tipologie di payload; a differenza della Devnet dove gli sviluppatori possono creare dei payload personalizzati per gestire più agevolmente qualsiasi tipo di dato.
I tre tipi di payload definiti nella Mainnet sono:

1. **Transazione:** usato per le transazioni che trasferiscono valore. Richiede la firma dell'utente.
2. **Milestone:** usato per le milestone emesse dal Coordinatore. Richiede la firma del Coordinatore.
3. **Indicizzato:** usato per inviare un messaggio contenente dati generici. Non richiede la firma. A ogni messaggio viene associato un indice, utile per consentire agli utenti di ritrovarlo all'interno del Tangle.

Poiché il protocollo IOTA evolve continuamente, in futuro saranno disponibili nuovi payload per gestire nuove tipologie di dati e nuove casistiche di utilizzo come, ad esempio, lo scambio di NFT.

1.2.7.5 Proof of Work

Il protocollo IOTA utilizza il Proof-of-Work (PoW) come mezzo per limitare la velocità della rete e prevenire gli attacchi di spam. Un messaggio, prima di essere pubblicato, deve dimostrare di aver soddisfatto i requisiti minimi di PoW richiesti dalla rete. Per la Mainnet, il punteggio minimo da soddisfare è pari a 4000; mentre per la Devnet il punteggio minimo è solo di 2000, la metà. Il punteggio PoW è definito come il numero medio di iterazioni necessarie per trovare il numero di zeri ternari finali nell'hash diviso per la dimensione del messaggio.
La validazione del PoW è eseguita nel seguente modo:

- Calcolare l'hash BLAKE2b-256 del messaggio serializzato, esclusa la parte nel Nonce. Convertire l'hash ottenuto nella codifica 192-trit.
- Prendere gli 8 byte del Nonce, rappresentarli in formato little endian, convertirli nella codifica 48-trit e attaccarli in fondo all'hash ottenuto dal punto precedente.
- Aggiungere un padding di tre zeri trits per ottenere una stringa di 243-trit.
- Calcolare l'hash Curl-P-81.
- Contare il numero di zeri ternari finali all'interno dell'hash ottenuto al punto precedente.
- Infine, il punteggio PoW sarà uguale a $3^{\#zero}/dimensione(messaggio)$.

Fortunatamente, il calcolo del punteggio PoW non deve essere implementato passo dopo passo dagli utenti, ma è direttamente integrato nelle librerie client e nei software di alcuni nodi.

1.3 IOTA Streams

Il framework IOTA Streams nasce per verificare l'autenticità e garantire la protezione dei messaggi che vengono scritti su uno specifico layer di trasporto. Alla base di Streams, si ha il protocollo Channels che è stato creato per sostituire la libreria MAM (Masked Authentication Messaging), già usata in precedenza per inviare sul Tangle messaggi cifrati e autenticati. I canali del protocollo Channels sono strutturati per essere più flessibili e adattarsi meglio a tutte le possibili casistiche del modello pub/sub, basato sulla figura dell'autore e dell'iscritto [29].

1.3.1 Autori

L'autore è il responsabile della generazione del canale e della sua configurazione. Ad esempio, lui può scegliere se il canale sarà a singolo branch o a branch multipli. Inoltre, potrà decidere se creare un canale pubblico, a cui sarà possibile accedere senza restrizioni; oppure un canale privato, a cui l'accesso sarà consentito solo dopo che l'autore avrà accettato le richieste di iscrizione inviate da altri utenti.

1.3.2 Iscritti

L'iscritto è qualsiasi utente, differente dall'autore, all'interno di un canale. Un iscritto può essere generato indipendentemente dall'autore ma, per scrivere su un branch o per elaborare il contenuto di un canale privato, è necessario che l'autore accetti la richiesta di iscrizione dell'iscritto. L'iscrizione al canale può avvenire secondo diverse modalità, che saranno approfondite nel prossimo paragrafo.

1.3.3 Messaggio keyload

Un messaggio keyload è un messaggio usato per restringere e controllare gli accessi ai contenuti del canale. Al suo interno, l'autore può specificare chi

dovrebbe essere in grado di decifrare i messaggi attaccati al keyload stesso.
Il messaggio keyload supporta due modalità di accesso:

- **Chiavi pubbliche degli iscritti:** l'iscritto maschera la propria chiave pubblica all'interno della richiesta di iscrizione al canale. La chiave sarà letta dall'autore che la archivierà nella sua istanza e la integrerà in un nuovo messaggio keyload che poi scriverà sul canale. D'ora in poi, l'iscritto potrà accedere a tutti i messaggi successivi al messaggio keyload che contiene la sua chiave pubblica.
- **Chiavi condivise in precedenza:** vengono utilizzate delle chiavi condivise tra i vari utenti del canale. La chiave condivisa viene distribuita tramite sistemi esterni a IOTA. Queste chiavi possono essere impiegate per implementare delle restrizioni sugli accessi al canale, senza la necessità del classico processo di iscrizione. Naturalmente, gli utenti che usano le chiavi condivise sono responsabili della sicurezza del processo di distribuzione.

1.3.4 Ramificazioni

I branch (rami) possono essere definiti come una sequenza di messaggi collegati fra di loro. Questa sequenza parte dal messaggio di annuncio, ovvero dal primo messaggio che l'autore posta sul canale al momento della sua creazione. Il messaggio di annuncio è di fondamentale importanza per gli iscritti perché la loro richiesta di iscrizione deve essere presentata come un messaggio che si collega direttamente al messaggio di annuncio. I branch sono tipicamente generati a partire da un messaggio firmato o da un messaggio keyload.
Il canale può assumere due forme:

- **Singolo branch:** una sequenza lineare di messaggi (simile ai canali MAM) dove ogni messaggio è collegato al precedente.

- **Branch multipli:** una sequenza di messaggi che non necessariamente sono collegati fra di loro sequenzialmente.

Quando viene generato un canale, l'autore deve decidere se il canale utilizzerà la ramificazione singola o la ramificazione multipla. Questa scelta influenzerà il lavoro dell'istanza Streams che eseguirà in maniera differente il sequenziamento dei messaggi da scrivere sul canale. Anche gli iscritti, durante l'elaborazione del messaggio di annuncio, capiranno il tipo di ramificazione utilizzata e configureranno la loro istanza per elaborare i messaggi basandosi sull'ordine in cui sono stati sequenziati.

1.3.5 Sequenziamento dei messaggi

Il sequenziamento è la metodologia usata all'interno di Streams per generare sequenzialmente gli identificatori dei messaggi, dipendentemente dalla forma del canale (branch singolo o multipli). Ogni messaggio è identificabile nel Tangle tramite la sua posizione indicizzata, ovvero un indirizzo univoco. L'identificativo viene generato tramite la combinazione delle seguenti informazioni:

- Istanza dell'applicazione (identifica il canale).
- Chiave pubblica di chi invia il messaggio.
- ID del messaggio precedente, a cui si attaccherà il nuovo messaggio.
- Numero del branch (identifica il branch).
- Numero di sequenza (posizione della sequenza di chi invia il messaggio).

Man mano che i messaggi vengono pubblicati e letti dal canale, si aggiornerà l'identificativo del messaggio, del branch e della sequenza all'interno di uno stato locale associato ai vari attori che operano sul canale. Questo modus operandi facilita gli utenti durante l'esplorazione della sequenza di messaggi e nella sincronizzazione con il canale.
Il sequenziamento varia in base al tipo di ramificazione del canale:

- **Singolo branch:** lo stato di sequenziamento viene aggiornato egualmente per tutti gli utenti. Ciò significa che, dopo l'invio di un messaggio, lo stato di ciascun utente aggiornerà il valore dell'ID del messaggio precedente ponendolo pari all'indirizzo del messaggio appena pubblicato e il numero di sequenza sarà incrementato di uno.

- **Branch multipli:** lo stato di sequenziamento di ciascun utente sarà aggiornato in modo indipendente. In questo caso, per poter tracciare la sequenza di messaggi pubblicati da ciascun utente all'interno di una struttura ad albero, è necessario che venga inviato un secondo messaggio assieme a quello principale. Questo messaggio secondario è chiamato messaggio di sequenza e contiene tutte le informazioni utili per derivare l'ID del messaggio appena pubblicato sul canale. I messaggi di sequenza vengono pubblicati su un branch secondario, generato nello stesso momento in cui è avvenuta la creazione del canale principale. Non appena un utente genera un nuovo messaggio, contemporaneamente sul branch secondario viene generato un messaggio di sequenza che consentirà agli altri utenti di avere un riferimento sulla posizione del messaggio appena pubblicato.

1.4 IOTA Identity

Il framework IOTA Identity implementa gli standard e i modelli più comuni per gestire l'identità decentralizzata, in modo agnostico rispetto alla DLT e agli altri framework di IOTA. Identity è stato progettato per identificare persone, organizzazioni, dispositivi e oggetti, agevolando la creazione di uno strato di fiducia reciproca tra tutti e tutto.

1.4.1 Identità decentralizzata

L'identità decentralizzata, conosciuta anche come Decentralized Identity (DID) o Self-Sovereign Identity (SSI), definisce un nuovo metodo per gestire

e autenticare le identità, rimuovendo la centralizzazione e conferendo il pieno controllo al soggetto identitario. La SSI restituisce autonomia e privacy agli individui, i quali potranno decidere quali informazioni personali comunicare e con chi. L'intento di questo approccio decentralizzato non è quello di limitare lo scambio di dati; anzi, l'obiettivo è quello di incrementarlo, favorendo però gli interessi dell'individuo che fornisce i dati e non quelli dell'azienda che sfrutta le informazioni degli utenti per trarne profitto.

Una DID è un identificatore univoco che può essere associato a una persona, un'organizzazione, un dispositivo IoT oppure a un generico oggetto. L'identificatore viene impiegato dal soggetto per identificarsi digitalmente e si presenta come un insieme di caratteri casuali preceduti da alcuni prefissi che determinano lo standard e l'implementazione usata. Ad esempio, una DID di IOTA si presenta come:
did:iota:8dQAzVbbf6FLW9ckwyCBnKmcMGcUV9LYJoXtgQkHcNQy.
I primi tre caratteri indicano che è necessario utilizzare lo standard DID del W3C per risolvere l'identificatore. Il successivo insieme di caratteri rappresenta il nome del metodo univoco, in questo caso iota, da usare per elaborare la DID e ottenere l'ultima versione del Documento DID dal Tangle.

In IOTA, la creazione di una DID inizia con la generazione casuale di una coppia di chiavi asimmetriche (chiave pubblica e chiave privata). La coppia può essere generata direttamente dal framework IOTA Identity o può essere fornita come parametro iniziale durante il processo di creazione. La chiave pubblica viene sottoposta a hashing utilizzando l'algoritmo BLAKE2b-256. Il fingerprint ottenuto diventerà la DID e fungerà da collegamento permanente e dimostrabile tra la coppia di chiavi iniziale e la DID. La chiave pubblica viene quindi incorporata nel documento DID iniziale e potrà essere utilizzata per verificare le firme create con la chiave privata corrispondente. La chiave privata sarà usata come una sorta di identificativo del soggetto. Il Documento DID sarà pubblicato sul Tangle diventando immutabile. Questa immutabilità è ciò che rende affidabile l'identità decentralizzata basata sulla DLT. Le chiavi pubbliche all'interno del documento DID non potranno mai

essere modificate senza avere accesso alla chiave privata, concedendo così all'utente il pieno controllo sulla propria identità [30].

1.4.2 Documento DID

La DID è strutturata in maniera tale da contenere tutte le informazioni necessarie per recuperare dal Tangle l'ultima versione del Documento DID. Questo documento contiene tutte le informazioni associate al soggetto identificato. Più nello specifico, il Documento DID contiene principalmente due tipologie di dati: chiavi pubbliche e servizi.

Le chiavi pubbliche possono essere utilizzate per dimostrare la proprietà dei dati personali associati all'identità. Innanzitutto, bisogna usare la chiave privata per firmare crittograficamente i dati. Successivamente, tramite la relativa chiave pubblica, è possibile verificare la firma digitale e controllare l'effettiva proprietà dei dati. In pratica, la proprietà della chiave privata dimostra la proprietà dell'identità. Di conseguenza, è importante proteggere le chiavi private e non divulgarle mai. Inoltre, è possibile sfruttare la chiave pubblica di un altro utente per cifrare i dati a lui indirizzati, garantendo la loro riservatezza. Una volta ricevuti, il destinatario potrà decifrarli con la sua chiave privata e leggere i dati in chiaro.

I servizi presenti nel Documento DID non sono altro che degli URI (Uniform Resource Identifier) che puntano a ulteriori informazioni sull'identità. Questi servizi sono pubblicamente disponibili e possono essere letti da tutti; pertanto, non dovrebbero contenere informazioni di identificazione personale (Personally Identifiable Information) [31].

1.4.3 Credenziali verificabili

Le credenziali verificabili (Verifiable Credentials) sono dichiarazioni fatte su un soggetto che possono essere verificate crittograficamente da una terza parte. Esistono diversi attori che ricoprono ruoli distinti all'interno di un sistema di credenziali verificabili:

- **Subject:** entità su cui vengono fatte dichiarazioni.
- **Holder:** entità che possiede una copia delle credenziali verificabili.
- **Issuer:** entità che rilascia dichiarazioni su un Subject.
- **Verifier:** entità che controlla se le VC dell'Holder sono legittime o meno.

Nel framework Identity la generazione delle credenziali verificabili avviene come segue:

1. Generare le identità digitali per l'Issuer e il Subject. Le loro DID devono essere pubblicate sul Tangle.
2. Issuer crea una credenziale verificabile per il Subject, all'interno della quale ci saranno delle apposite dichiarazioni sul soggetto. Le credenziali devono essere firmare con la chiave privata dell'Issuer, la cui corrispondente chiave pubblica è pubblicata sul Tangle.
3. L'Issuer convalida le proprietà delle credenziali appena emesse.
4. Le credenziali verificabili vengono archiviate e trasmesse all'Holder off-chain in modo sicuro.

Per quanto riguarda il processo di verifica, l'Holder presenta le sue credenziali a un Verifier. Durante la verifica bisogna controllare la DID dell'Holder, appurare che le VC siano di sua proprietà e verificare la firma dell'Issuer tramite la sua chiave pubblica presente sul Tangle.

Tutto questo meccanismo restituisce agli Holder il pieno controllo sui propri dati, i quali diventano persino più affidabili proprio perché sono verificabili [32].

1.4.4 Perché scegliere IOTA Identity?

L'ecosistema IOTA è particolarmente adatto per implementare un'identità digitale multi-soggetto. Infatti, la rete IOTA è progettata sia per gli esseri umani che per le macchine, fornendo una piattaforma affidabile per la comunicazione tra individui, organizzazioni e dispositivi IoT. I principi della Fondazione IOTA come la piena trasparenza, apertura e innovazione decentralizzata forniscono un ambiente ideale per questo tipo di identificazione:

- **Nessuna autorizzazione e piena decentralizzazione:** a differenza delle reti permissioned, tutti possono partecipare al consenso senza che gli venga richiesto l'accesso. Nessun soggetto è incentivato dal profitto come in tutte le blockchain. Ciò rende IOTA neutrale e resistente alla censura.

- **Rete pubblica:** tolti i casi in cui viene esplicitamente impiegata la crittografia, ognuno può analizzare le transazioni nella rete. La rete è intrinsecamente trasparente.

- **Zero commissioni:** le transazioni di dati e valore sulla rete sono eseguite gratuitamente. Al contrario di Bitcoin o Ethereum, la registrazione e la comunicazione delle identità possono essere eseguite sulla rete senza la necessità di acquistare i token della criptovaluta.

- **Open Source:** tutti possono visualizzare e contribuire al codice.

- **Facile da usare:** il framework Identity può essere facilmente utilizzato tramite un'API di alto livello, che si fa carico di interagire con le più complesse API di basso livello. Inoltre, gli aspetti relativi alla protezione dei dati vengono gestiti automaticamente dal framework Stronghold.

I dati devono essere archiviati in modo immutabile su un registro distribuito per garantire la natura decentralizzata del protocollo di identità digitale. Questo sistema aumenta persino la privacy individuale, perché il

contatto con l'Issuer viene eliminato dall'interazione diretta tra Holder e Verifier. Gli Issuer non saranno più in grado di tracciare quando e con quale frequenza gli Holder utilizzano le proprie credenziali [33].
Nell'ambito della gestione delle identità digitali, il Tangle svolge le seguenti funzionalità:

1. **Registro delle chiavi pubbliche:** il Tangle viene utilizzato dagli Issuer come un'infrastruttura decentralizzata per chiavi pubbliche (Decentralized Public Key Infrastructure). Ciò consente ai Verifier di verificare la validità di una firma digitale senza far affidamento su un server centralizzato. Inoltre, il Tangle può essere usato anche come registro per le VC.

2. **Revoca delle credenziali verificabili:** una credenziale verificabile può essere revocata, ovvero non sarà più in grado di superare la verifica. La revoca viene archiviata in modo immutabile sul Tangle, per far sì che nessun Holder possa tentare di utilizzare le proprie credenziali revocate. Questa operazione è stata pensata come una semplice disattivazione della chiave pubblica, in maniera tale da essere conforme al GDPR.

1.5 IOTA Smart Contracts

IOTA Smart Contracts (ISC) è una piattaforma che implementa smart contract scalabili e flessibili all'interno dell'ecosistema IOTA. Permette a chiunque di creare una blockchain di smart contract e ancorarla al Tangle, sfruttando tutti i relativi vantaggi.

1.5.1 Smart Contract

Gli smart contract consistono in una serie di funzioni che operano sul loro stato interno (insieme di variabili). L'esecuzione delle funzioni deve essere deterministica: partendo da un certo stato e fornendo specifici dati in input, le funzioni dovranno produrre sempre la stessa modifica dello stato

e restituire un particolare output. Il determinismo è fondamentale affinché tutti i nodi possano raggiungere il consenso e convalidare il risultato prodotto dall'esecuzione indipendente e distribuita dello smart contract.

Di conseguenza, durante la sua esecuzione, uno smart contract non può recuperare dati da fonti esterne, perché queste non garantirebbero la restituzione dello stesso dato a ogni invocazione. Pertanto, le funzioni devono essere implementate per lavorare solo sull'insieme completo e atomico di dati forniti in input e sullo stato interno dello smart contract.

Le funzioni dello smart contract non hanno accesso a un file system, né possono utilizzare fonti di temporizzazione o casualità. Eventuali dati temporali o casuali devono essere forniti sempre come parte dei dati in input. Nemmeno il multithreading è consentito, visto che è altamente non deterministico.

Per evitare i cicli infiniti e per garantire che le differenze temporali tra i processori non influenzino il consenso sul risultato finale di processi che eseguono per lunghi periodi di tempo, gli smart contract utilizzano un sistema che limita il tempo massimo di esecuzione delle funzioni tramite l'utilizzo del gas. Ogni istruzione brucia una certa quantità di gas, per cui qualsiasi funzione che esaurisce il gas lo farà esattamente nello stesso punto, indipendentemente da chi la esegue. Questa è l'unica soluzione che consente una computazione Turing-completa ma, allo stesso tempo, deterministica. Inoltre, il gas può essere impiegato anche come strumento per valutare il costo monetario per eseguire una certa funzione [34].

1.5.2 Caratteristiche degli ISC

Una caratteristica unica degli ISC è quella di eseguire, in parallelo e in modo sicuro, più blockchain. Ogni catena gestisce la propria serie di smart contract. Questi ultimi possono ricevere richieste asincrone, anche se ogni blockchain le gestirà in modo sincrono, ordinandole in base al raggiungimento del consenso tra i vari nodi che le elaborano. Alcuni smart contract sono già integrati nelle catene, altri possono essere definiti e caricati dinamicamente

dagli utenti.

Gli smart contract possono chiamare le funzionalità di un altro smart contract in modo sincrono o asincrono. Le chiamate sincrone sono simili alle chiamate di una subroutine; mente quelle asincrone vengono racchiuse in una transazione che verrà pubblicata sul Tangle per poi essere letta ed eseguita solo in un secondo momento. I nodi di elaborazione espongono persino un'API web alla quale le applicazioni utente possono inviare direttamente le richieste. Queste richieste prendono il nome di richieste off-ledger, che si distinguono da quelle pubblicate sul Tangle chiamate on-ledger. Le richieste off-ledger possono essere inviate con una frequenza molto più elevata rispetto alle richieste on-ledger perché non bisogna attendere la conferma della scrittura sul Tangle.

Per entrambi i tipi, il mittente deve firmare la richiesta con la propria chiave privata per garantire la sua identificazione in maniera univoca e sicura. È altrettanto importante identificare il chiamante della funzione dello smart contract. Durante l'elaborazione di una richiesta, il mittente rimarrà lo stesso ma il chiamante cambierà a ogni chiamata sincrona effettuata. Ciò consentirà il trasferimento di risorse tra chiamate a diversi smart contract e consentirà alla funzione chiamata di identificare più facilmente l'origine di tali risorse. Bisogna ricordare che la funzione può accedere solamente alle risorse fornite dal chiamante della funzione.

