Paolo Speziali

Condividere informazioni in modo sicuro combinando Git e Blockchain

Relatore:
Prof. Luca Grilli

Tesi di laurea in Ingegneria Informatica

Perugia, Anno Accademico 2020/2021

Università degli Studi di Perugia
Corso di laurea triennale in Ingegneria Informatica ed Elettronica
Dipartimento d'Ingegneria

0. Indice

1. Introduzione

2. Concetti Preliminari

3. Il Problema e L'Obiettivo

4. Il Software PineSU

La concretizzazione della soluzione al problema esposto è l'applicativo **PineSU**.
PineSU si presenta come un software leggero scritto in Javascript e che sfrutta il runtime Node.js. Il software va a considerare gli insiemi di file come delle entità chiamate **Storage Unit** (SU) con cui va ad avvolgere logicamente una repository Git.
Queste SU sono le singole unità su cui si andranno ad effettuare le singole operazioni eccetto la registrazione su Blockchain che si svolgerà collettivamente con l'ausilio di accumulatori crittografici.

Vedremo come il ciclo di vita di una SU sia scandito dai **Blockchain Synchronization Point** (BSP), ovvero gli eventi che si generano quando si decide di andare a registrare lo stato e la presenza di una SU su Blockchain inserendola, tramite gruppi di suoi simili chiamati **Storage Group** (SG), nella grande struttura chiamata **Merkle Calendar** (MC). Infine, la root di questo MC sarà salvata nella Blockchain, da qui in poi sarà possibile, in qualsiasi momento, ricostruire un MC in quel preciso istante in cui un nuovo BSP è stato creato e verificare se la sua root è presente.

4.1 Architettura

Il sistema va ad interfacciarsi con il client Git e con l'API web3.js per la comunicazione con la blockchain Ethereum, possiamo descrivere la sua architettura come in Fig. 4.1, dove troviamo i componenti principali:

- *PineSU **CLI** (Command Line Interface)*. L'interazione con PineSU da parte degli utenti avviene attraverso un emulatore di terminale di una macchina avente NodeJS installato[1], analogamente a come avviene con Git, tuttavia con un'interazione più guidata. Oltre al permettere l'uso delle normali funzioni di PineSU, questo modulo permette anche l'inserimento di un qualsiasi comando di Git in modo da rendere l'utilizzo diretto di quest'ultimo non necessario durante una tipica sessione di lavoro.

[1] L'interfaccia si basa sul modulo npm Inquirer.js `https://github.com/SBoudrias/Inquirer.js`.

- *PineSU **BEL** (Back End Logic).* Questo componente è il nucleo di PineSU. Gestisce tutte le SU e controlla la comunicazione con la blockchain e il client Git locale. Il client Git viene utilizzato inoltre per interagire indirettamente con i server Git remoti. Il modulo BEL, inoltre, si occupa dei due Storge Group, Open (OSG) e Closed (CSG), e mantiene il Merkle Tree dinamico chiamato Merkle Calendar che permette di recuperare efficientemente l'hash registrato in Blockchain per un qualsiasi BSP. La gestione del salvataggio remoto del Merkle Calendar avviene tramite una repository Git scelta dall'utente.

- *PineSU **EC** (Ethereum Connector)* Si interfaccia con il modulo "web3.js"[2].

- *PineSU **GC** (Git Connector)* Si interfaccia con il modulo "simple-git"[3].

- *PineSU **SM** (Smart Contract)* Questo modulo entra in gioco solamente nel caso di una registrazione "forte" di una Storage Unit nella Blockchain, in una fruizione standard dell'applicativo non entrerà probabilmente in azione.

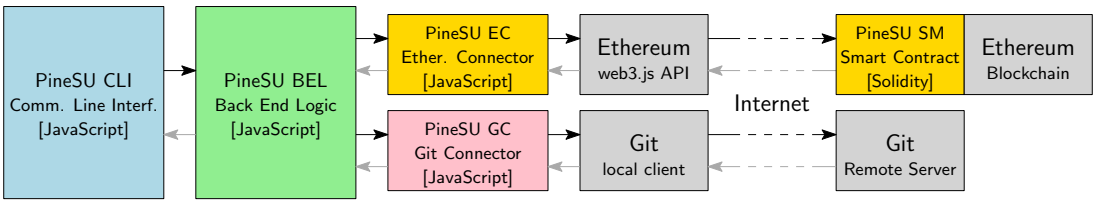

Figura 4.1: Rappresentazione dell'architettura ad alto livello di PineSU. Le frecce nere sono messaggi scatenati dalle entità sorgente corrispondenti mentre le frecce grigie sono risposte passive dell'entità interrogata.

[2] `https://github.com/ChainSafe/web3.js`.
[3] `https://github.com/steveukx/git-js`.

4.2 Moduli in dettaglio

4.2.1 PineSU CLI

Il modulo si occupa di creare l'effettiva interfaccia utente con cui è possibile interagire, le domande vengono create dal modulo apposito "inquirer", dove sono definite insieme alle risposte possibili e ai controlli di consistenza delle risposte date dall'utente.

Dopo un setup una tantum in cui all'utente vengono chieste informazioni come gli indirizzi dei suoi due wallet, a seconda delle scelte dall'utente, la prima di cui sarà quella dell'effettiva operazione da eseguire (Fig. 4.2), il modulo va a delineare il workflow preciso che descrive il ciclo vitale di una SU, il tutto richiamando all'occorrenza le librerie del modulo *PineSU BEL*.