Il modo in cui le risorse vengono fornite a una funzione dello smart contract è specificato nella cosiddetta allowance (ciò che è consentito). La funziona riceve il consenso per trasferire a sé la proprietà delle risorse indicate nella allowance, a seconda delle esigenze della funzione stessa. La allowance viene presa dall'account on-chain del chiamante. Ciò significa che il chiamante deve assicurarsi che queste risorse siano disponibili nel suo account quando eseguirà la chiamata della funzione. Per le richieste off-ledger, il chiamante dovrà depositare le risorse necessarie sul suo account on-chain prima di inviare la richiesta all'API. Per le richieste on-ledger, le risorse potranno essere depositate prima dell'invio della richiesta oppure potranno anche essere

inviate come parte della richiesta. In quest'ultimo caso, qualsiasi risorsa inviata come parte di una richiesta on-ledger finirà nell'account on-chain del mittente.

Il meccanismo dell'allowance rende impossibile l'assegnazione delle risorse a uno smart contract sbagliato o inesistente, evitando il rischio di perderle per sempre. Tutte le risorse inviate all'account on-chain resteranno lì al sicuro fino a quando il mittente non deciderà di usarle per una funzione oppure di ritirarle in qualsiasi momento [34].

1.5.3 Macchina virtuale Wasm

ISC opera su un ambiente sandbox, che espone un'API attraverso la quale gli utenti possono interagire, in modo deterministico e senza rischi per la sicurezza, con le funzionalità messe a disposizione dagli smart contract. L'API è compatibile con vari tipi di macchine virtuali (VM) e viene impiegata proprio per archiviare, accedere e modificare lo stato degli smart contract in esecuzione sulle VM.

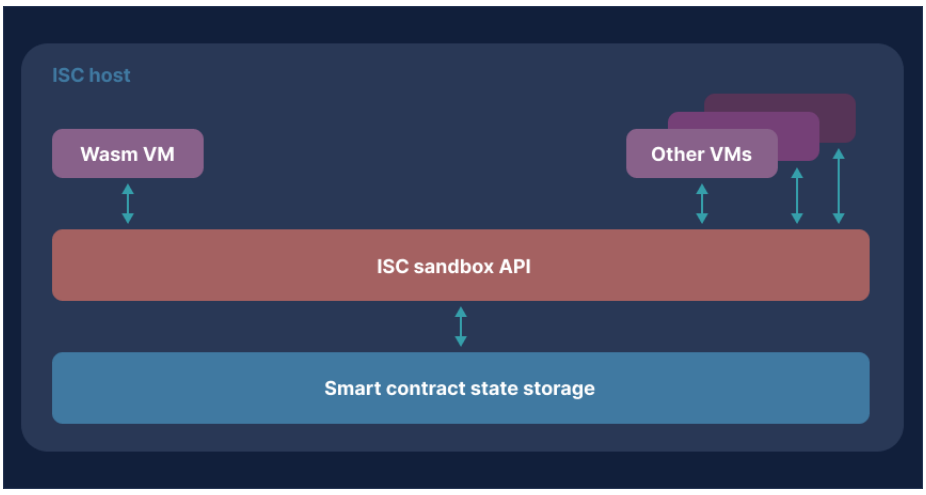

Figura 1.7: Rappresentazione delle componenti interne agli host ISC [36]

Più nello specifico, l'ambiente sandbox mette a disposizione le seguenti funzionalità:

- Accesso ai metadati dello smart contract.
- Accesso ai dati della richiesta usata per chiamare una funzione dello smart contract.
- Accesso ai dati presenti nello stato dello smart contract.
- Restituzione al chiamante del risultato ottenuto dalla funzione chiamata.
- Accesso ai token destinati allo smart contract.
- Possibilità di inizializzare o richiamare le funzioni di un altro smart contract.
- Accesso a diverse funzioni di utilità fornite dall'host.

ISC mette a disposizione delle VM predefinite, ma supporta persino quelle personalizzate degli utenti. Tra le VM di IOTA, è presente la VM Wasm implementata usando WebAssembly (Wasm). La macchina virtuale in questione è basata sulla runtime Wasmtime, che consente il caricamento dinamico e l'esecuzione di smart contract scritti in Rust, Go o TypeScript e compilati in codice intermedio Wasm.

Il codice Wasm è ideale per gli smart contract sicuri, visto che ogni porzione di codice viene eseguita nel proprio spazio di memoria e non può accedere a nulla al di fuori di tale memoria. Inoltre, la VM Wasm fornisce l'accesso all'esterno solo alle funzionalità necessarie per il corretto funzionamento degli smart contract. Nel caso degli ISC, viene fornito l'accesso solo all'API dell'ambiente sandbox. Più nello specifico, l'accesso sarà concesso tramite una libreria piccola, autonoma e collegata al codice Wasm: WasmLib.

Inizialmente, la libreria WasmLib era stata implementata con il linguaggio di programmazione Rust. In seguito, il supporto è stato esteso anche ai linguaggi Go e TypeScript. Tutte le implementazioni della WasmLib utilizzano solo un ristretto sottoinsieme comune del linguaggio host. Lo stesso vale per l'interfaccia che viene usata dagli sviluppatori degli smart contract.

Ciò mantiene lo stile di codifica molto simile, salvo alcune idiosincrasie sintattiche. Queste peculiarità tendono a semplificare la scrittura degli smart contract, soprattutto per chi è alle prime armi e non ha mai usato i linguaggi di programmazione citati precedentemente.

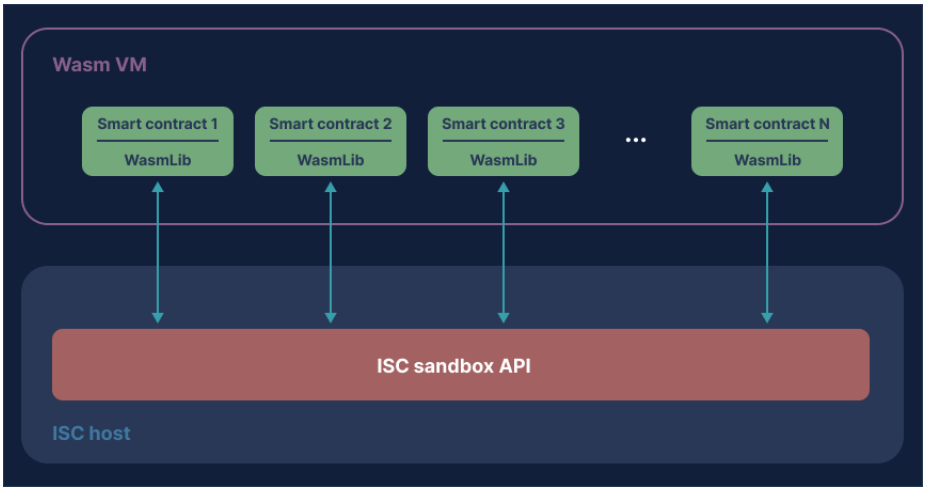

Figura 1.8: Rappresentazione nel dettaglio della VM Wasm [37]

Come è possibile evincere dalla Figura 1.8, la VM Wasm può eseguire contemporaneamente un numero elevato di smart contract. Ognuno di questi incorpora la propria copia della WasmLib, che fornisce allo smart contract corrispondente le funzionalità necessarie per interagire con il sandbox sottostante e, di conseguenza, con il layer di archiviazione che gestisce il suo stato [35].

1.5.4 Schema Tool

Gli smart contract devono essere molto robusti. La natura generica della WasmLib favorisce la flessibilità, ma anche la possibilità di commettere degli errori. Il modo migliore per aumentare la robustezza è utilizzare un generatore di codice che si occuperà delle attività di codifica più ripetitive. Il codice generato sarà accurato e affidabile al 100%. Inoltre, nel caso in cui si dovessero apportare delle modifiche all'interfaccia dello smart contract, il

generatore potrebbe rigenerare nuovamente il codice, integrando i cambiamenti effettuati. Infine, un generatore di codice riesce a supportare persino linguaggi di programmazione differenti.
Le operazioni che spesso contengono codice ripetitivo sono:

- Verifica dei diritti di accesso alle funzioni.
- Verifica dei tipi di parametri passati alla funzione.
- Verifica della presenza dei parametri obbligatori.
- Impostazione dell'accesso ai parametri e al risultato delle funzioni e allo stato dello smart contract.
- Definizione delle costanti.

Per gli ISC, la generazione del codice è affidata a uno strumento chiamato Schema Tool. Quest'ultimo mette a disposizione un file in cui è possibile definire schematicamente lo smart contract che si desidera implementare. Tutti gli aspetti peculiari sullo smart contract dovranno essere chiaramente definiti all'interno di questo file. Così facendo, sarà più semplice generare automaticamente uno scheletro di partenza per lo smart contract e la sua interfaccia che verrà poi usata come riferimento da chi vorrà interagire con esso. Dopo la generazione, lo sviluppatore dovrà solo implementare le funzioni già definite.

All'interno del file di definizione dello schema è possibile specificare le variabili di stato dello smart contract, delle strutture personalizzate e le funzioni (Func e View) da implementare con i relativi diritti di accesso, parametri e risultati attesi. Con così tante informazioni dettagliate, avviene persino la generazione delle interfacce delle funzioni, che saranno poi impiegate lato client per effettuare le chiamate in modo unico e coerente. Inoltre, per cercare di ridurre al minimo gli errori accidentali, vengono generati ed eseguiti dei controlli rigidi sui parametri, sui risultati e sulle variabili di stato [38].
Lo smart contract supporta due tipi di funzione:

- **Func:** consente l'accesso completo allo stato dello smart contract, con la possibilità di modificarlo. Le Func possono essere chiamate tramite richieste on-ledger e off-ledger. La chiamata a una Func risulta completata solo dopo la registrazione sul Tangle dell'aggiornamento di stato effettuato.

- **View:** consente l'accesso allo stato dello smart contract solo in lettura, senza la possibilità di modificarlo. Le View possono essere chiamate solo tramite richieste off-ledger e vengono utilizzate soprattutto per recuperare lo stato dello smart contract in modo efficiente.

La distinzione tra questi due tipi di funzione è supportata dall'utilizzo di differenti contesti di chiamata, definiti direttamente dalla WasmLib. Per le Func, il contesto di chiamata è ScFuncContext, il quale fornisce un insieme più completo e mutabile di funzionalità per operare sullo stato. Mentre, per le View, il contesto è ScViewContext, che mette a disposizione un sottoinsieme limitato e immutabile di funzionalità per accedere in maniera più ristrettiva allo stato. Grazie alla separazione dei contesti di chiamata, diventa anche più semplice eseguire i controlli in fase di compilazione sui tipi di dati utilizzati nell'implementazione delle funzioni [39].

1.5.5 Vantaggi degli ISC

Tra i principali vantaggi degli ISC troviamo:

- **Scalabilità e fee:** a causa della struttura ordinata da gestire e dei tempi di esecuzione degli smart contract, una singola blockchain può processare solo un certo numero di transazioni al secondo. Questo comporta un throughput basso e commissioni elevate. ISC invece supporta il collegamento di più catene al Tangle, favorendo l'esecuzione parallela degli smart contract e la comunicazione fra di loro. Grazie a ciò, la piattaforma ISC può vantare un throughput più elevato e fee quasi nulle. Inoltre, ISC è una soluzione del Layer 2 in cui sono presenti solo

nodi dedicati all'esecuzione degli smart contract. Dunque, il resto della rete IOTA non è influenzato dal traffico degli ISC.

- **Catene personalizzate:** visto che chiunque può avviare una nuova catena e stabilirne le regole, diventa possibile implementare vari scenari d'uso che altrimenti non sarebbero disponibili nelle piattaforme a singola catena. Questo rende possibile la creazione di una blockchain privata che opera solo su dati privati, sfruttando la sicurezza offerta dalla rete IOTA.

- **Flessibilità:** ISC è agnostico rispetto alla macchina virtuale che esegue il codice dello smart contract. Grazie a questa caratteristica, tutti i tipi di VM possono essere supportati in una catena ISC. Purtroppo, questa libertà rende gli ISC più complessi rispetto agli smart contract convenzionali.

Capitolo 2

Progettazione

Lo scopo di questo capitolo è quello di introdurre il sistema, chiarendo meglio il suo funzionamento ad alto livello. Nello specifico, si analizzeranno le caratteristiche principali e verranno elencati i requisiti funzionali e non. Infine, verranno illustrati vari casi d'uso per comprendere meglio le funzionalità offerte agli utenti.

2.1 Concept

Oggigiorno, viviamo in una società data-driven dove la produzione, lo scambio e l'analisi dei dati rappresenta il fulcro per molti settori. La maggior parte dei dati sono memorizzati, elaborati e distribuiti sfruttando entità centralizzate. Gli utenti e i dispositivi IoT svolgono il ruolo di produttori ma, allo stesso tempo, sono figure assenti nel processo di analisi e monetizzazione delle informazioni prodotte. Questo è causato dalla poca trasparenza delle aziende ma anche dal poco interesse degli utenti sul ciclo di vita dei dati.

L'intento di questa tesi è realizzare un ecosistema all'interno del quale l'utente possa riacquisire maggiore controllo sui propri dati, cambiando il modus operandi soprastante. La DLT rappresenta le fondamenta di questo ecosistema perché consente il raggiungimento della completa decentralizzazione, eliminando tutte le autorità centralizzate. Grazie a ciò, gli utenti non

solo ricopriranno il ruolo di produttori ma anche quello di gestori dei propri dati tramite l'utilizzo di un sistema di autorizzazioni. Le case di sviluppo dovranno adattarsi a questo nuovo paradigma, sviluppando delle applicazioni in grado di raccogliere ed elaborare i dati degli utenti solo quando questi avranno fornito l'esplicito consenso. Inoltre, le applicazioni non potranno più memorizzare i dati su database centralizzati, ma dovranno salvarli su registri decentralizzati. Persino le elaborazioni dovranno essere eseguite sui dispositivi dei produttori e non su dispositivi esterni, proprio per conservare la tracciabilità delle operazioni svolte. Quello che si viene a delineare è un modello produttore/consumatore decentralizzato, alla base del quale si ha un sistema di autorizzazioni che governa l'accesso ai dati.

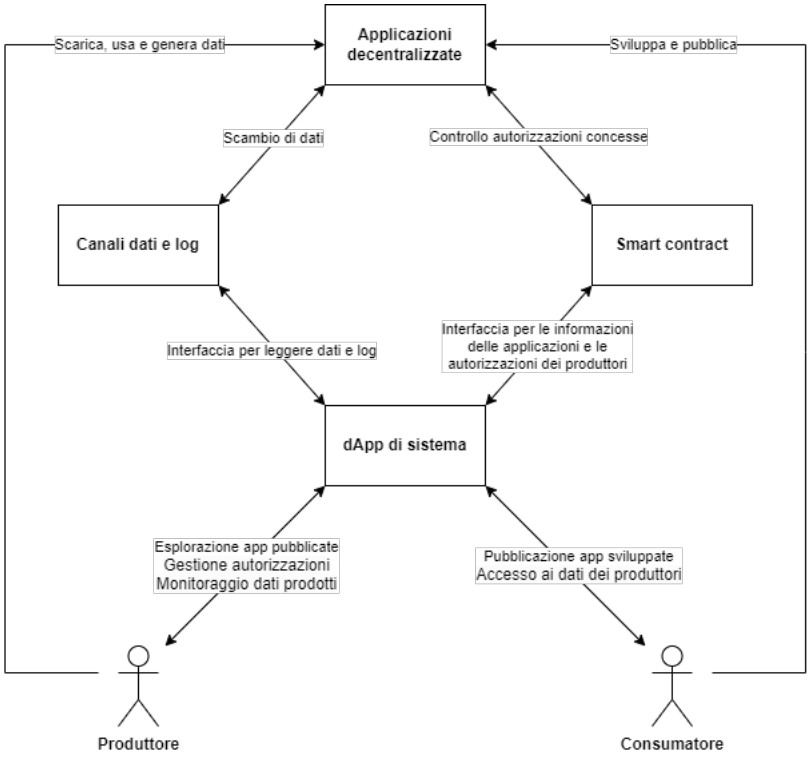

Figura 2.1: Rappresentazione schematica dell'ecosistema

L'ecosistema diventa un vero e proprio store di applicazioni decentralizzate, simile a Google Play e App Store per le app moderne. La particolarità di questo store è che non punta solamente a contenere le informazioni sulle

applicazioni pubblicate; ma consente ai produttori di gestire le autorizzazioni per ogni applicazione che usano e di monitorare gli accessi dei consumatori ai dati prodotti. La gestione delle applicazioni pubblicate e delle autorizzazioni degli utenti viene affidata a degli smart contract. Per interagire con questi, l'ecosistema mette a disposizione una dApp di sistema, tramite la quale è possibile registrarsi e ricevere un'identità decentralizzata, pubblicare nuove app, leggere le informazioni delle app già pubblicate, concedere o rimuovere le autorizzazioni e, infine, monitorare l'utilizzo dei dati prodotti mediante appositi log.

Tutto questo processo garantisce la protezione dei dati perché è basato su una tecnologia decentralizzata sicura. In aggiunta, i sistemi decentralizzati hanno persino il vantaggio di essere sempre disponibili e di offrire i propri servizi con una bassa latenza. Infine, la nuova gestione dei dati deve essere gratuita in maniera tale da incentivare la sua adozione a discapito delle soluzioni centralizzate già esistenti.

Gestione dei dati Questo ecosistema nasce con l'intento di gestire l'intero ciclo di vita dei dati, composto principalmente da quattro fasi: produzione, memorizzazione, condivisione ed elaborazione. Per quanto riguarda la seconda fase, un ruolo fondamentale è svolto dalla DLT, che assicura disponibilità continua e nessuna censura. I dati vengono salvati su appositi canali, accessibili esclusivamente dal produttore e dai consumatori autorizzati. L'accesso ai dati è mediato dagli smart contract, che si occupano di identificare gli utenti, controllare le loro autorizzazioni e interagire con le applicazioni pubblicate. Gli utenti possono interagire con gli smart contract tramite l'apposita dApp di sistema. Ogni dato deve essere accessibile solo dopo il rilascio dell'apposito consenso da parte del suo produttore. Ovviamente, ogni dato sulla DLT pubblica è pubblicamente accessibile; pertanto, tutte le informazioni prodotte all'interno di questo ecosistema saranno cifrate, in maniera tale da garantire la lettura del contenuto in chiaro solo a chi è autorizzato. Le informazioni raccolte potranno essere gestite secondo tre

modalità: privata, ristretta e aggregata. Nel primo caso, i dati saranno accessibili solo dal loro produttore. Nella seconda modalità, il produttore e i consumatori autorizzati potranno accedere ed elaborare i dati. Nell'ultimo caso, i consumatori autorizzati riceveranno solo a un versione aggregata dei dati e non potranno accedere ai loro singoli valori.

Applicazioni decentralizzate Con il termine applicazione si intende qualsiasi software utilizzato dai produttori per generare dei dati, che poi successivamente saranno acceduti dai consumatori. Un'applicazione può essere sia un'app per smartphone che un pezzo di codice in esecuzione su un dispositivo IoT. Però, le applicazioni in questo ecosistema hanno delle caratteristiche che si discostano dalle classiche app odierne. Prendiamo come esempio Instagram: gli iscritti (produttori) interagiscono con il social, producendo dati che verranno usati da Meta (consumatore) per eseguire analisi di mercato, testare il funzionamento di nuove funzionalità e profilare gli utenti per proporre contenuti pubblicitari personalizzati. In questo caso, il produttore è conscio del fatto che sta fornendo i suoi dati, ma non conosce esattamente come saranno processati e se verranno ceduti a terzi.

Le applicazioni all'interno di questo ecosistema dovranno essere progettate per essere decentralizzate e più trasparenti nei processi di raccolta ed elaborazione dei dati degli utenti. Ciò si traduce in un software che non stravolge le sue funzionalità; ma cambia semplicemente le modalità in cui avviene la raccolta dei dati usufruendo della DLT e rende il processo di elaborazione esplicito per far sì che il produttore sappia esattamente in che modo i suoi dati vengono utilizzati dal consumatore. Tornando all'esempio di Instagram, il social potrà conservare tutte le funzionalità implementate, però dovrà ricevere il consenso esplicito dagli utenti prima di raccogliere ed elaborare i loro dati. Pertanto, per valutare se concedere o meno il consenso, gli utenti dovranno conoscere come avviene la raccolta, qual è il suo fine, chi può accedere

a questi dati e monitorare il loro utilizzo. Lo stesso discorso vale per l'elaborazione dei dati, i cui dettagli dovranno essere noti e che dovrà essere eseguita localmente dispositivo del produttore. Per ogni operazione da svolgere sui dati, gli sviluppatori dovranno fornire il relativo codice software, dando la possibilità ai produttori di poter verificare se la raccolta o l'elaborazione delle informazioni avviene effettivamente in maniera coerente con quanto dichiarato. Ogni volta che il produttore fornisce il consenso, questo verrà registrato nello stato interno degli smart contract. Questi ultimi saranno accessibili dalle applicazioni per controllare se uno specifico utente ha fornito o meno il suo consenso per una certa operazione di raccolta o di elaborazione. L'applicazione procederà con l'esecuzione dell'operazione sui dati solo se è presente l'autorizzazione per farlo.

Produttori I produttori di dati sono tutti quei soggetti che usano le applicazioni, generano i dati e li memorizzano su registri distribuiti. Tra i produttori possiamo identificare sia essere umani che dispositivi IoT. All'interno di questo ecosistema, chi produce deve mantenere il pieno controllo sui propri dati. Questo è reso possibile dal meccanismo delle autorizzazioni, tramite il quale i produttori possono decide quali dati fornire, chi può visionarli e revocare l'accesso in qualsiasi momento. Siccome i permessi sono gestiti dagli smart contract, i produttori usano la dApp di sistema per fornire, revocare e visionare le varie autorizzazioni. Inoltre, l'ecosistema implementa i log per tracciare ogni operazione di lettura e scrittura sui dati svolta da qualsiasi utente (produttore e consumatori). Così facendo, il produttore potrà controllare il comportamento dei consumatori per verificare chi, quando e quanto spesso accede ai dati, facilitando l'individuazione di anomalie e di abusi.

Consumatori I consumatori di dati sono tutti coloro che sfruttano i dati dei produttori. Nella maggior parte dei casi, il ruolo di consumatore è

svolto dalle case di sviluppo del software utilizzato dai produttori; ma, in generale, può essere ricoperto da qualsiasi utente dell'ecosistema che riceve l'autorizzazione per accedere ai dati prodotti da un altro utente. Il consumatore, che è anche sviluppatore di un'applicazione, può definire secondo quale modalità (privata, ristretta o aggregata) avverrà la gestione dei vari dati raccolti tramite la sua applicazione. Queste impostazioni, unite alle informazioni sull'applicazione (nome, descrizione, url per download), devono essere salvate all'interno degli smart contract per poter essere pubblicamente accessibili da tutti gli utenti dell'ecosistema. Anche in questo caso, il consumatore sviluppatore interagisce con gli smart contract tramite la dApp di sistema. Tutte le operazioni svolte sui dati dai consumatori saranno registrate nei log dedicati, consultabili dal relativo produttore.

2.2 Requisiti

Il sistema da realizzare è basato su un'architettura decentralizzata formata da un insieme di entità che interagiscono fra di loro tramite delle apposite applicazioni decentralizzate. Tra queste ultime sono presenti le applicazioni implementate dai consumatori per i produttori, ma anche tutti i software in esecuzione sui nodi della DLT. Le entità coinvolte comprendono gli utenti dell'ecosistema (produttori e consumatori), i dispositivi che loro utilizzano, i nodi che interagiscono con i registri distribuiti e quelli che eseguono gli smart contract.

2.2.1 Requisiti funzionali

- Gli utenti dell'ecosistema possono essere sia esseri umani che macchine e possono ricoprire il ruolo di produttore e/o di consumatore di dati.
- Gli utenti dell'ecosistema devono essere identificati univocamente tramite un'identità decentralizzata.

- Qualsiasi utente può sviluppare e pubblicare un'applicazione decentralizzata in grado di interagire con questo ecosistema. Al momento della pubblicazione, lo sviluppatore deve fornire tutte le informazioni sull'applicazione tra cui il nome, una breve descrizione e l'url da cui poterla scaricare.

- Gli utenti dell'ecosistema devono poter esplorare tutte le applicazioni pubblicate.

- Le applicazioni pubblicate che raccolgono ed elaborano i dati degli utenti produttori devono spiegare chiaramente le modalità in cui lo fanno. Per ogni operazione sui dati, lo sviluppatore deve specificare la modalità di accesso ai dati (privata, ristretta o aggregata), fornire una breve descrizione e il relativo codice software.

- L'utente viene considerato produttore quando scarica e utilizza un'applicazione che raccoglie e opera sui suoi dati.

- Le applicazioni possono operare sui dati dei produttori se e solo se questi hanno fornito esplicitamente l'autorizzazione per farlo.

- I produttori devono poter liberamente scegliere se fornire o meno l'autorizzazione a un'operazione che opera sui loro dati. In qualsiasi momento, il produttore può rimuovere un'autorizzazione che aveva concesso precedentemente.

- Quando un produttore fornisce l'autorizzazione a un'applicazione, implicitamente sta fornendo il consenso al suo sviluppatore.

- I produttori possono consentire l'accesso ai propri dati anche ad altri utenti che sono diversi dallo sviluppatore dell'applicazione usata per generare i dati in questione.

- I produttori devono poter accedere sempre ai dati che hanno prodotto usando le applicazioni.

- I consumatori possono accedere ai dati dei produttori se e solo se ne hanno l'autorizzazione e la modalità di accesso è ristretta. Possono accedere a una versione aggregata dei dati se e solo se hanno il permesso e la modalità di accesso è aggregata. In tutti gli altri casi, i consumatori non devono poter mai accedere ai dati altrui.

- Ogni operazione svolta sui dati da parte dei produttori e dei consumatori deve essere memorizzata nei log. In qualsiasi momento, i produttori devono poter visionare i log.

- Il sistema deve consentire la comunicazione tra le applicazioni decentralizzate e i canali dei dati. Questa comunicazione permette la scrittura e la lettura dei dati da parte dei produttori, nonché utenti delle applicazioni.

- Il sistema deve consentire la comunicazione tra le applicazioni decentralizzate e gli smart contract. Questa comunicazione permette il controllo delle autorizzazioni dei produttori, necessarie all'applicazione per comprendere se può o meno eseguire certe operazioni sui dati.

- Il sistema deve mettere a disposizione degli utenti dell'ecosistema una dApp di sistema. Quest'ultima agevola l'interazione con i canali di dati e log e con gli smart contract per gestire le autorizzazioni e le applicazioni pubblicate.

2.2.2 Requisiti non funzionali

- **Sicurezza**
  - **Confidenzialità e autenticità dei dati:** i dati prodotti vengono cifrati e firmati usando la crittografia asimmetrica.
  - **Controllo degli accessi:** i dati sono accessibili solo a coloro che hanno ricevuto l'autorizzazione da parte del loro produttore. Ogni accesso viene registrato in appositi log.

  - **Diversi livelli di autorizzazione:** i dati possono essere acceduti in diverse modalità: privata, ristretta e aggregata. A modalità diverse corrispondono differenti livelli di ristrettezza.
  - **Anonimato:** ogni utente è identificato solo tramite la sua identità decentralizzata, alla quale possono essere associate una o più chiavi pubbliche.

- **Affidabilità**
  - **Isolamento dei guasti:** in una rete decentralizzata p2p composta da molti nodi, il fallimento di un singolo nodo non interferisce con l'esecuzione degli altri.
  - **Alto numero di richieste:** è possibile gestire un elevato numero di richieste grazie ai nodi che contemporaneamente possono elaborare richieste differenti.
  - **Corruzione e perdita dei dati:** i dati sono replicati su ogni nodo; quindi, è praticamente impossibile corrompere o perdere i dati memorizzati nel registro distribuito.
  - **Disponibilità continua:** i sistemi decentralizzati, grazie alla replicazione, sono sempre disponibili.

- **Performance**
  - **Throughput dei dati:** i dati devono essere condivisi con un throughput sufficientemente elevato per ottenere buone performance sulle applicazioni usate dagli utenti.
  - **Conferma delle transazioni:** la DLT deve assicurare la conferma delle transazione in pochi secondi, riducendo i tempi di attesa per le applicazioni.
  - **Throughput delle transazioni:** la DLT deve riuscire a elaborare un numero sufficientemente elevato di transazioni al secondo per supportare la produzione multipla di dati da parte di più utenti.

- **Scalabilità:** il sistema deve essere in grado di scalare in base al numero variabile di richieste da parte degli utenti. Queste richieste comprendono sia operazioni sui dati che operazioni sugli smart contract.
- **Accessibilità:** il sistema deve essere facilmente accessibile ed esporre un'interfaccia user-friendly.

2.3 Casi d'uso

In questo paragrafo verranno analizzati i servizi che il sistema mette a disposizione dei propri utenti. Come viene mostrato nella Figura 2.2, alcuni servizi sono disponibili a tutti; mentre altri sono dedicati a una certa tipologia di utente (produttore o consumatore). Questi ultimi necessitano dell'autenticazione per essere eseguiti.

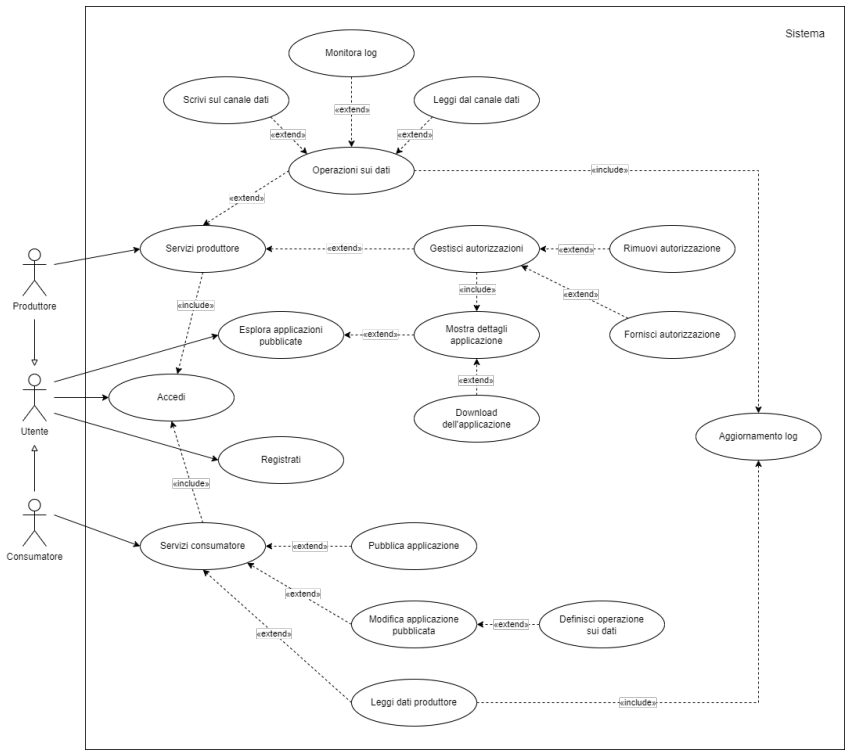

Figura 2.2: Diagramma dei casi d'uso

Approfondiamo i casi d'uso più rilevanti:

- **Accedi**
  - **Descrizione:** L'utente effettua l'autenticazione usando la sua identità decentralizzata.
  - **Attori:** Utente.
  - **Pre-condizioni:** - .
  - **Scenario principale:**
    1. La dApp di sistema presenta la schermata di login.
    2. L'utente inserisce la stringa che identifica la sua identità decentralizzata e la password scelta da lui al momento della registrazione.
    3. Il sistema verifica le credenziali inserite usando i dati salvati negli smart contract.
    4. Il controllo restituisce un esito positivo.
    5. La dApp mostra la schermata principale con tutte le applicazioni pubblicate.
  - **Post-condizioni:** L'utente ha eseguito l'autenticazione e può accedere a tutti i servizi dell'ecosistema.
- **Pubblica app**
  - **Descrizione:** Il consumatore pubblica la sua applicazione all'interno dello store di applicazioni dell'ecosistema.
  - **Attori:** Consumatore.
  - **Pre-condizioni:** Il consumatore ha effettuato l'autenticazione.
  - **Scenario principale:**
    1. La dApp di sistema presenta la schermata per pubblicare l'applicazione.

2. Il consumatore inserisce il nome, la descrizione e l'url per il download dell'applicazione.
3. Il consumatore conferma le informazioni inserite.
4. Il sistema verifica la correttezza delle informazioni inserite.
5. Il controllo restituisce un esito positivo.
6. Il sistema salva le informazioni dell'applicazione nell'apposito smart contract.
7. La dApp mostra la schermata dedicata all'applicazione appena pubblicata.

– **Post-condizioni:** L'applicazione è stata pubblicata sullo store di app dell'ecosistema, diventa disponibile durante l'esplorazione delle applicazioni pubblicate e può essere scaricata dagli utenti.

- **Definisci operazione sui dati**

  – **Descrizione:** Il consumatore definisce un'operazione di raccolta o di elaborazione dei dati prodotti dagli utenti che usano una sua applicazione pubblicata precedentemente.

  – **Attori:** Consumatore.

  – **Pre-condizioni:** Il consumatore ha effettuato l'autenticazione e ha pubblicato l'applicazione per cui vuole definire l'operazione sui dati.

  – **Scenario principale:**

    1. La dApp di sistema presenta la schermata per definire un'operazione sui dati.
    2. Il consumatore inserisce la descrizione e il codice dell'operazione, poi seleziona la modalità di accesso ai dati scegliendo tra: privata, ristretta o aggregata.
    3. Il consumatore conferma le informazioni inserite.
    4. Il sistema verifica la correttezza delle informazioni inserite.

5. Il controllo restituisce un esito positivo.
6. Il sistema salva le informazioni dell'operazione nell'apposito smart contract.
7. La dApp mostra la schermata dedicata alla modifica dell'applicazione per cui è stata definita l'operazione.

– **Post-condizioni:** L'operazione è definita e i produttori possono decidere se concederle o meno l'autorizzazione.

- **Fornisci autorizzazione**

  – **Descrizione:** Il produttore fornisce la sua autorizzazione a un'operazione di raccolta o di elaborazione dei suoi dati, generati usando una determinata applicazione.

  – **Attori:** Produttore.

  – **Pre-condizioni:** Il produttore ha effettuato l'autenticazione e si trova nella schermata che mostra i dettagli dell'applicazione.

  – **Scenario principale:**

    1. Il produttore seleziona un'operazione sui dati.
    2. La dApp di sistema mostra la schermata contenente i dettagli sull'operazione.
    3. Il produttore fornisce l'autorizzazione all'operazione.
    4. Il sistema salva la scelta del produttore nell'apposito smart contract.
    5. La dApp di sistema mostra la schermata aggiornata.

  – **Post-condizioni:** L'operazione riceve l'autorizzazione per essere eseguita.

- **Scrivi sul canale dati**

  – **Descrizione:** Il produttore salva sul canale dati delle informazioni che ha generato usando una determinata applicazione.

- **Attori:** Produttore.
- **Pre-condizioni:** Il produttore ha effettuato l'autenticazione e ha concesso l'autorizzazione all'operazione che deve effettuare la scrittura dei dati prodotti sul canale dati.
- **Scenario principale:**
  1. Il produttore usa un'applicazione sviluppata da un consumatore
  2. Il produttore genera dei dati che saranno gestiti da un'operazione di raccolta dati definita dallo sviluppatore dell'applicazione
  3. L'applicazione comunica con il sistema per controllare se l'operazione di raccolta ha l'autorizzazione per gestire i dati del produttore.
  4. Il sistema controlla gli smart contract per verificare se quello specifico produttore ha concesso o meno l'autorizzazione richiesta.
  5. Il sistema comunica l'esito del controllo all'applicazione.
  6. L'applicazione riceve un esito positivo.
  7. L'applicazione esegue l'operazione di raccolta e scrittura dati. Questi ultimi vengono cifrati e autenticati usando la coppia di chiavi (pubblica e privata) del produttore. I dati vengono scritti sul canale dati dedicato al produttore.
  8. Il canale dati aggiorna i riferimenti necessari per concatenare il prossimo blocco di dati.
  9. Sul canale log viene registrata l'operazione svolta memorizzando l'identificativo del produttore, il timestamp, il tipo di operazione svolta e il riferimento della transazione che contiene i dati scritti sul canale dati.
  10. L'applicazione mostra al produttore un messaggio che indica il successo dell'operazione di raccolta dati.

– **Post-condizioni:** I dati prodotti sono salvati sul canale dati e sono accessibili al produttore e ai consumatori autorizzati.

Capitolo 3

Architettura

I requisiti definiti nel precedente capitolo rappresentano la base di partenza per realizzare l'architettura dell'ecosistema. Quest'ultimo è composto da più ambienti che interagiscono fra di loro ma che non possono essere considerati dei veri e propri sottocomponenti; per questo motivo viene usato il termine “ecosistema” e non “sistema”. L'ecosistema punta a preservare le relazioni tra i vari ambienti, anche nel caso in cui le loro caratteristiche dovessero mutare nel corso del tempo.

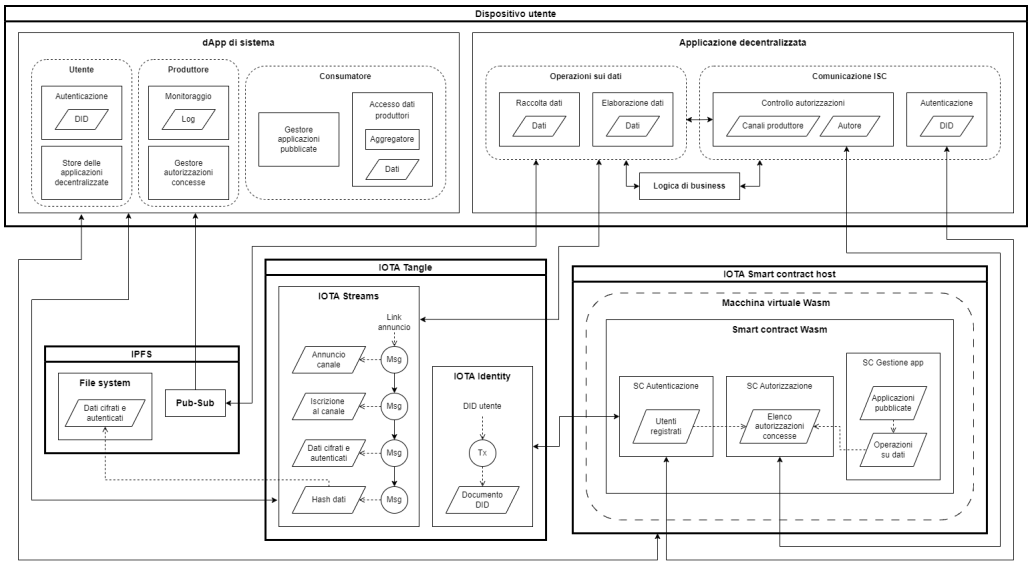

Figura 3.1: Rappresentazione dell'architettura dell'ecosistema

Come è possibile evincere dalla Figura 3.1, l'architettura è di tipo peer-to-peer (P2P). All'interno di questo schema decentralizzato, l'utente usa il suo dispositivo personale per connettersi e interagire con nodi facenti parte di diverse reti. Ad esempio, ci sono i nodi Hornet della Mainnet che elaborano le transazioni del Tangle, i nodi Wasp di Shimmer che eseguono gli smart contract e, infine, i nodi dell'IPFS (InterPlanetary File System) che gestiscono i file di grandi dimensioni. Pertanto, l'utente interagisce sempre con reti decentralizzate per scambiare dati e usufruire dei servizi offerti.

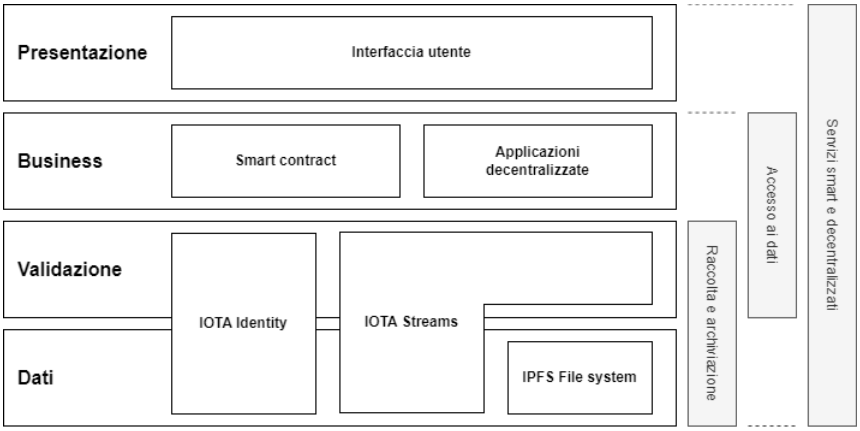

Figura 3.2: Rappresentazione dell'architettura a livelli dell'ecosistema

L'ecosistema può essere rappresentato persino tramite un'architettura a livelli:

- **Livello dati:** si occupa dell'archiviazione dei dati. Comprende il file system IPFS per i dati di grandi dimensioni, IOTA Streams per i dati più piccoli e IOTA Identity per i documenti DID che contengono le informazioni sulle identità decentralizzate.
- **Livello validazione:** dedicato alla validazione sia dei dati prodotti dagli utenti che delle credenziali verificabili per l'identificazione.
- **Livello logica di business:** include le funzionalità degli smart contract e i servizi offerti dalle applicazioni decentralizzate sviluppate dagli utenti.

- **Livello presentazione:** comprende l'interfaccia delle applicazioni usate dai produttori e l'interfaccia della dApp di sistema tramite la quale poter interagire con gli smart contract.

Ciò che l'ecosistema ha da offrire può essere raggruppato in tre aree:

- Raccolta e archiviazione dei dati.
- Accesso ai dati.
- Servizi smart.

Nei prossimi paragrafi queste aree verranno analizzate nel dettaglio.

3.1 Raccolta e archiviazione dei dati

L'ambiente dedicato alla raccolta e all'archiviazione dei dati è fondamentale per l'ecosistema. Senza di questo, non sarebbe possibile condividere ed elaborare i dati all'interno delle applicazioni.

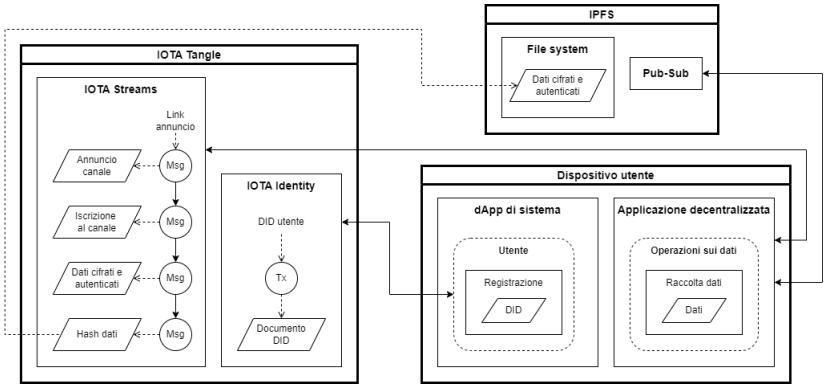

Figura 3.3: Rappresentazione dell'architettura per la raccolta e l'elaborazione dei dati

Naturalmente, una delle sue caratteristiche è quella di essere decentralizzato, differenziandosi da altri servizi simili centralizzati. Questi ultimi tendono ad appropriarsi dei dati, limitando il controllo del loro produttore.

Al contrario, l'ambiente di raccolta e archiviazione proposto è pensato per essere privo di entità centrali, lasciando al produttore il pieno controllo sulle informazioni prodotte. Come suggerisce il nome, le operazioni eseguite in questo ambiente sono principalmente due: raccolta e archiviazione.

3.1.1 Raccolta

La fase di raccolta ha il compito di raccogliere tutti i dati prodotti. Un utente diventa produttore quando genera dei dati usando le applicazioni decentralizzate sviluppate da altri utenti dell'ecosistema. Non a caso, in questa fase un ruolo primario è ricoperto dal dispositivo del produttore, che potrebbe essere un computer, uno smartphone, un dispositivo IoT o altro ancora. Indipendentemente dal tipo, i dati vengono prodotti tramite le interazioni dell'utente con l'applicazione e sfruttando le periferiche di input, come tastiere e sensori, per acquisire informazioni dall'ambiente esterno.

Ogni operazione di raccolta deve essere definita dallo sviluppatore, il quale specificherà in maniera dettagliata i dati che vuole acquisire e per quale scopo. Di conseguenza, lo sviluppatore ricoprirà anche il ruolo di consumatore. Chiaramente, il produttore dovrà fornire la propria autorizzazione per far sì che la raccolta dei dati possa effettivamente essere eseguita dall'applicazione. Tutte le informazioni collezionate vengono momentaneamente memorizzate nella memoria dell'applicazione, per poi essere archiviate definitivamente nella rete decentralizzata.

3.1.2 Archiviazione

I dati acquisiti devono essere archiviati in maniera tale da consentire l'accesso a chiunque abbia i diritti per farlo. La memorizzazione e la validazione di queste informazioni avviene attraverso l'utilizzo di IOTA e IPFS. Entrambe le tecnologie duplicano i dati su più nodi facenti parte di una rete P2P dedicata. Così facendo, l'accesso può essere gestito senza dipendere da

sistemi centralizzati, assicurando al produttore il pieno controllo sulla propria produzione.

3.1.2.1 Canali IOTA Streams

I dati sono principalmente archiviati sul Tangle di IOTA. Le informazioni memorizzate su quest'ultimo diventano immutabili; pertanto, non possono essere né censurate né rimosse e saranno sempre disponibili fino a quando il Tangle sarà online. I dati pubblicati sono pubblicamente accessibili da chiunque. Di conseguenza, se si volesse limitare l'accesso solo a chi ne ha il diritto, bisognerebbe cifrarli.

IOTA Streams è un framework usato per pubblicare dati sul Tangle, garantendo la loro riservatezza e autenticità. Più nello specifico, i dati sono ordinati in ordine cronologico all'interno di una struttura logica ramificata, chiamata canale. Su ogni canale operano due figure: autore e iscritto. L'autore crea il canale e gestisce le richieste di iscrizione. L'iscritto può accedere al canale e al suo contenuto solo dopo aver ricevuto l'autorizzazione dell'autore, ovvero a seguito dell'approvazione della sua richiesta di iscrizione. A ogni canale viene associato un link pubblico che può essere usato come punto di accesso o come indirizzo a cui gli utenti possono inviare le richieste di iscrizione.

All'interno dell'ecosistema proposto, il produttore ricopre il ruolo di autore e i consumatori quello di iscritti. Quando un produttore fornisce la sua autorizzazione a un'operazione di raccolta, avviene la creazione di due canali: uno per i dati e uno per registrare i relativi log. Il produttore, essendo anche autore, detiene il pieno controllo sui canali e decide quali consumatori hanno il diritto di accedere ai suoi dati. Ogni consumatore che vuole ottenere l'accesso deve implicitamente o esplicitamente inviare la richiesta per iscriversi al canale. L'applicazione, che esegue l'operazione di raccolta dati sul dispositivo del produttore, scrive le informazioni raccolte direttamente sul canale dati usando l'oggetto autore associato al produttore. Così facendo, IOTA Streams si occuperà autonomamente di firmare digitalmente i dati con

la chiave privata del produttore, garantendo l'autenticità dell'informazione. Inoltre, i dati saranno cifrati con una chiave simmetrica condivisa tra tutti i partecipanti al canale, assicurando la loro riservatezza. Infine, i dati firmati e cifrati verranno racchiusi in un nuovo messaggio che sarà collegato all'ultimo messaggio pubblicato sul canale. La pubblicazione sul canale, naturalmente, sottintende la pubblicazione sul Tangle.

3.1.2.2 Oggetti IPFS

I dati di grandi dimensioni, a differenza di quelli più piccoli, non possono essere archiviati direttamente sul Tangle. Infatti, servirebbero troppi messaggi per memorizzare interamente un'immagine, un video o un audio all'interno di un canale IOTA Streams, diventando così un'operazione inefficiente. Di conseguenza, questo tipo di dati dovranno essere archiviati all'interno di un file system come IPFS.

Ogni dato è rappresentato sottoforma di oggetto IPFS ed è identificato tramite il Content Identifier (CID). Quest'ultimo è un digest ottenuto da una funzione di hash eseguita sul contenuto del dato da memorizzare. Grazie alla mancanza del PoW, l'oggetto IPFS diventa subito disponibile a tutti i peer della rete non appena viene salvato. Sia l'archiviazione dell'oggetto IPFS sia la pubblicazione del suo CID nel canale dati sono operazioni a carico del produttore del dato. I consumatori potranno accedere a questo dato solo dopo aver ottenuto il suo CID dall'apposito messaggio pubblicato sul canale.

3.1.2.3 Documenti DID

Ogni utente dell'ecosistema deve essere identificato univocamente tramite la sua identità decentralizzata. Per far ciò, l'ecosistema si serve del framework IOTA Identity, che è in grado di rilasciare le DID (Decentralized Identifiers) e controllare la loro validità. In realtà, la DID non è altro che un puntatore per ottenere il Documento DID associato. Quest'ultimo contiene varie informazioni sull'utente, tra cui le chiavi pubbliche usate per verificare i dati firmati digitalmente da lui e diversi URI che possono puntare a ul-

teriori informazioni personali. Quando un utente si registra e riceve la sua identità digitale, il relativo Documento DID viene generato e archiviato sul Tangle in maniera tale da renderlo immutabile e pubblicamente accessibile. Il Documento DID non contiene informazioni riservate; di conseguenza può essere archiviato senza la necessità di cifrarlo.

L'ambiente di raccolta e archiviazione si occupa anche di gestire la produzione e la memorizzazione dei dati collegati alle identità digitali. Identificare un utente è un compito importante all'interno dell'ecosistema perché consente il riconoscimento dei produttori, dei consumatori e la verifica dei diritti a loro associati. Pertanto, è importante curare la raccolta e l'archiviazione di queste informazioni al pari dei dati generati dai produttori.

3.2 Accesso ai dati

L'ecosistema nasce per condividere ed elaborare i dati prodotti utilizzando le applicazioni decentralizzate. Pertanto, l'ambiente che si occupa dell'accesso ai dati svolge un ruolo centrale nella mediazione delle richieste dei consumatori che vogliono usufruire delle informazioni dei produttori.

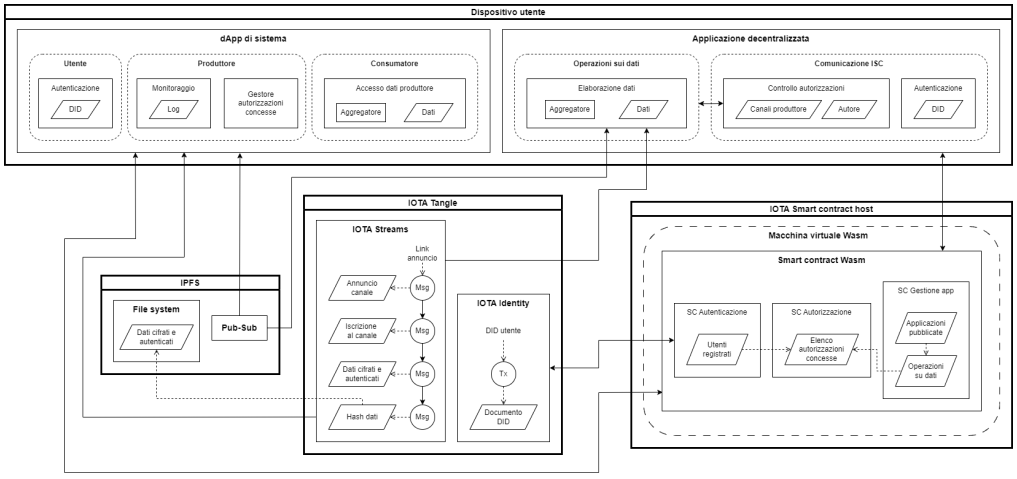

Figura 3.4: Rappresentazione dell'architettura per l'accesso ai dati

Per accedere ai dati archiviati sul canale dati e nel file system IPFS, il consumatore deve ricevere l'autorizzazione da parte del loro proprietario. Il meccanismo delle autorizzazioni viene completamente gestito tramite gli IOTA Smart Contracts. Infatti, la dApp di sistema è necessaria proprio per semplificare le interazioni con gli smart contract e far sì che i produttori possano facilmente gestire le operazioni sui dati a cui hanno fornito il consenso. I dati possono essere acceduti direttamente dal produttore e dai consumatori autorizzati tramite la dApp di sistema; oppure possono essere recuperati dall'applicazione decentralizzata durante l'esecuzione di un'operazione di elaborazione dati.

3.2.1 Smart contract

Gli smart contract usati nell'ecosistema sono gli IOTA Smart Contracts. Questi sono codificati nel linguaggio di programmazione Wasm e vengono eseguiti su un'apposita macchina virtuale. Per gestire gli accessi ai dati, l'ambiente si serve di tre smart contract: Autenticazione, Autorizzazione e Gestione app.

Il primo ha il compito di autenticare gli utenti tramite la loro DID. Riconoscere un utente è fondamentale per poter risalire alle autorizzazioni che ha concesso e che gli sono state concesse.

Lo smart contract Gestione app si occupa di gestire le applicazioni e tutte le relative operazioni sui dati. Tra queste ultime sono presenti anche le operazioni che hanno la necessità di accedere ai dati prodotti per elaborarli. Tutti i dettagli operativi sono specificati dallo sviluppatore dell'applicazione e vengono memorizzati all'interno dello smart contract.

L'ultimo, ma non per importanza, è lo smart contract Autorizzazione. Questo è pensato per gestire e memorizzare al suo interno l'insieme di autorizzazioni che un utente fornisce alle varie operazioni sui dati. Per funzionare correttamente, ha bisogno degli altri due smart contract. Infatti, solo un utente registrato nell'ecosistema può fornire l'autorizzazione a un'operazione sui dati esistente e associata a un'applicazione pubblicata. Le autorizzazioni

concesse sono salvate nello stato interno dello smart contract.

Gli utenti possono interagire con gli smart contract tramite la dApp di sistema. Quest'ultima consente l'autenticazione dell'utente, la gestione delle sue autorizzazioni e persino l'accesso ai dati. Allo stesso modo, le operazioni di elaborazione dati, interne alle applicazioni decentralizzate, possono comunicare con gli host ISC per interrogare gli smart contract e verificare la concessione delle autorizzazioni richieste.

3.2.2 Accesso ai canali dati e log

I dati sono archiviati sul canale IOTA Streams e sul file system IPFS. Le informazioni sul canale dati possono essere accedute solo dall'autore e dagli iscritti. Se un consumatore è iscritto al canale, vuol dire che ha ricevuto l'autorizzazione da parte dell'autore, ovvero dal produttore dei dati presenti sul canale.

L'accesso al canale avviene tramite due particolari oggetti IOTA Streams: Author e Subscriber, rispettivamente usati dall'autore e dagli iscritti del canale. Questi oggetti sono memorizzati all'interno dello smart contract Autorizzazione e vengono forniti all'utente che ne fa richiesta solo dopo essere stato identificato e aver controllato se possiede le autorizzazioni necessarie. In caso di esito positivo, l'utente riceve l'oggetto Streams (Author o Subscriber) per accedere al canale ed effettua la sua lettura. In caso contrario, l'utente non riceverà l'oggetto Streams e non potrà accedere al canale perché gli mancheranno i riferimenti necessari.

L'accesso agli oggetti IPFS segue lo stesso procedimento inizialmente. In questo caso, l'utente legge dal canale dati il CID che poi userà per accedere all'oggetto IPFS archiviato sul file system distribuito.

Ogni operazione di accesso ai dati deve essere sempre registrata all'interno del canale log, creato in concomitanza al canale dati. Similmente a quest'ultimo, il canale log è un canale IOTA Streams il cui autore è sempre il produttore e i consumatori sono gli iscritti. A differenza dell'altro canale, gli iscritti al canale log possiedono il diritto di scrittura sul canale, proprio

per registrare i loro accessi.

L'accesso ai dati può avvenire secondo diverse modalità: privata, ristretta o aggregata. Per l'ultima tipologia, sia la dApp di sistema che le applicazioni decentralizzate devono usare l'aggregatore, ovvero un componente che ha il compito di leggere tutti i singoli dati e di mostrarli al consumatore esclusivamente in forma aggregata. Il produttore, invece, potrà accedere ai dati singolarmente, indipendentemente dalla modalità scelta.

3.2.3 Identificazione con credenziali verificabili

L'ambiente sotto analisi gestisce anche l'accesso alle informazioni personali degli utenti. Ad esempio, un'app di fitness potrebbe richiedere all'utente di specificare l'altezza, il peso e l'età. Diversamente, un'app bancaria potrebbe necessitare dell'iban e del codice fiscale. Queste informazioni personali vengono richieste da applicazioni differenti ma sono associate allo stesso utente, quindi alla stessa DID.

Questo tipo di informazioni sono memorizzate all'interno delle credenziali verificabili. Queste ultime contengono il riferimento alla DID di appartenenza e tutte le informazioni personali sull'utente. L'applicazione le rilascia quando l'utente si registra e verifica la loro correttezza quando effettua il login. Di conseguenza, allo stesso utente possono essere associate più credenziali verificabili, una per ogni applicazione usata. L'entità che si occupa del rilascio e della verifica prende il nome di Issuer ed è integrata nell'applicazione.

La segretezza delle informazioni contenute nelle credenziali verificabili è garantita dal fatto che sono salvate sul dispositivo in forma cifrata. Inoltre, l'applicazione può usarle solo localmente, senza mai condividerle all'esterno.

3.3 Servizi smart

L'ecosistema è in grado di offrire diversi servizi smart basati sull'infrastruttura vista nei precedenti paragrafi.

Prima di tutto, l'ecosistema può essere considerato come un vero e pro-

prio store di applicazioni, al pari delle controparti centralizzate come Google Play, App Store e Microsoft Store. In questo caso, agli utenti viene offerta la possibilità di pubblicare le applicazioni che sviluppano e di esplorare le applicazioni altrui già pubblicate.

Inoltre, l'ecosistema è pensato per offrire agli sviluppatori un insieme di funzionalità che agevolano la raccolta, l'archiviazione, l'accesso e l'elaborazione dei dati prodotti tramite le loro applicazioni.

3.3.1 Store di applicazioni decentralizzate

L'ecosistema offre ai suoi utenti la possibilità di pubblicare le applicazioni decentralizzate sviluppate da loro. Durante la pubblicazione, lo sviluppatore deve inserire un insieme di informazioni sull'applicazione che verranno poi memorizzate negli smart contract. Tra i dati forniti è presente anche l'url a cui collegarsi per scaricare e installare l'applicazione. Questa può essere memorizzata in una repository personale, su un sito web, all'interno di un file system IPFS e così via.

Grazie al fatto che le informazioni delle app sono salvate all'interno degli smart contract, gli utenti dell'ecosistema possono facilmente esplorare tutte le applicazioni pubblicate per decidere quale scaricare e usare. Gli utenti possono consultare persino le relative operazioni sui dati, decidendo se concedere o meno l'autorizzazione.

Sia la pubblicazione sia l'esplorazione avvengono tramite la dApp di sistema perché è necessario interagire con gli smart contact.

3.3.2 Supporto agli sviluppatori

Generalmente, quando si sviluppa un'applicazione centralizzata, lo sviluppatore deve preoccuparsi di cercare e configurare un database, curare la sicurezza dei dati, gestire le informazioni personali degli utenti secondo le normative vigenti e pagare per pubblicare la sua applicazione.

Questo ecosistema non solo nasce per conferire ai produttori maggiore

controllo sui loro dati; ma anche per fornire degli strumenti in grado di semplificare il lavoro degli sviluppatori. Questi ultimi, nella maggior parte dei casi, possono eseguire le operazioni sui dati semplicemente integrando alcuni framework di IOTA oppure effettuando delle richieste HTTP al nodo Wasp su cui eseguono gli smart contact. I framework IOTA forniscono potenti strumenti per gestire i dati e le identità decentralizzate, togliendo allo sviluppatore l'onere di cercare, configurare e gestire in maniera sicura un database o un altro supporto simile. Inoltre, grazie agli smart contract è possibile ricreare le stesse funzionalità dei classici store di applicazioni, con il vantaggio di essere tutto gratuito per gli sviluppatori. Tutto ciò che viene offerto dall'ecosistema ha performance, disponibilità, resilienza, scalabilità e affidabilità pari ai competitor centralizzati.

Capitolo 4

Implementazione

Dopo aver chiarito il focus principale di questo lavoro e aver analizzato l'architettura dell'ecosistema, in questo capitolo verrà trattata la sua implementazione.

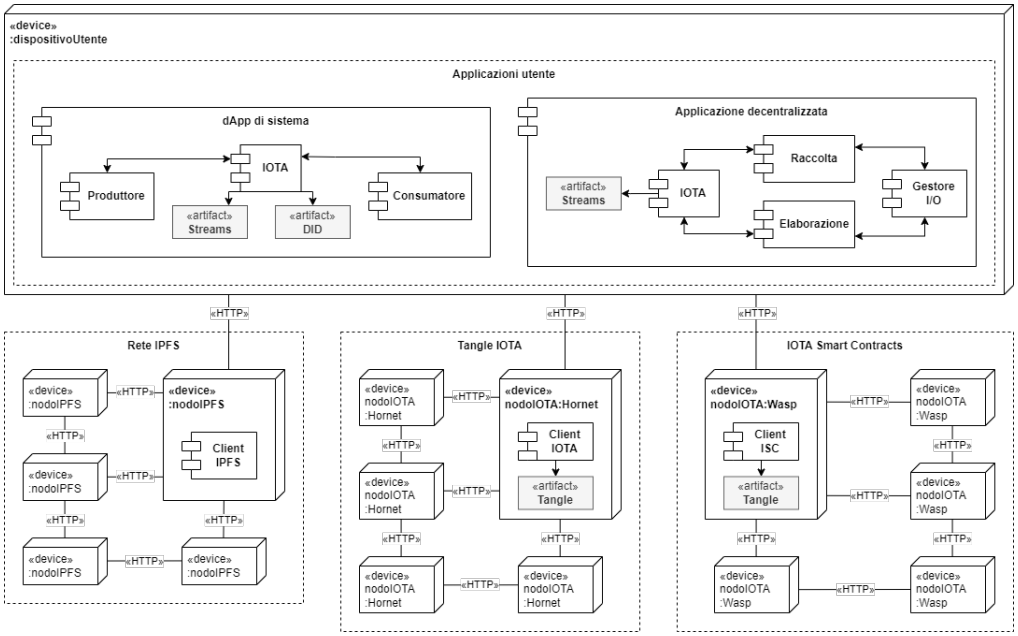

Figura 4.1: Diagramma di deployment dell'ecosistema

La Figura 4.1 illustra il diagramma di deployment dell'ecosistema, con una particolare attenzione alle applicazioni utente (dApp di sistema e gene-

rica applicazione decentralizzata) e alle reti di nodi IPFS e IOTA, fondamentali per la decentralizzazione e il funzionamento complessivo.
Nei prossimi paragrafi saranno analizzati i canali di comunicazione tra le applicazioni utente e i seguenti componenti:

- **Tangle IOTA:** comunicazione con i nodi Hornet della rete IOTA per archiviare, validare e leggere i dati dei produttori. In questa comunicazione rientra anche la pubblicazione dei documenti DID delle identità decentralizzate.
- **IPFS:** comunicazione con i nodi IPFS per archiviare dati di grandi dimensioni.
- **IOTA Smart Contracts:** comunicazione con i nodi Wasp della rete IOTA per interagire con gli smart contract Wasm.

Infine, l'ultima parte del capitolo sarà dedicata all'implementazione della dApp di sistema e di un'applicazione decentralizzata, mettendo in risalto i pregi e le limitazioni delle tecnologie adottate.

4.1 Tangle IOTA

Per pubblicare un dato sul Tangle di IOTA, l'applicazione utente deve creare una transazione e poi inviarla in broadcast ai nodi della rete IOTA. Per eseguire queste operazioni bisogna integrare i giusti framework all'interno dell'applicazione: IOTA Streams per i dati e IOTA Identity per le identità. In entrambi i casi, è possibile suddividere il processo di pubblicazione in due parti: creazione e propagazione della transazione. La prima è eseguita completamente nell'applicazione; mentre la seconda inizia dal dispositivo utente e prosegue su tutti i nodi IOTA.

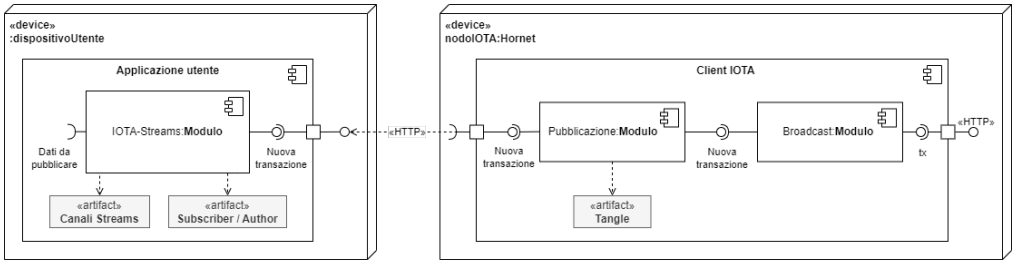

Figura 4.2: Comunicazione tra l'applicazione utente e il nodo Hornet per la pubblicazione di un dato sul canale Streams

4.1.1 Creazione e propagazione della transazione

Affinché la pubblicazione sul Tangle avvenga correttamente, la transazione deve essere correttamente creata e inviata in broadcast a tutti i nodi della rete IOTA.

4.1.1.1 Creazione

La creazione di una transazione è a carico dei framework integrati nell'applicazione utente. IOTA Identity si occupa della creazione di una semplice transazione contenente il documento DID, ovvero una serie di informazioni in formato JSON sull'identità decentralizzata. Invece, IOTA Streams non solo cura la creazione del messaggio da pubblicare, ma lo persino cifra e autentica. In seguito, verrà analizzato quest'ultimo caso proprio per la sua complessità maggiore.

Prima di pubblicare i dati, il modulo di raccolta deve assicurarsi di aver ricevuto l'autorizzazione dal produttore. Questo controllo viene fatto interrogando gli ISC. Nel caso in cui l'esito fosse positivo, il modulo di raccolta riceverebbe i riferimenti del canale Streams sui cui pubblicare i dati e un oggetto Author impiegato proprio per la creazione delle transazioni.
Il processo di creazione è mostrato nella Figura 4.3 ed è composto dai seguenti punti:

1. Ipotizzando di aver ricevuto l'autorizzazione necessaria, i dati vengono raccolti dal gestore dei dispositivi di I/O e inviati al modulo di raccolta.

2. Il modulo di raccolta associa i dati ricevuti alla rispettiva operazione per comprendere quale oggetto Author usare per la creazione del messaggio Stream e su quale canale scriverlo.

3. Viene controllata la dimensione del dato. Se è troppo grande, il dato dovrà essere memorizzato nel file system IPFS e il messaggio conterrà solo il relativo Content Identifier (CID). Altrimenti, il dato potrà essere interamente inserito all'interno del messaggio.

4. Le informazioni da scrivere nel messaggio vengono convertite in byte. Questi ultimi saranno cifrati con la chiave simmetrica condivisa tra gli iscritti al canale, ottenendo il payload cifrato.

5. Il nuovo messaggio dovrà collegarsi in coda a quelli già pubblicati sul canale. Di conseguenza, bisogna specificare l'indirizzo dell'ultimo messaggio del canale.

6. Il messaggio viene firmato digitalmente con la chiave privata del produttore.

7. Il messaggio Streams così ottenuto rappresenta la transazione da pubblicare sul Tangle.

8. Il processo di creazione si conclude con l'esecuzione del PoW.

4.1.1.2 Propagazione

Il passo successivo alla creazione di una transazione è la sua propagazione. L'applicazione utente invia una richiesta HTTP a un load balancer della rete IOTA. Quest'ultimo, innanzitutto, effettua una serie di controlli sintattici e semantici sulla transazione ricevuta. Nel caso in cui dovessero esserci errori, la transazione verrebbe immediatamente scartata. Altrimenti, se la transazione risultasse valida, il nodo Hornet procederebbe con la sua propagazione. La transazione verrebbe inviata a tutti i nodi della rete prossimi al nodo di partenza. Successivamente, ogni nodo invierebbe la transazione ricevuta al

proprio vicinato e aggiornerebbe la sua versione del Tangle. Infine, la transazione verrebbe confermata dal consenso distribuito raggiunto tra i nodi, anche grazie all'aiuto fornito dal Coordinatore.

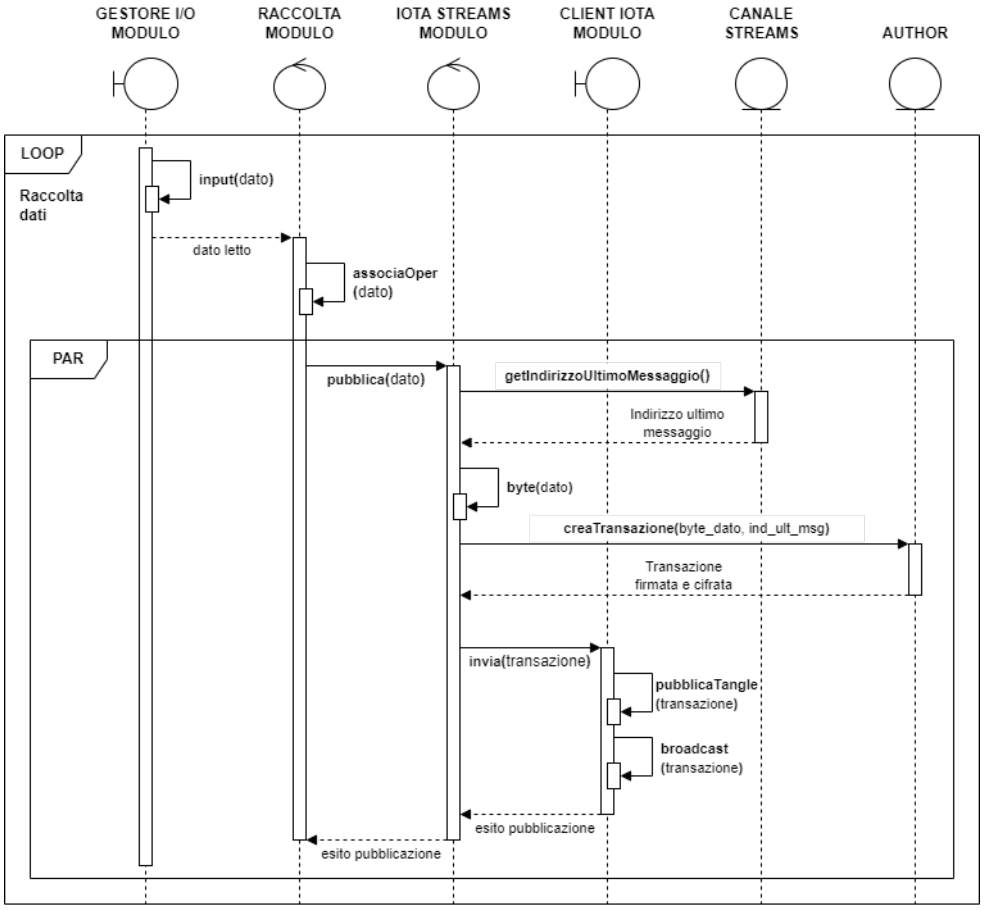

Figura 4.3: Diagramma di sequenza per la pubblicazione di dati di piccole dimensioni

4.1.2 Struttura del canale Streams

La pubblicazione e la lettura delle transazioni dai canali Streams viene fatta sia dalla dApp di sistema che dall'applicazione decentralizzata. Entrambe non lavorano solamente sul canale dati, ma anche su quello dei log. Nel primo sono archiviati tutti i dati prodotti da un utente usando le ap-

plicazioni decentralizzate sviluppate dai consumatori. Nel secondo vengono memorizzati i log riguardanti le operazioni di raccolta, accesso ed elaborazione eseguite dagli utenti.

I due canali svolgono funzioni diverse ma sono pur sempre canali Streams con la stessa struttura. Ambedue hanno un autore e degli iscritti. Questi ruoli sono gestibili rispettivamente tramite gli oggetti Author e Subscriber che, uniti all'indirizzo dell'ultimo messaggio pubblicato, devono essere conservati all'interno di un apposito smart contract. Quest'ultimo ha il compito di fornire queste informazioni solo alle operazioni che ne hanno l'autorizzazione, evitandone l'abuso.

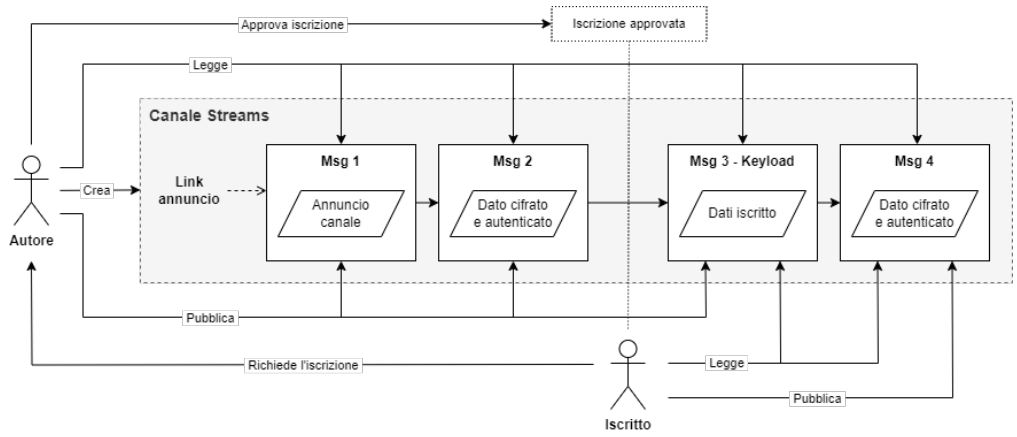

Figura 4.4: Rappresentazione della struttura e del funzionamento del canale Streams

Il primo messaggio di ogni canale Streams viene pubblicato dall'autore, è indirizzabile mediante il cosiddetto link di annuncio e contiene un insieme di informazioni necessarie agli utenti per proporre la loro richiesta di iscrizione al canale. Quando l'autore riceve e approva una richiesta di iscrizione, pubblica un messaggio keyload che serve per gestire gli accessi al canale. Infatti, l'utente appena iscritto potrà leggere solamente i messaggi successivi a quello di keyload contenente la sua chiave pubblica, ovvero i messaggi pubblicati dopo la sua iscrizione al canale. Dal canale è possibile leggere tutti i messaggi

successivi al keyload di riferimento oppure uno specifico messaggio fornendo il suo indirizzo.

4.2 IPFS

Il file system IPFS memorizza tutti i dati di grandi dimensioni condivisi dagli utenti ed è possibile accedervi direttamente senza la necessità di eseguire alcun PoW. I dispositivi dell'utente potrebbero essere persino usati come dei nodi IPFS in maniera tale da avere una copia del dato sempre disponibile, rimuovendo la completa dipendenza dai nodi esterni.

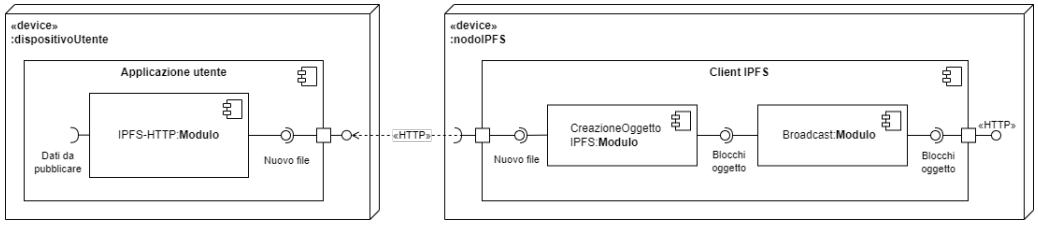

Figura 4.5: Comunicazione tra l'applicazione utente e il nodo IPFS per la pubblicazione di un dato di grandi dimensioni

I dati di grandi dimensioni da pubblicare vengono inseriti all'interno di un file che poi sarà inviato al nodo IPFS di riferimento. Quest'ultimo si preoccupa della creazione dell'oggetto IPFS, composto da un insieme di blocchi, e della successiva propagazione agli altri nodi della rete. La condivisione dei blocchi tra più nodi garantisce la sua persistenza. Una volta archiviato, il dato sarà disponibile fino a quando i nodi che lo contengono saranno online. Per accedere al file basterà usare il relativo CID, memorizzato sul canale dati e ottenuto eseguendo la funzione di hash sul dato stesso.

4.2.1 Preparazione e persistenza dell'oggetto IPFS

L'applicazione decentralizzata non si occupa direttamente di creare l'oggetto IPFS, ma solo di inviare il file cifrato a un nodo IPFS. Quest'ultimo

si farà carico della creazione dell'oggetto IPFS e della relativa propagazione. Grazie a ciò, il dispositivo utente dovrà preoccuparsi solamente della comunicazione con la rete IPFS.

4.2.1.1 Preparazione del file

Se durante l'operazione di raccolta venisse appurato che il dato è troppo grande per essere salvato sul canale Streams, allora bisognerebbe memorizzarlo sul file system IPFS. Il dato, che d'ora in poi sarà considerato come file, deve essere cifrato. Questo è necessario per proteggere le informazioni, visto che saranno pubblicamente accessibili una volta pubblicate. Il modo migliore per cifrare il dato è quello di generare una chiave simmetrica e inserirla all'interno del messaggio contenente il suo CID. Così facendo, solo gli iscritti al canale che possono accedere al dato, tramite il suo identificativo, potranno anche decifrarlo. Una volta cifrato, il file viene inviato a un nodo IPFS tramite una richiesta HTTP POST.

Per quanto riguarda la cifratura, un'alternativa poteva essere quella di impiegare la chiave condivisa tra gli iscritti al canale come chiave simmetrica. Purtroppo, questa chiave non è accessibile direttamente, quindi è inutilizzabile. Anche altre alternative basate sulla cifratura asimmetrica usando le chiavi pubbliche dei consumatori sarebbero state troppo costose e poco efficienti.

4.2.1.2 Persistenza dell'oggetto IPFS

Una volta ricevuto il file cifrato dal dispositivo utente, il nodo IPFS deve occuparsi dell'archiviazione nel file system e dell'invio in broadcast agli altri nodi della rete. Per prima cosa, il file viene convertito in oggetto IPFS. Quest'ultimo è composto da un insieme di blocchi, accessibili tramite l'apposito CID. Successivamente, tutti i blocchi verranno trasmessi in broadcast agli altri nodi IPFS, seguendo un protocollo di gossip simile a quello già visto per i nodi Hornet. La distribuzione del dato su più nodi favorisce la sua persistenza. Purtroppo, in alcuni casi potrebbe capitare che i nodi IPFS eliminino

i dati meno richiesti per far spazio ai nuovi. Per questo motivo, il produttore dovrebbe sempre avere una copia locale dei dati di questa tipologia.

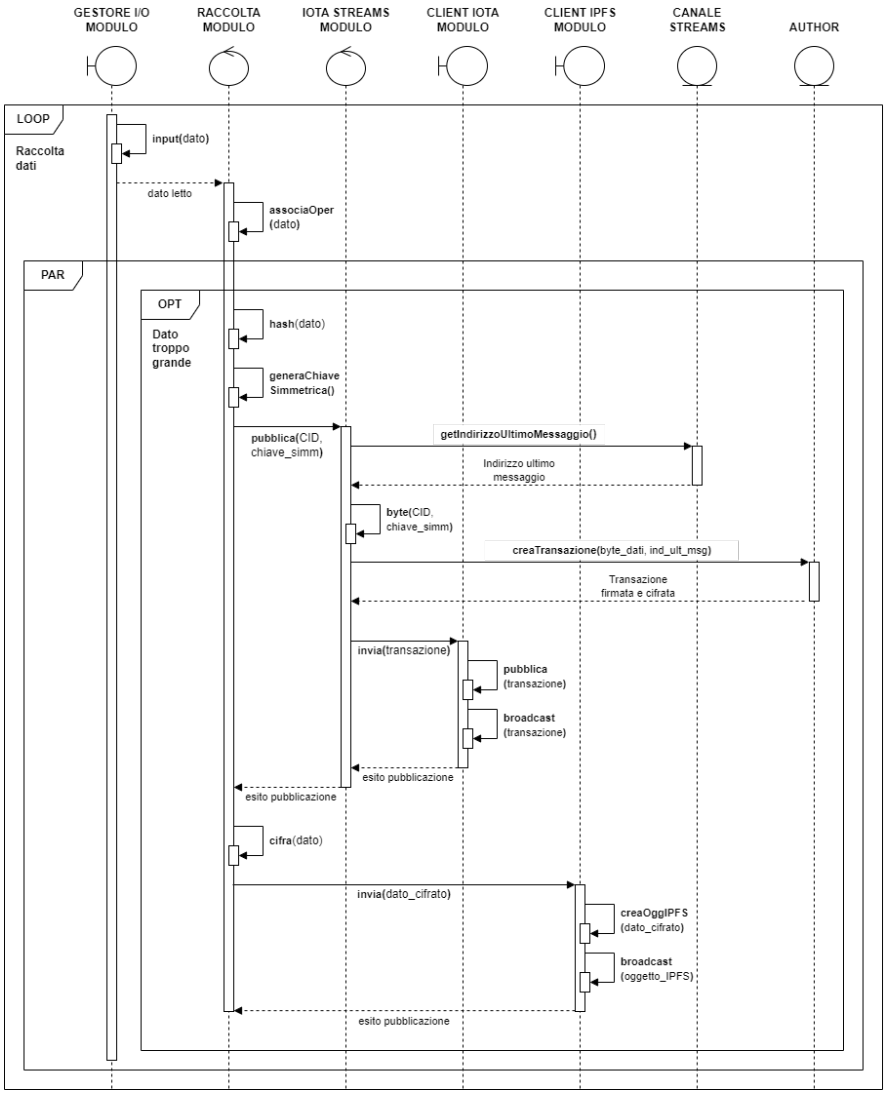

Figura 4.6: Diagramma di sequenza per la pubblicazione di dati di grandi dimensioni

4.3 IOTA Smart Contracts

Gli IOTA Smart Contracts sono fondamentali per la logica di business dell'ecosistema. Tramite questi è possibile gestire l'autenticazione degli utenti, le autorizzazioni fornite, le informazioni sulle applicazioni pubblicate e sulle relative operazioni sui dati. Grazie agli smart contract è possibile implementare un ecosistema privo di autorità centralizzate.

Il deployment degli ISC avviene sui nodi Wasp della rete IOTA, più nello specifico all'interno della macchina virtuale Wasm. Ogni smart contract è scritto con il linguaggio di programmazione Rust però, prima del deployment, viene convertito in linguaggio Wasm sfruttando lo strumento Schema Tool.
L'interazione tra le applicazioni utente e gli ISC avviene principalmente tramite richieste HTTP, ma può variare in base al tipo di funzione chiamata:

- **Func:** consente di modificare lo stato interno dello smart contract. In questo caso, l'applicazione utente può chiamare la funzione tramite richieste on-ledger e off-ledger.
- **View:** accede allo stato interno dello smart contract solo in lettura. Questo tipo di funzione può essere chiamata solo usando richieste off-ledger.

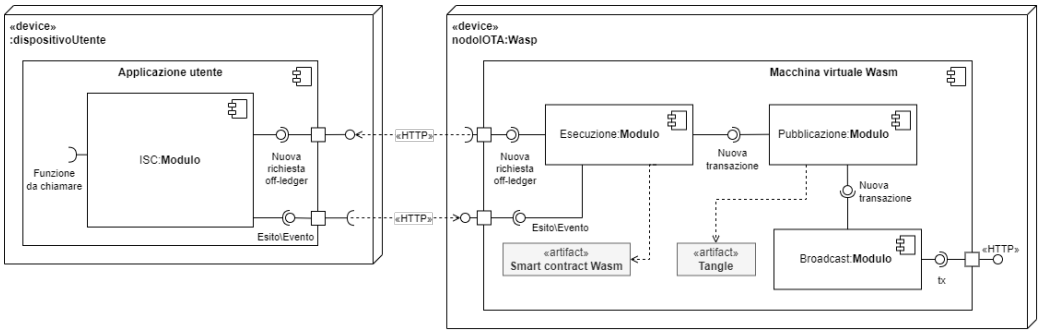

Figura 4.7: Comunicazione tra l'applicazione utente e il nodo Wasp per chiamare le funzioni degli smart contract

4.3.1 Smart contract

Per gestire l'autenticazione degli utenti, le autorizzazioni fornite e le applicazioni pubblicate con le relative operazioni sui dati, l'ecosistema si serve di tre smart contract: Autenticazione, Autorizzazioni e GestioneApplicazioni. Per implementare uno smart contract la prima cosa da fare è definire le Func, le View, le variabili di stato e le strutture dati all'interno di un file Yaml. Lo schema creato sarà sottoposto allo Schema Tool per generare dei template di smart contract. Schema Tool si occupa persino di creare le interfacce necessarie per gestire le richieste degli utenti. Grazie a ciò, lo sviluppatore potrà preoccuparsi esclusivamente dell'implementazione della logica delle funzioni. In questo progetto, le funzioni sono codificate in Rust ma potrebbero essere implementate anche in linguaggio Go o TypeScript.

Dopo aver definito gli smart contract, questi dovranno eseguire sulla macchina virtuale Wasm all'interno di un nodo Wasp. Essendo gli IOTA Smart Contracts ancora in fase sperimentale, non esistono nodi Wasp pubblici; quindi, per poterli usare bisogna scaricare e installare un nodo Wasp locale che sia in grado di comunicare con la rete IOTA pubblica. Questo procedimento sicuramente disincentiva l'adozione degli ISC a favore dei più consolidati e popolari smart contract Ethereum scritti in Solidity. La scelta di usare gli ISC in questo progetto è dovuta ai vantaggi che essi offrono: scalabilità, fee quasi nulle, catene personalizzate e flessibilità.

4.3.1.1 Autenticazione

Il primo smart contract a essere analizzato è quello dedicato all'autenticazione degli utenti. Ogni utente che vuole usufruire dei servizi smart dell'ecosistema deve essere identificato tramite un'identità decentralizzata.

È presente solo l'attributo utenti, ovvere una hash map contenente tutti gli utenti registrati nell'ecosistema. La chiave è rappresentata dalla DID dell'utente e punta all'hash della password scelta dall'utente durante la fase di registrazione.
Invece, i metodi sono i seguenti:

- **registrazione:** registra l'utente all'interno dell'ecosistema. La creazione della DID e la pubblicazione del documento DID avvengono sul dispositivo dell'utente. Successivamente, la DID e l'hash della password scelta dall'utente vengono inviate allo smart contract come parametri, per poi essere memorizzati all'interno dell'attributo utenti.
- **login:** effettua il login dell'utente usando la relativa DID e password.
- **eliminazione:** elimina un account.

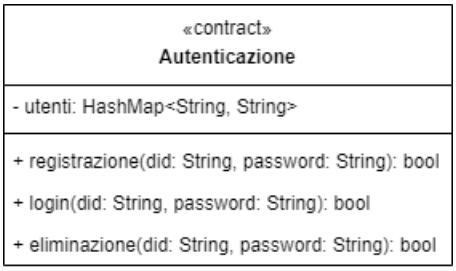

Figura 4.8: Diagramma UML dello smart contract Autenticazione

Questo smart contract contribuisce all'autenticazione decentralizzata, già in parte supportata con la scelta di impiegare l'identità decentralizzate per identificare gli utenti.

4.3.1.2 Autorizzazioni

Lo smart contract Autorizzazioni è pensato per memorizzare e gestire tutte le autorizzazioni che i produttori forniscono ai consumatori per una specifica operazione dati appartenente a una determinata applicazione.

Le informazioni delle autorizzazioni sono organizzate all'interno della struttura Autorizzazione. A ogni utente produttore devono essere associate tutte le autorizzazioni che ha concesso. Questa associazione viene effettuata grazie all'impiego dell'attributo autorizzazioni, ovvero di una HashMap dove alla DID del produttore si associa l'elenco delle sue autorizzazioni.
I metodi disponibili sono:

- **forniscAutorizzazione:** utente produttore fornisce la sua autorizzazione a un utente consumatore per un'operazione dati di una particolare applicazione.
- **rimuoviAutorizzazione:** rimuove un'autorizzazione precedentemente concessa.
- **controllaAutorizzazione:** controlla se un'autorizzazione è stata concessa o meno.
- **elencoAutorizzazioni:** elenca le autorizzazioni associate a un determinato produttore.

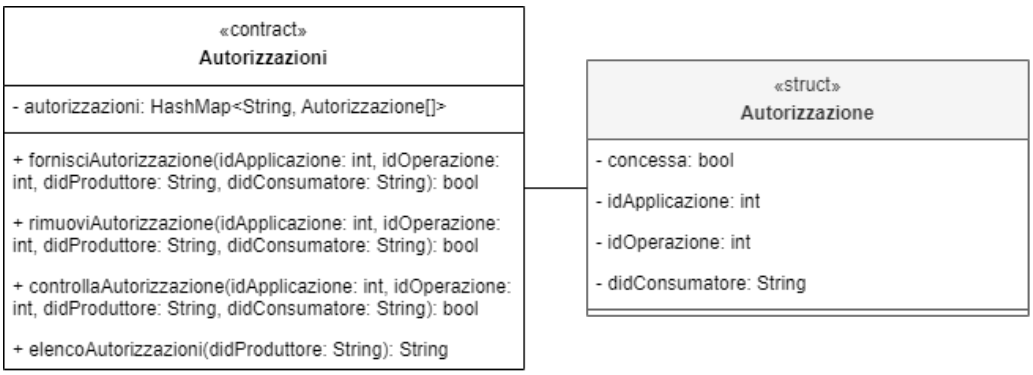

Figura 4.9: Diagramma UML dello smart contract Autorizzazioni

Lo smart contract Autorizzazioni lavora a stretto contatto con quello GestioneApplicazioni per far sì che quest'ultimo fornisca le informazioni sui canali dati e log solo a chi ha le giuste autorizzazioni.

4.3.1.3 Gestore applicazioni

Lo smart contract GestioneApplicazioni è abbastanza complesso perché è pensato per gestire la pubblicazione e l'esplorazione delle applicazioni, la definizione delle operazioni sui dati, la memorizzazione e l'accesso ai riferimenti utili per scrivere e leggere sui canali dati e log.
Per realizzare le funzionalità appena citate, sono necessarie varie strutture:

- **Applicazione:** struttura per le applicazioni decentralizzate. Contiene al suo interno i dati inerenti all'applicazione pubblicata come nome, descrizione, url per download e DID del suo sviluppatore.
- **Operazione:** dedicata alle operazioni di raccolta ed elaborazione dei dati. Contiene l'ID dell'applicazione, il titolo, la descrizione, il codice software e la tipologia (privata, ristretta e aggregata).
- **Canali:** usata per memorizzare i riferimenti associati ai canali dati e log usati da un certo produttore per una specifica operazione sui dati. Al suo interno, vengono memorizzati gli oggetti Author serializzati, i link di annuncio dei canali e gli indirizzi degli ultimi messaggi pubblicati.
- **Iscritto:** contiene gli oggetti Subscriber usati dai consumatori per accedere ai canali dati e log.

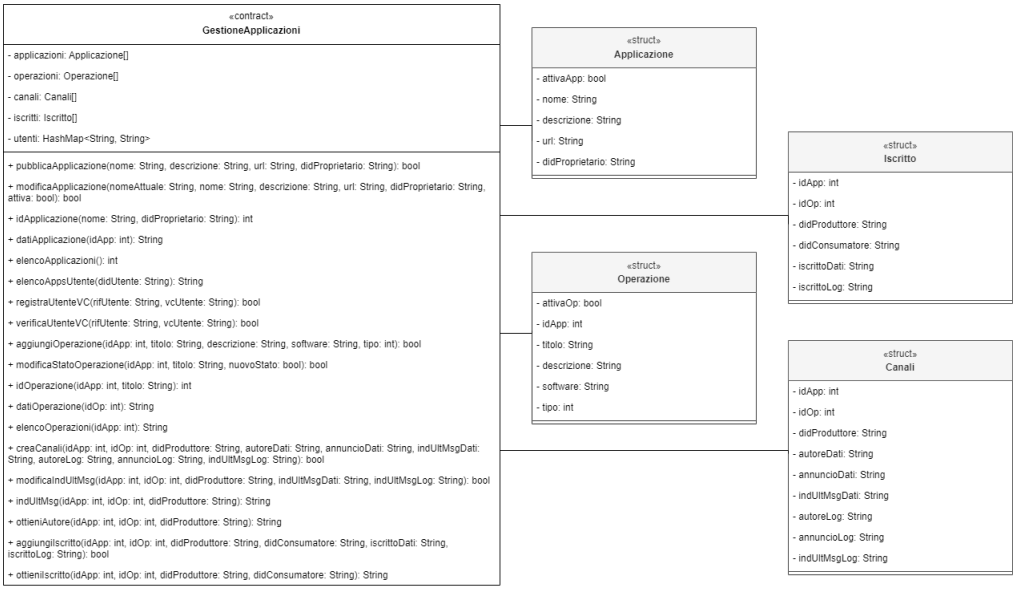

Figura 4.10: Diagramma UML dello smart contract GestioneApplicazioni

Le strutture appena discusse vengono adoperate per realizzare i vari attributi. Tra questi è presente anche l'attributo utenti, ovvero una hash map

che memorizza le credenziali verificabili che gli utenti produttori ricevono quando si registrano in un'applicazione decentralizzata.
I metodi implementati sono i seguenti:

- **pubblicaApplicazione:** pubblica l'applicazione con i relativi dati nello store.
- **modificaApplicazione:** modifica i dati di un'applicazione pubblicata.
- **idApplicazione:** restituisce l'ID univoco associato a un'applicazione.
- **datiApplicazione:** restituisce i dati di un'applicazione pubblicata.
- **elencoApplicazioni:** restituisce il numero totale delle applicazioni pubblicate.
- **elencoAppsUtente:** restituisce le applicazioni pubblicate da uno specifico utente.
- **registraUtenteVC:** registra le credenziali verificabili di un utente.
- **verificaUtenteVC:** verifica le credenziali verificabili di un utente.
- **aggiungiOperazione:** aggiunge un'operazione sui dati per un'app.
- **modificaStatoOperazione:** attiva o disattiva un'operazione sui dati.
- **idOperazione:** restituisce l'ID univoco associato a un'operazione.
- **datiOperazione:** restituisce i dati di un'operazione.
- **elencoOperazioni:** elenca tutte le operazioni di una determinata app.
- **creaCanali:** crea i canali dati e log quando un utente fornisce l'autorizzazione a un'operazione.
- **indUltMsg:** restituisce gli indirizzi degli ultimi messaggi pubblicati sui canali dati e log associati a un'autorizzazione concessa.

- **modificaIndUltMsg:** aggiorna gli indirizzi degli ultimi messaggi pubblicati sui canali dati e log a seguito di un'operazione di scrittura sui canali stessi.
- **ottieniAutore:** restituisce gli oggetti Author usati da un certo produttore per accedere ai canali dati e log associati a un'autorizzazione concessa.
- **aggiungiIscritto:** aggiunge gli oggetti Subscriber usati da un certo consumatore per accedere ai canali dati e log associati a un'autorizzazione ottenuta.
- **ottieniIscritto:** restituisce gli oggetti Subscriber usati da un certo consumatore per accedere ai canali dati e log associati a un'autorizzazione ottenuta.

Grazie a questo smart contract, diventa possibile eseguire la maggior parte delle funzionalità offerte dall'ecosistema in maniera decentralizzata.

4.4 Applicazioni utente

Le applicazioni utente comprendono sia la dApp di sistema che le applicazioni decentralizzate implementate dai consumatori per operare sui dati dei produttori. Queste due tipologie di applicazioni sono realizzate in maniera tale da comunicare con gli ISC e condividono l'utilizzo dei framework IOTA Streams e Identity

4.4.1 dApp di sistema

La dApp di sistema nasce per offrire agli utenti un'applicazione per interagire facilmente con l'ecosistema. Essa interagisce con gli smart contract e con i dati nei canali, offrendo le seguenti categorie di funzionalità:

- **Utente:** offerte ai generici utenti dell'ecosistema. Comprendono l'autenticazione e l'esplorazione delle applicazioni pubblicate con le relative operazioni sui dati. Nel primo caso, avviene un'interazione con lo ISC Autenticazione e viene sfruttato anche il framework IOTA Identity. Nel secondo caso, viene adoperato lo ISC GestioneApplicazioni per leggere le informazioni delle app pubblicate e delle operazioni annesse.

- **Produttore:** dedicate agli utenti che ricoprono il ruolo di produttore. Quest'ultimo deve poter gestire le autorizzazioni che ha concesso, accedere ai dati prodotti e controllare i log. Per far ciò, è necessario interagire con gli smart contract Autorizzazioni e GestoreApplicazioni sia per recuperare tutte le autorizzazioni concesse sia per ottenere i riferimenti ai canali dati e log usati. Tramite questi ultimi, sfruttando il framework IOTA Streams, diventa possibile accedere ai canali per leggere i dati prodotti e i log.

- **Consumatore:** specifiche per gli utenti che svolgono il ruolo di consumatore. Nell'ecosistema trattato, il consumatore è un semplice utente che possiede l'autorizzazione per accedere ai dati di un produttore. Più nello specifico, un consumatore spesso può essere anche lo sviluppatore di un'applicazione decentralizzata usata proprio dai produttori. Di conseguenza, tra le funzionalità offerte al consumatore deve esserci la possibilità di pubblicare nuove applicazioni, gestire quelle già pubblicate, aggiungere o rimuovere operazioni sui dati e accedere ai dati dei produttori, sempre se l'autorizzazione ricevuta lo contempla. Anche in questo caso, bisogna interagire con lo smart contract GestoreApplicazioni e con il canale dati.

Come è possibile evincere dalla Figura 4.11, un ruolo centrale è svolto dal modulo Core che si interfaccia con l'utente, tramite l'apposita GUI, e coordina tutti gli altri componenti per soddisfare le sue richieste. La dApp di sistema comunica con le reti IPFS e IOTA mediante apposite interfacce web che consentono lo scambio di informazioni con i nodi tramite delle richieste

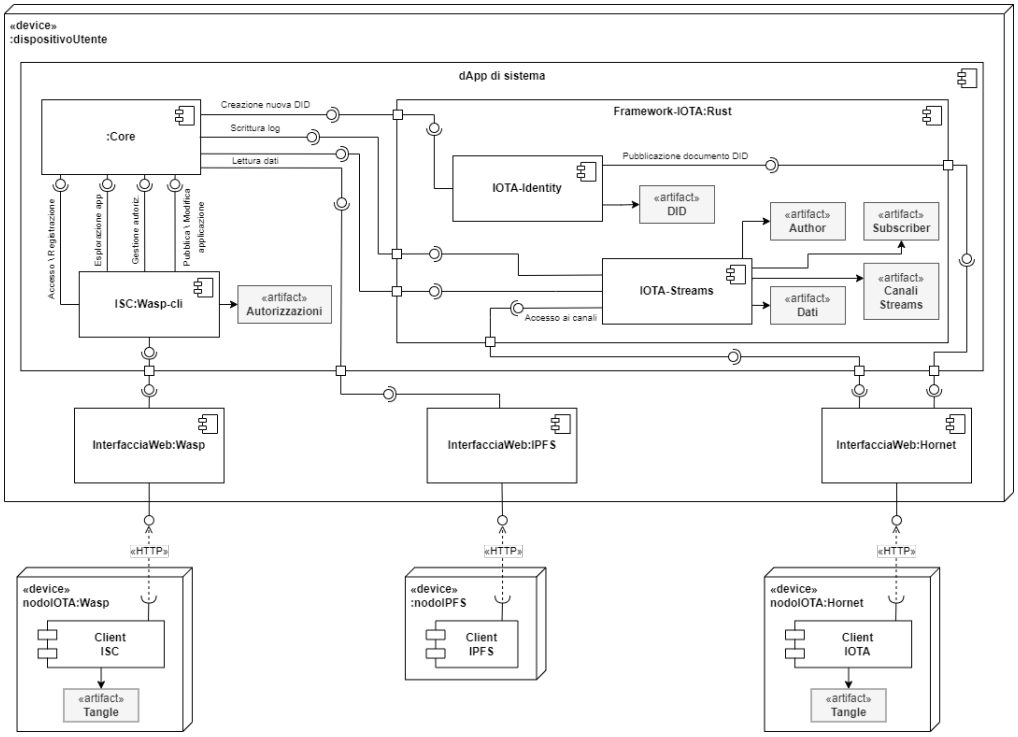

Figura 4.11: Diagramma dei componenti della dApp di sistema

HTTP. Queste interfacce sono sfruttate soprattutto dai framework IOTA e dalla Wasp-cli.

Per quanto riguarda quest'ultima, è un'interfaccia a riga di comando che consente l'interazione con gli smart contract. Infatti, la Wasp-cli è in grado di inviare richieste on-ledger e off-ledger per chiamare le funzioni Func e View di uno specifico ISC. Il nodo Wasp riceve le richieste HTTP sulla sua API web che, purtroppo, non è ancora pubblica. Per questo motivo non è possibile inviare le richieste direttamente all'API web e bisogna sfruttare i comandi della Wasp-cli. Tramite l'interazione con gli ISC è possibile ottenere le autorizzazioni concesse dall'utente produttore e i riferimenti necessari per interagire con i vari canali.

Un altro modulo fondamentale è quello dei framework. Identity viene usato per creare la DID quando l'utente deve registrarsi nell'ecosistema. Naturalmente, in questo caso il framework si occupa persino di pubblicare il

relativo documento DID sul Tangle. Per quanto riguarda Streams, questo gestisce le letture e le scritture sui canali dati e log. Per eseguire queste operazioni, sono necessari i riferimenti dei canali e gli oggetti Author o Subscriber, rispettivamente per produttori o consumatori. Queste informazioni vengono ottenute a seguito dell'interazione con gli smart contract. Le letture possono essere eseguite sul canale dati e log oppure nel file system IPFS. La scrittura avviene solo sul canale log per registrare le operazioni di accesso ai dati.

Siccome IOTA Streams e Identity sono codificati in Rust, per utilizzarli bisogna realizzare un'applicazione in Rust oppure cercare di integrarli tramite dei binding con altri linguaggi di programmazione. Purtroppo, quest'ultima operazione risulta essere difficile a causa della complessità dei framework stessi. Infatti, non esistono dei binding ufficiali realizzati dalla community di sviluppatori di IOTA. Questo sicuramente rappresenta un punto a sfavore perché complica la loro integrazione in applicazioni sviluppate con un linguaggio di programmazione differente da Rust.

4.4.2 Applicazione decentralizzata

Le applicazioni decentralizzate sono le applicazioni sviluppate dagli utenti consumatori dell'ecosistema per raccogliere ed elaborare i dati degli utenti produttori. Queste applicazioni possono essere sviluppate per diversi dispositivi e possono offrire svariate funzionalità. Nonostante ciò, la complessità e la varietà delle applicazioni decentralizzate possono essere riassunte nei moduli evidenziati nella Figura 4.12.

Anche in questo caso, il modulo Core si interfaccia con l'utente e coordina le operazioni che gli altri componenti devono eseguire. Molti componenti sono gli stessi già analizzati nella dApp di sistema: la Wasp-cli, le interfacce per interagire con le reti IPFS e IOTA, i framework Streams e Identity.

In questa tipologia di applicazione, Identity deve essere integrato per gestire le credenziali verificabili. Queste vengono create e associate alla DID di un utente quando si registra nell'applicazione. Le credenziali verificabili

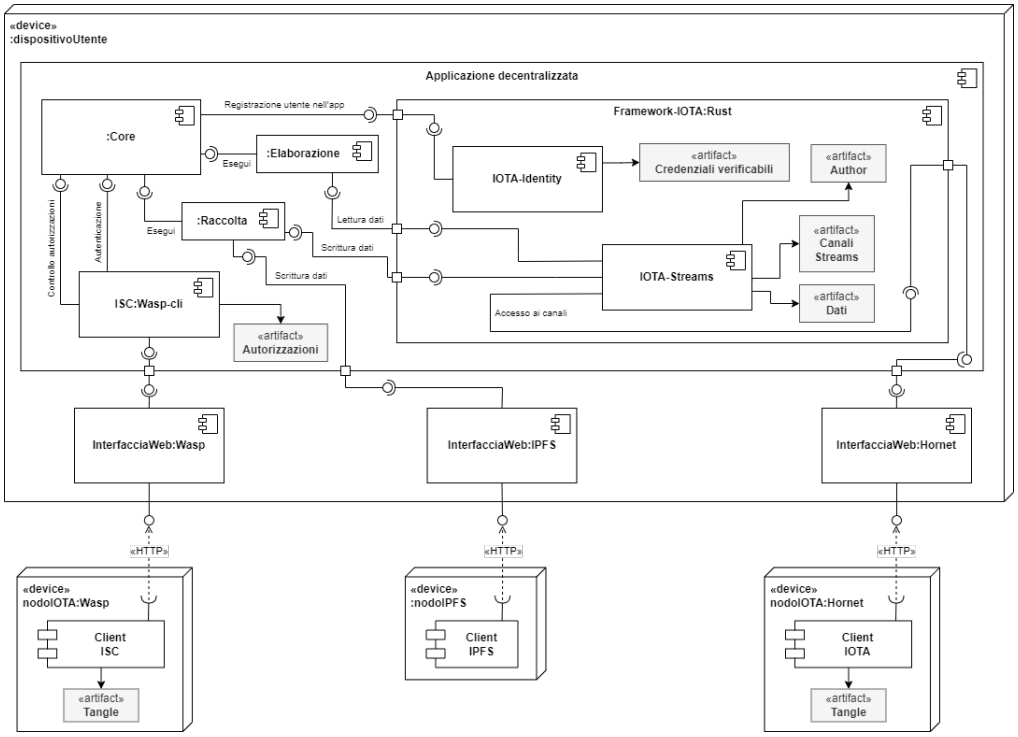

Figura 4.12: Diagramma dei componenti dell'applicazione decentralizzata

sono fondamentali per far sì che l'applicazione possa raccogliere e usufruire dei dati personali dell'utente, associandoli direttamente alla sua DID. Una volta create, le credenziali dovranno essere salvate localmente e memorizzate sullo smart contract per poter essere usate quando l'utente eseguirà il login. Per quanto riguarda il framework Streams, questo viene usato per scrivere i dati prodotti e leggerli quando bisogna elaborarli. Infatti, il modulo Streams viene gestito direttamente da quelli di Raccolta ed Elaborazione, non dal Core. È importante sottolineare che nella fase di raccolta ed elaborazione, il pieno controllo resta sempre nelle mani del produttore. Per questa ragione, l'accesso ai canali viene fatto solo con l'oggetto Author.

Nell'applicazione decentralizzata sono presenti due nuovi moduli: Raccolta ed Elaborazione. Come suggeriscono i nomi, il primo è impiegato nelle operazioni di raccolta dati; mentre il secondo è dedicato alla loro elaborazione. Siccome le applicazioni decentralizzate possono essere di varie tipologie,

le modalità di raccolta ed elaborazione possibili diventano molteplici ed eterogenee. Pertanto, questi moduli non sono stati approfonditi nel dettaglio a livello di componenti interni. Esistono però varie operazioni che sono comuni alle varie implementazioni dei moduli. Innanzitutto, prima di eseguire un'operazione sui dati, bisogna controllare che l'utente produttore loggato abbia effettivamente concesso l'autorizzazione. Questo controllo viene fatto interrogando lo ISC Autorizzazioni tramite la Wasm-cli. In caso di esito positivo, bisognerà recuperare i riferimenti dei canali e l'oggetto Author, però dallo smart contract GestioneApplicazioni. Solo allora sarà possibile eseguire l'operazione di raccolta ed elaborazione. Per la raccolta, i dati saranno scritti sul canale dati o nel file system IPFS. Al contrario, l'elaborazione comporterà la lettura da queste due reti. Ogni operazione eseguita con successo comporterà anche la sua registrazione sul canale log.

Come già sottolineato per la dApp di sistema, l'utilizzo di framework scritti in Rust potrebbe rappresentare un problema in fase di implementazione. Visto che le applicazioni decentralizzate sono pensate per diversi utilizzi e dispositivi, possono essere sviluppate usando svariati linguaggi di programmazione. Purtroppo, non è detto che tutti i linguaggi di programmazione riescano a integrare con successo i moduli in Rust. Quindi, è importante che gli sviluppatori delle applicazioni decentralizzate si informino a riguardo e che la community di IOTA si appresti a rilasciare dei binding per i suoi framework.

Capitolo 5

Valutazione sperimentale

Nei capitoli precedenti l'ecosistema è stato interamente analizzato, partendo dalla sua progettazione fino ad arrivare alla sua implementazione. Questo capitolo è dedicato alla sua valutazione sperimentale, cercando di simulare degli scenari applicativi quanto più possibile realistici.

5.1 Reti IOTA

La valutazione sperimentale dell'ecosistema è influenzata dalla tipologia di rete IOTA che usiamo. I framework Streams e Identity sfruttano le reti IOTA basate sul protocollo Chrysalis: Mainnet e Devnet. Mentre, gli IOTA Smart Contracts sono distribuiti sulla rete IOTA Shimmer, che usa il protocollo Stardust.
Ogni rete espone degli endpoint con particolari caratteristiche:

- **Devnet**
  - **API HTTP REST:** https://api.lb-0.h.chrysalis-devnet.iota.cafe e https://api.lb-1.h.chrysalis-devnet.iota.cafe
  - **PoW minimo richiesto:** 2000
  - **Messaggi ricevuti al secondo:** 5,2
  - **Messaggi referenziati al secondo:** 5,5

- **Mainnet**
  - **API HTTP REST:** https://chrysalis-nodes.iota.org/ e https://chrysalis-nodes.iota.cafe/
  - **PoW minimo richiesto:** 4000
  - **Messaggi ricevuti al secondo:** 5,7
  - **Messaggi referenziati al secondo:** 6,9
- **Shimmer**
  - **API HTTP REST:** https://api.shimmer.network/
  - **PoW minimo richiesto:** 1500
  - **Messaggi ricevuti al secondo:** 7,3
  - **Messaggi referenziati al secondo:** 12,2

Una delle differenze più importanti è sicuramente il PoW minimo richiesto (approfondito nel paragrafo 1.2.7.5). Infatti, quest'ultimo incide sulle prestazioni perché influenza lo sforzo computazionale richiesto al dispositivo dell'utente affinché le sue transazioni vengano accettate e pubblicate.

5.2 Valutazione delle performance

La condivisione dei dati rappresenta un aspetto cruciale per l'ecosistema. Di conseguenza, bisogna testare i meccanismi di condivisione per valutare le loro prestazioni e rilevare la presenza o meno di colli di bottiglia. Tra le operazioni eseguite sui dati, le più importanti sono la scrittura e la lettura sui canali Streams. Queste due sono composte da più sotto operazioni, di cui le più onerose sono sicuramente la cifratura e la firma digitale dei messaggi. A seguire, per ordine di rilevanza, ci sono tutte le operazioni che scambiano informazioni con gli ISC. Queste ultime sono più efficienti perché sono basate principalmente su una serie di richieste HTTP. La parte di elaborazione è svolta soprattutto sui nodi Wasp che si occupano di pubblicare le transazioni

e di eseguire le funzioni degli smart contract.
Le prestazioni delle operazioni appena citate saranno misurate su reti diverse e impiegando dispositivi differenti:

- **PC:** CPU Intel Core i7-770HQ (4 core @2,80 GHz), RAM 16 GB (DDR4 2400MHz), GPU Nvidia Geforce GTX 1050 4 GB, Windows 10 64 bit.
- **Raspberry Pi 400:** CPU Broadcom ARM Cortex-A72 (4 core @1,8 GHz), RAM 4 GB (DDR4 3200MHz), GPU Broadcom VideoCore VI 2GB, Raspberry Pi OS 64 bit.

I risultati ottenuti verranno commentati e mostrati tramite grafici appositi.

5.2.1 Scrittura e lettura messaggi sul Tangle e sul canale Streams

Il primo test è pensato per confrontare le prestazioni del sistema quando esegue le operazioni di scrittura e lettura sul Tangle e sui canali Streams. Creare ed elaborare un messaggio Streams è più complesso e meno efficiente rispetto a una transazione semplice: i messaggi sono cifrati e autenticati. Pertanto, è necessario valutare l'impatto delle primitive crittografiche sulle prestazioni.

La scrittura sul Tangle si basa su due fasi: creazione e pubblicazione di una transazione semplice. Nella prima fase, viene creata la transazione contenente certi valori, formattati secondo precise regole. Ogni transazione deve referenziare almeno altre due transazioni già pubblicate sul Tangle (tip), i cui indirizzi vengono ottenuti interrogando un nodo della rete IOTA. Infine, il dispositivo utente deve eseguire il PoW sulla transazione appena creata. Nella fase di pubblicazione, la transazione viene inviata a un nodo Hornet, il quale la controlla e la invia in broadcast agli altri nodi della rete.

La scrittura sul canale Streams segue le stesse procedure del caso precedente, con l'aggiunta delle operazioni di cifratura e di firma digitale durante la fase di creazione. La cifratura è eseguita con una chiave simmetrica condivisa tra gli iscritti al canale; mentre la firma digitale viene apposta usando la chiave privata di chi pubblica. L'esecuzione di queste primitive crittografiche avrà sicuramente un impatto non trascurabile sui tempi necessari per creare la transazione che poi dovrà essere inviata a un nodo Hornet per la pubblicazione.

Per quanto riguarda le operazioni di lettura, valgono le stesse considerazioni fatte per la scrittura. Infatti, la lettura e l'elaborazione dei messaggi Streams prevede le fasi di decifrazione e verifica della firma digitale, oltre a quelle eseguite anche per le transazioni normali. Di conseguenza, le letture sui canali Streams impiegheranno più tempo rispetto a quelle eseguite direttamente sul Tangle.

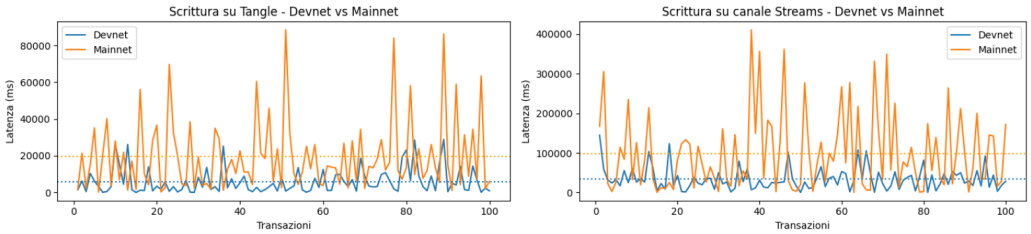

Figura 5.1: Grafici della latenza per le scritture sul Tangle e sui canali Streams

Analizzando i grafici, è possibile notare fin da subito la variabilità della latenza per eseguire sia le operazioni di scrittura che quelle di lettura. Pertanto, saranno considerati i tempi medi per effettuare i confronti tra i vari casi. La scrittura sul Tangle impiega 6 secondi sulla Devnet e 19,52 secondi sulla Mainnet; a differenza della scrittura sul canale Streams che dura 34,42 secondi sulla Devnet e 98,85 secondi sulla Mainnet. Innanzitutto, possiamo evincere che le scritture sul Tangle e sui canali Streams sono rispettivamente il 69,3% e il 65,2% più veloci sulla Devnet rispetto alla Mainnet. Questo sicuramente è dovuto al PoW minimo richiesto: 2000 sulla Devnet contro

4000 sulla Mainnet, il doppio. Inoltre, a parità di rete IOTA, confrontando la scrittura delle transazioni semplici con quella dei messaggi Streams, emerge che la pubblicazione di una transazione semplice è circa l'80% più veloce rispetto a quella di un messaggio Streams.

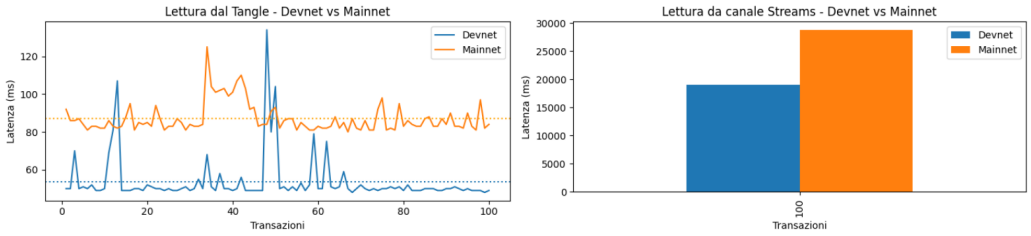

Figura 5.2: Grafici della latenza per le letture dal Tangle e dai canali Streams

Per quanto riguarda le letture, dal Tangle impiega 53,82 ms sulla Devnet e 87,02 ms sulla Mainnet; mentre dal canale Streams impiega 190,66 ms sulla Devnet e 287,96 ms sulla Mainnet. Sicuramente la lettura dai canali è più lenta rispetto a quella dal Tangle, però il framework IOTA Streams offre la possibilità di leggere ed elaborare più messaggi in una singola operazione, ottimizzando complessivamente i tempi di elaborazione. Le letture dal Tangle sono più veloci di quelle sul canale dati del 71,77% sulla Devnet e del 69,78% sulla Mainnet. Le letture sulla Devnet rispetto a quelle sulla Mainnet risultano essere più veloci del 38,15% e del 33,79% rispettivamente per le transazioni semplici e per i messaggi Streams.

5.2.2 Confronto tra PC e Raspberry Pi 400

Il primo test ha messo in risalto la differenza tra le operazioni sui canali Streams e quelle che operano direttamente sul Tangle. È importante considerare che questo confronto è stato eseguito su un PC di fascia media abbastanza prestante, eppure c'è stato un peggioramento non trascurabile dal punto di vista prestazionale.

Questo test è stato pensato per confrontare le prestazioni del PC con quelle del Raspberry Pi 400. La comparazione tra i due dispositivi è essen-

ziale perché le applicazioni decentralizzate possono essere sviluppate anche per i dispositivi IoT, meno potenti sotto tutti i punti di vista rispetto ai moderni PC. Dalle specifiche tecniche iniziali è possibile notare che entrambi i dispositivi sono basati su un'architettura a 64 bit. La differenza è che quella del PC è di tipo CISC (Complex Instruction Set Computing), mentre quella del Raspberry è di tipo RISC (Reduced Instruction Set Computing). Sarà interessante valutare se e quanto questa differenza renderà più o meno efficienti le operazioni eseguite sul Raspberry che, tra l'altro, rappresenta persino il target del progetto IOTA, nato principalmente per rendere la DLT accessibile ai dispositivi IoT.

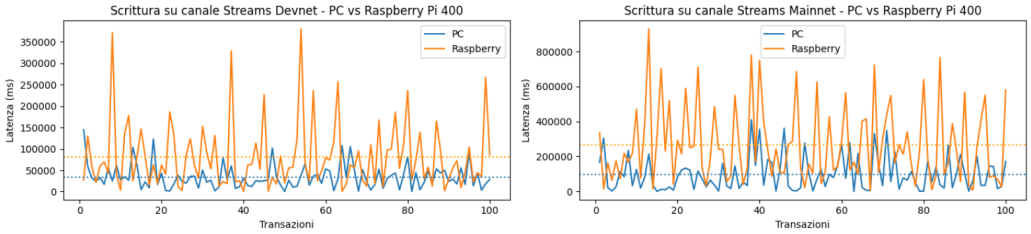

Figura 5.3: Grafici della latenza per le scritture sui canali Streams da PC e Raspberry Pi 400

Anche in questo caso, dai grafici emerge una variabilità abbastanza accentuata per i valori della latenza, soprattutto per il Raspberry. Di conseguenza, saranno presi in considerazione i valori medi: la scrittura da PC impiega 34,42 secondi sulla Devnet e 98,85 secondi sulla Mainnet; diversamente la scrittura da Raspberry impiega 81,30 secondi sulla Devnet e 268,11 secondi sulla Mainnet. I tempi medi confermano la differenza prestazionale. Il PC è più veloce nella scrittura del 57,66% sui canali Streams della Devnet e del 63,13% su quelli della Mainnet. Nello stesso lasso di tempo impiegato dal Raspberry per concludere una scrittura, il PC ne avrebbe già completato almeno due. Il calo delle prestazioni è sicuramente influenzato dalle primitive crittografiche che impiegheranno più tempo per concludere la loro esecuzione a causa dell'hardware meno prestante.

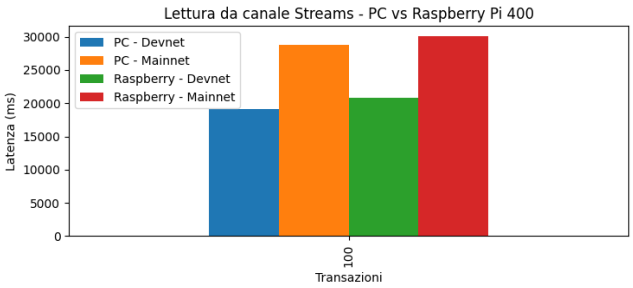

Figura 5.4: Grafici della latenza per le letture sui canali Streams da PC e Raspberry Pi 400

Per le letture consideriamo i tempi impiegati per leggere ed elaborare 100 messaggi in varie situazioni: la lettura da PC avviene in 19,06 secondi sulla Devnet e in 28,80 secondi sulla Mainnet, a differenza del Raspberry che impiega 20,80 secondi sulla Devnet e 30,10 secondi sulla Mainnet. Comparando le misurazioni, emerge che il Raspberry riesce a reggere il confronto con il PC per quanto riguarda le operazioni di lettura. Infatti, il PC risulta essere più veloce solo di circa il 6

5.2.3 Interazione con gli IOTA Smart Contracts

Il test in questione è pensato per valutare le interazioni tra le applicazioni utente e gli IOTA Smart Contracts. Per questo scopo è stato creato uno ISC apposito chiamato Performance. Al suo interno è presente un array di stringhe, una funzione Func per salvare nell'array la stringa ricevuta come parametro e una funzione View per ottenere la stringa associata a un certo indice dell'array. Le funzioni dello ISC Performance sono state pensate per eseguire elaborazioni semplici. Questa scelta è stata fatta proprio perché lo scopo di questo test è verificare la latenza delle comunicazioni tra applicazione utente e smart contract, senza tener conto dei tempi impiegati dallo ISC per eseguire il codice della funzione. La comunicazione tra l'applicazione e gli smart contract viene gestita tramite la wasp-cli. Anche in questo test saranno usati i suoi comandi per chiamare le funzioni Func e View e saranno misurati

i tempi impiegati per eseguire le elaborazioni richieste.

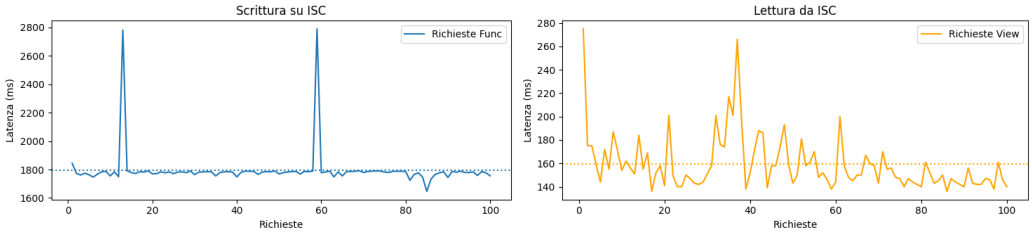

Figura 5.5: Grafici della latenza per le scritture e le letture sugli ISC

Analizzando il grafico delle scritture è possibile notare un'omogeneità nei tempi, che si attestano intorno a una latenza media pari a 1,80 secondi. Invece, nel grafico delle letture è presente una irregolarità maggiore, che si attesta intorno a un valore medio di 0,16 secondi. I tempi medi ottenuti sono ottimi e sono sinonimo di una comunicazione efficiente e veloce.

5.2.4 Confronto con sistema centralizzato simile

L'ultimo test nasce con l'intento di confrontare le prestazioni delle operazioni di scrittura e lettura sui canali Streams e sugli ISC con operazioni simili eseguite su un database. Questa comparazione è utile per comprendere se le prestazioni dell'ecosistema decentralizzato si avvicinano o meno a quelle di un sistema analogo centralizzato. Questo test si compone di due parti. La prima effettua scritture e letture di messaggi semplici, similmente come avviene per le interazioni con gli ISC. La seconda parte esegue scritture e letture di messaggi cifrati e autenticati, simulando un po' ciò che succede sui canali Streams. Per la cifratura simmetrica viene usato l'algoritmo AES-GCM e per la firma digitale il cifrario asimmetrico Ed25519. I programmi che eseguono i test sono codificati in Rust in maniera tale da mantenere lo stesso linguaggio di programmazione utilizzato per i moduli delle applicazioni utente. Per quanto riguarda il database, è stato scelto MongoDB per la sua facilità di utilizzo e di integrazione nei programmi Rust.

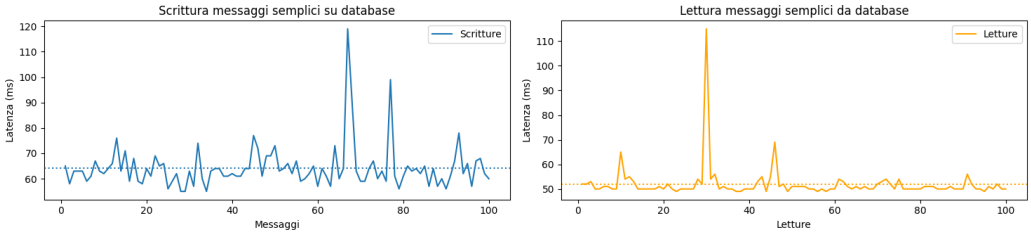

Figura 5.6: Grafici della latenza per le scritture e le letture di messaggi semplici su database

Analizzando i grafici soprastanti, è possibile notare delle latenze abbastanza basse: un valore medio di 64,21 millisecondi per le scritture e di 51,92 millisecondi per le letture. Questi tempi andrebbero confrontati con quelli raccolti per lo stesso tipo di operazioni eseguite sugli ISC, dove avveniva la scrittura e la lettura di messaggi semplici sullo stato interno degli smart contract stessi. Da questo confronto, emerge che le scritture sugli ISC sono il 96% più lente rispetto a quelle sul database. Inoltre, le letture dal database risultano il 66% più veloci rispetto a quelle dagli smart contract.

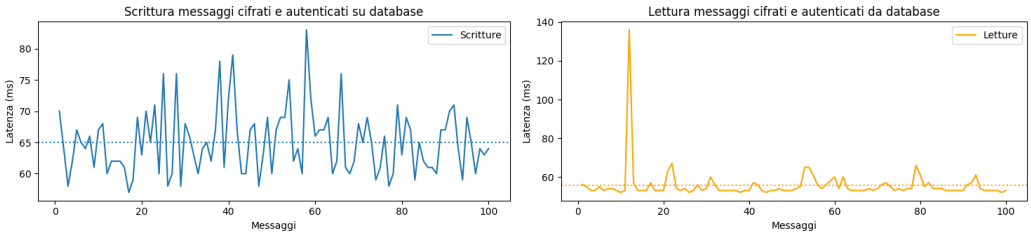

Figura 5.7: Grafici della latenza per le scritture e le letture di messaggi cifrati e autenticati su database

Per quanto riguarda la seconda fase del test, dai grafici è possibile osservare un aumento della latenza a causa dell'esecuzione delle primitive crittografiche. Prendendo come riferimento i tempi medi, la preparazione del messaggio e la sua scrittura impiegano 65,01 millisecondi; a differenza della lettura e dell'elaborazione del messaggio che impiegano 55,66 millisecondi. Naturalmente, anche in questo caso il database si conferma più veloce dei

canali Streams.

Questo test mette in risalto la superiorità prestazionale delle scritture e delle letture su un database centrale a discapito delle stesse operazioni eseguire sulla DLT di IOTA. La discrepanza emersa è causata principalmente dai meccanismi che regolano la DLT, ovvero dall'esecuzione del PoW, dalla pubblicazione delle transazioni, dalla loro conferma e dal raggiungimento del consenso distribuito. Queste ultime operazioni non sono presenti nella controparte centralizzata: i dati sono scritti e acceduti direttamente. Di conseguenza, la latenza totale diminuisce ma, allo stesso tempo, vengono meno tutte le caratteristiche peculiari descritte nei prossimi paragrafi.

5.3 Scalabilità

L'ecosistema è stato implementato usando tecnologie scalabili, in grado di supportare dinamicamente il carico utente. Le operazioni più eseguite sono sicuramente la pubblicazione dei dati e la comunicazione con gli smart contract.

I dati prodotti vengono pubblicati sul Tangle, ovvero su una struttura a grafo aciclico. Grazie a quest'ultima si superano i limiti della blockchain e diventa possibile validare parallelamente più transazioni tramite un consenso probabilistico. Il Tangle è la struttura ideale per l'ecosistema proposto visto che riesce ad archiviare, più velocemente rispetto alle blockchain, gli innumerevoli dati prodotti da molti utenti che usano le varie applicazioni decentralizzate.

La comunicazione con gli smart contract è altrettanto importante dato che rappresentano la base per gestire gli utenti, le applicazioni decentralizzate e le relative operazioni sui dati. Pertanto, sono stati scelti gli IOTA Smart Contracts perché hanno delle bassissime fee e possono essere eseguiti su più chain. Quest'ultima feature conferisce scalabilità orizzontale agli smart contract, caratteristica necessaria per supportare un carico utente dinamico e crescente.

5.4 Sicurezza delle informazioni

La sicurezza delle informazioni è un altro aspetto particolarmente curato nell'ecosistema. Tra le informazioni da proteggere rientrano i dati personali degli utenti, i dati prodotti tramite le applicazioni decentralizzate e gli oggetti Author e Subscriber usati per accedere ai canali Streams.

Per proteggere le informazioni personali degli utenti, vengono adoperate le identità decentralizzate e le credenziali verificabili. La DID garantisce la privacy degli utenti e consente l'identificazione di persone, organizzazioni e oggetti, senza la necessità di usare dati personali o i classici username e password. Inoltre, la DID può essere impiegata in concomitanza con le credenziali verificabili. Queste ultime vengono rilasciate agli utenti da un Issuer integrato all'interno dell'applicazione decentralizzata. Le credenziali diventano accessibili solo all'utente, il quale le potrà sfruttare per eseguire il login su una specifica applicazione. Tutte le informazioni personali richieste dalle applicazioni potranno essere memorizzate nelle credenziali verificabili. Grazie a ciò, i dati degli utenti non dovranno più essere archiviati all'interno dei database, sui quali potrebbero verificarsi accessi indesiderati e furti.

I dati prodotti dagli utenti sono altrettanto sensibili. Pertanto, devono essere protetti sfruttando la massima sicurezza offerta dalle tecnologie adoperate. Innanzitutto, l'ecosistema sfrutta l'immutabilità del Tangle per assicurarsi che nessuno possa eliminare o censurare i dati pubblicati. Fino a quando ci sarà almeno un nodo della rete IOTA online, i dati sul Tangle saranno pubblicamente accessibili. Proprio perché l'accesso è pubblico, i dati devono essere cifrati e autenticati. Questa necessità viene soddisfatta dal framework IOTA Streams e dai suoi canali, sui quali è possibile archiviare i dati proprio in maniera cifrata e autenticata. Inoltre, il framework implementa le migliori best practice in campo di sicurezza informatica. Ad esempio, non rende accessibile agli utenti la chiave simmetrica usata da autori e iscritti per cifrare e decifrare i dati pubblicati sul canale. Inoltre, consente l'esportazione protetta da password degli oggetti Author e Subscriber. Quest'ultima operazione è fondamentale per l'ecosistema perché consente l'archiviazione degli

Author e dei Subscriber sugli smart contract, rendendoli accessibili solo dopo i dovuti controlli delle autorizzazioni richieste. Infine, anche nel caso in cui un soggetto non autorizzato riuscisse ad accedere a un oggetto Author o Subscriber serializzato, non riuscirebbe a utilizzarlo per accedere al canale perché gli mancherebbe la password per deserializzarlo. In conclusione, sia i punti di accesso ai canali Streams che i dati pubblicati sono protetti per evitare alterazioni e accessi indesiderati.

5.5 Analisi dei costi

Tra le varie analisi effettuate sul sistema, è importante valutare anche l'aspetto economico. Se l'ecosistema è troppo costoso da realizzare e manutenere o se i costi a carico dei produttori e dei consumatori sono troppo elevati, questo progetto diventa irrealizzabile.

Fortunatamente, lo scheletro dell'ecosistema è basato sul progetto IOTA che nasce con l'idea di rendere la DLT accessibile a tutti gratuitamente. Di conseguenza, le spese inerenti all'archiviazione dei dati sono nulle, proprio perché l'uso del Tangle è a costo zero. Persino i servizi dedicati agli sviluppatori sono gratuiti, a differenza degli store di applicazioni dei competitor (Google Play, Microsoft Store e App Store) che richiedono una tassa di iscrizione o percepiscono una percentuale sui guadagni. Inoltre, se dovessimo implementare un ecosistema simile ma centralizzato, sicuramente i costi di realizzazione sarebbero più elevati. In questo caso, ci sarebbero da considerare i seguenti costi: database, server scalabili orizzontalmente, protezione dei dati, sviluppo con framework proprietari e tanto altro. In definitiva, tutte spese che sono evitabili usando IOTA.

Purtroppo, l'ecosistema non riesce a essere completamente gratuito a causa degli smart contract. Questi ultimi sono composti da varie operazioni (calcoli aritmetici, scrittura e lettura dello stato interno, generazione di eventi, ecc.), ognuna delle quali ha un costo di esecuzione chiamato gas. Quando un utente invia una richiesta a uno smart contract, deve specificare il gas budget,

ovvero quanto è disposto a pagare per eseguire quella richiesta. Questo meccanismo è necessario per evitare l'elaborazione senza fine, magari a causa di un loop infinito. Gli IOTA Smart Contracts non sono da meno. Per fortuna, i loro gas sono sensibilmente più bassi di quelli degli smart contract di Ethereum. In conclusione, l'ecosistema risulta essere competitivo persino sotto l'aspetto economico. In particolar modo, i servizi offerti gratuitamente agli sviluppatori potrebbero favorire l'adozione di questo ecosistema a discapito di altre soluzioni centralizzate.

5.6 Sviluppi futuri

La valutazione sperimentale ha messo in risalto pregi e difetti dell'ecosistema. IOTA è un progetto in continua evoluzione; pertanto, anche l'ecosistema lo è. Nei prossimi mesi verrà rilasciato sulla Mainnet il protocollo Stardust, che sostituirà l'attuale protocollo Chrysalis. Questo cambiamento consentirà l'utilizzo degli IOTA Smart Contracts sulla rete principale e non più su quella di test (Shimmer). Inoltre, diventerà possibile creare e distribuire NFT (Non-Fungible Token), che potranno contenere particolari dati dei produttori ed essere usati come merce di scambio con i consumatori a seguito del pagamento di un compenso in token. I framework Identity e Streams verranno ottimizzati o sostituiti con nuovi framework, che renderanno l'ecosistema maggiormente efficiente e veloce. Nei prossimi anni, verrà rilasciato IOTA 2.0 che renderà l'intera rete completamente decentralizzata, eliminando gli ultimi elementi centralizzati ancora presenti come il Coordinatore.

L'ecosistema potrebbe espandersi per gestire non solo applicazioni decentralizzate ma anche altri servizi che prevedono uno scambio consenziente di dati tra due soggetti. Per perseguire questa evoluzione basterebbe definire altri smart contract in grado di integrare i nuovi servizi, lasciando invariati quelli esistenti che gestiscono le identità decentralizzate e le autorizzazioni per accedere ai dati. Infine, il futuro ecosistema prevederà i micropagamenti basati sul token IOTA. Questi ultimi potrebbero tornare utili per tutte quelle

applicazioni decentralizzate che vendono servizi e prodotti ai propri utenti, come le applicazioni di e-commerce e simili.

Conclusione

In questa tesi è stata presentata un'infrastruttura basata sulla Distributed Ledger Technology per la gestione dei dati prodotti dagli utenti di applicazioni decentralizzate. Negli ultimi anni, la raccolta e l'elaborazione dei dati sono diventate operazioni essenziali per diverse realtà e, contemporaneamente, è cresciuta l'attenzione degli utenti e delle istituzioni pubbliche verso i temi di privacy e protezione delle informazioni personali. La DLT è stata impiegata per assicurare agli utenti il pieno controllo sui propri dati, creando un rapporto diretto tra produttore e consumatore che garantisce maggiore trasparenza e consapevolezza. L'infrastruttura decentralizzata è stata presentata sotto forma di ecosistema, dove vari ambienti eterogenei interagiscono fra di loro per scambiarsi informazioni e offrire agli utenti molteplici funzionalità.

Durante la fase di progettazione sono stati presentati gli attori e i meccanismi coinvolti nel ciclo di vita dei dati. Le parti più importanti sono indubbiamente lo store di applicazioni e il gestore delle autorizzazioni. Lo store serve per supportare gli sviluppatori e la migrazione delle loro applicazioni da sistemi centralizzati a quelli decentralizzati. Il gestore delle autorizzazioni rappresenta il vero valore aggiunto dell'ecosistema in questione perché agevola gli utenti nel controllo dei permessi e nel monitoraggio degli accessi ai propri dati personali. Per supportare il concept del progetto, sono stati definiti i requisiti funzionali, i requisiti non funzionali e i casi d'uso che descrivono le funzionalità più rilevanti.

Tutto ciò che l'ecosistema ha da offrire può essere raggruppato in tre ambienti, i quali comprendono diversi componenti riconducibili alla DLT o

appartenenti ad altre architetture. Il primo ambiente è dedicato alla raccolta e all'archiviazione dei dati provenienti dalle applicazioni decentralizzate usate dai produttori. Ogni dato raccolto viene archiviato, in base alla sua dimensione, all'interno dei canali Streams o del file system IPFS. Il secondo ambiente si occupa dell'accesso ai dati sfruttando gli IOTA Smart Contracts per identificare gli utenti, controllare le autorizzazioni necessarie e ottenere i riferimenti utili per accedere ai canali Streams. L'ultimo ambiente offre diversi servizi smart dedicati agli utenti dell'ecosistema.

Dopo aver definito la sua architettura, l'ecosistema è stato implementato. Sono stati sviluppati i moduli principali della dApp di sistema e di un'applicazione decentralizzata. Questi componenti sono in grado di interagire con il Tangle IOTA, più nello specifico con i canali Streams tramite apposite interfacce che variano in base al ruolo (autore o iscritto) ricoperto dall'utente che usa l'applicazione. Ogni dato viene cifrato e firmato digitalmente per garantire rispettivamente confidenzialità e autenticità. Se il dato è troppo grande, viene archiviato sul file system IPFS e viene referenziato tramite il suo CID contenuto in un messaggio pubblicato sul canale. Qualsiasi operazione sui dati viene registrata su un apposito canale dedicato ai log. Gli IOTA Smart Contracts svolgono un ruolo altrettanto centrale nell'ecosistema. Infatti, vengono impiegati tre smart contract rispettivamente per identificare gli utenti tramite le identità decentralizzate, gestire le applicazioni e le relative operazioni sui dati, gestire le autorizzazioni che i produttori concedono ai consumatori. Grazie a tutti questi componenti, diventa possibile rivoluzionare il ciclo di vita dei dati. Ogni raccolta ed elaborazione viene eseguita solo se il produttore ha fornito le autorizzazioni necessarie. Queste ultime sono gestite in maniera sicura e fidata dagli smart contract. Inoltre, i produttori restano sempre i principali detentori dei propri dati perché controllano il canale Streams su cui sono memorizzati. Pertanto, possono decidere in qualsiasi momento chi e con quali modalità può accedere alle informazioni che hanno prodotto.

Infine, sono stati eseguiti diversi test per valutare sperimentalmente l'eco-

sistema, nello specifico le sue performance. Innanzitutto, sono state comparate le scritture e le letture sul Tangle con quelle sui canali, in maniera tale da valutare l'impatto delle operazioni aggiuntive eseguite dal framework Streams. Successivamente, è stata misurata la latenza delle interazioni con gli ISC. Infine, sono stati eseguiti dei test per confrontare l'ecosistema decentralizzato con uno simile centralizzato che usa un database al posto dei registri distribuiti. Dai test emerge che l'ecosistema raggiunge buone prestazioni ed è scalabile, economico e sicuro. Pertanto, rappresenta una valida alternativa alle controparti centralizzate. Naturalmente, l'ecosistema è destinato a evolversi e a migliorare di pari passo con il progetto IOTA e i suoi futuri aggiornamenti.

Bibliografia

[1] S. Behera, "Distributed Ledger Technology (DLT): advantages, disadvantages and impacted sectors", 2023
https://www.worldexcellence.com/distributed-ledger-technology-impact-on-finance/

[2] S. Troy, M. K. Pratt, "Distributed Ledger Technology", 2021
https://www.techtarget.com/searchcio/definition/distributed-ledger

[3] Fondazione IOTA, "What is IOTA?"
https://www.iota.org/get-started/what-is-iota

[4] Fondazione IOTA, "Logo IOTA"
https://www.iota.org/connect/brand

[5] Fondazione IOTA, "The Tangle", 2023
https://wiki.iota.org/learn/about-iota/tangle/

[6] Fondazione IOTA, "Blockchain bottleneck"
https://wiki.iota.org/assets/images/blockchain-bottleneck-c807d504ff52169427d3ca4cf0179a8c.gif

[7] Fondazione IOTA, "Tangle bottleneck"
https://wiki.iota.org/assets/images/tangle-bottleneck-ad9ab01d5399a5cefbac1cbd1be3900a.gif

[8] Fondazione IOTA, "An introduction to IOTA", 2022
https://wiki.iota.org/learn/about-iota/an-introduction-to-iota/

[9] Fondazione IOTA, "Introduction to protocols", 2023
https://wiki.iota.org/learn/protocols/introduction/

[10] Fondazione IOTA, "Introduction to Chrysalis", 2023
https://wiki.iota.org/learn/protocols/chrysalis/introduction/

[11] Fondazione IOTA, "White Flag Ordering"
https://wiki.iota.org/tips/tips/TIP-0002/

[12] Fondazione IOTA, "Milestone Merkle Validation"
https://wiki.iota.org/tips/tips/TIP-0004/

[13] Fondazione IOTA, "Uniform Random Tip Selection"
https://wiki.iota.org/tips/tips/TIP-0003/

[14] Fondazione IOTA, "Ed25519 Validation"
https://wiki.iota.org/tips/tips/TIP-0014/

[15] Fondazione IOTA, "Tangle Message"
https://wiki.iota.org/tips/tips/TIP-0006/

[16] Fondazione IOTA, "Transaction Payload"
https://wiki.iota.org/tips/tips/TIP-0007/

[17] Fondazione IOTA, "Binary To Ternary Encoding"
https://wiki.iota.org/tips/tips/TIP-0005/

[18] Fondazione IOTA, "Introduction to Stardust", 2023
https://wiki.iota.org/learn/protocols/stardust/introduction/

[19] Fondazione IOTA, "About Nodes", 2023
https://wiki.iota.org/maintain/welcome/

[20] Fondazione IOTA, "Welcome to Hornet", 2023
https://wiki.iota.org/hornet/welcome/

[21] Fondazione IOTA, "Welcome to Chronicle", 2023
https://wiki.iota.org/chronicle/welcome/

[22] Fondazione IOTA, "Output Features", 2022
https://wiki.iota.org/learn/protocols/stardust/core-concepts/output-features/

[23] Fondazione IOTA, "IOTA Client Libraries", 2023
https://wiki.iota.org/build/welcome/

[24] Fondazione IOTA, "Layered overview", 2023
https://wiki.iota.org/assets/images/layered_overview-301744b3b835f4f09ae40dfa09ac564f.svg

[25] Fondazione IOTA, "The Coordinator", 2022
https://wiki.iota.org/learn/protocols/coordinator/

[26] Fondazione IOTA, "The Coordinator", 2022
https://wiki.iota.org/assets/images/milestones-ca3479a44644b9ab2414b07e1e62d6ac.gif

[27] Fondazione IOTA, "Data Transfer", 2022
https://wiki.iota.org/learn/about-iota/data-transfer/

[28] Fondazione IOTA, "Message PoW"
https://wiki.iota.org/tips/tips/TIP-0012/

[29] Fondazione IOTA, "IOTA Streams", 2023
https://wiki.iota.org/streams/overview/

[30] Fondazione IOTA, "Creating a Decentralized Identity", 2023
https://wiki.iota.org/identity.rs/concepts/decentralized_identifiers/create/

[31] Fondazione IOTA, "Decentralized Identifiers (DID)", 2023
https://wiki.iota.org/identity.rs/concepts/decentralized_identifiers/overview/

[32] Fondazione IOTA, "Verifiable Credentials Overview", 2023
https://wiki.iota.org/identity.rs/concepts/verifiable_credentials/overview/

[33] Fondazione IOTA, "Introduction to Decentralized Identity", 2023
https://wiki.iota.org/identity.rs/decentralized_identity/

[34] Fondazione IOTA, "Smart Contract Concepts", 2023
https://wiki.iota.org/shimmer/smart-contracts/guide/wasm_vm/
concepts/

[35] Fondazione IOTA, "Introduction to the Wasm VM for ISC", 2023
hhttps://wiki.iota.org/wasp-wasm/introduction/

[36] Fondazione IOTA, "ISC Host", 2023
https://wiki.iota.org/assets/images/IscHost-
07dc8036e6e793eb77e66d011e1c3a33.png

[37] Fondazione IOTA, "Wasm VM", 2023
https://wiki.iota.org/assets/images/WasmVM-
434a91f1aeffc650a975d9a5387c4568.png

[38] Fondazione IOTA, "Smart Contract Schema Tool", 2023
https://wiki.iota.org/wasp-wasm/how-tos/schema-tool/introduction/

[39] Fondazione IOTA, "Call Context", 2023
https://wiki.iota.org/wasp-wasm/explanations/context/

Elenco delle figure