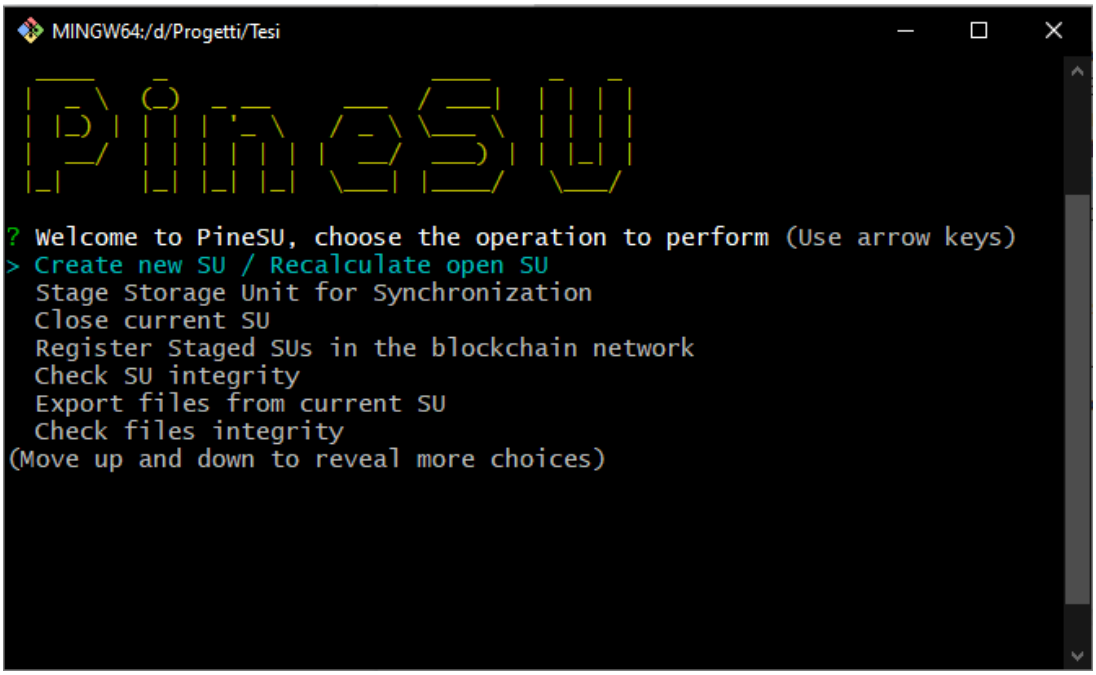

Figura 4.2: Menù principale dell'applicativo con la scelta principale dell'operazione.

4.2.2 PineSU BEL

Questo modulo è il nucleo centrale del software, si occupa dell'effettiva comunicazione con i connettori per Git e per la Blockchain, del gestire il File System andando a leggere e scrivere i file all'interno delle Storage Unit, di assegnare stringhe crittografiche ai singoli file e di creare e gestire le strutture di accumulazione crittografica.

Per la prima delle operazioni sopra citate troviamo due librerie, rispettivamente **GitLogic** e **EthLogic**, le quali si occupano essenzialmente di creare oggetti delle rispettive

classi di connettori, reperire tramite altre librerie le informazioni necessarie, chiamare le funzioni dei connettori con gli input dovuti e gestire gli output in maniera coerente con ciò che necessita *PineSU CLI*.

Andiamo ora a vedere le due classi che si occupano di creare e gestire le strutture di memorizzazione che il programma utilizza: Files e TreeList. La prima si occupa della lettura di file JSON e dei file di cui andrà letta la stringa Hash corrispondente, e della scrittura dei file JSON. Questi file JSON corrispondono ai descrittori delle SU (riguardanti la creazione e la registrazione su blockchain), alle informazioni sull'utente utilizzatore e alle strutture degli accumulatori crittografici. La seconda si occupa del reperimento delle stringhe di hash e del calcolo dei Merkle Tree binari. Essi sono fondamentali per varie operazioni che vanno dal calcolo degli hash delle directory al quello della creazione dei tre accumulatori crittografici che andremo ora a descrivere.

Il primo è il semplice SU Merkle Tree, necessario per il calcolo dell'hash corrispondente alla Storage Unit che altro non è che un Merkle Tree binario in cui ogni foglia corrisponde a un file o una directory contenuta nella SU. Questo MT verrà utilizzato anche nella fase di esportazione per generare le proof dei file esportati che serviranno per un eventuale controllo d'integrità singolo.

Il secondo è lo Storage Group, di cui ne esistono due, un Open (OSG) e un Closed (CSG). Si tratta essenzialmente di due MT binari in cui ogni foglia corrisponde all'hash di una Storage Unit "Staged", la differenza tra i due alberi è nel loro contenuto, uno contiene le SU Open, l'altro le SU Closed, differenza di cui parleremo in seguito. Le root di questi alberi verranno poi salvate all'interno della prossima struttura come foglie.

Il terzo e il più importante è il Merkle Calendar, formato da due sottoalberi in cui vengono accolte come foglie rispettivamente le istanze di OSG e di CSG. Le radici di questi due sottoalberi hanno come figli nodi corrispondenti agli anni, i quali a loro volta hanno come figli nodi corrispondenti ai mesi, i figli dei mesi saranno infine le foglie corrispondenti a ciò che chiamiamo Blockchain Synchronization Point (BSP) in quanto nodi contenenti un timestamp e la root dello SG corrispondente.
Possiamo vedere, anche grazie a Fig. 4.3, come questa struttura sia implementata tramite tre classi:

- **MerkleCalendar**: contiene i riferimenti alle due radici dei sottoalberi e mette a disposizione varie funzioni per la ricerca di hash e reperimento di determinati valori della root in un certo BSP.
- **InternalCalendar**: corrisponde a un nodo interno dell'albero, quando si aggiungono figli si può scegliere di effettuare il ricalcolo del loro hash in modo da non doverlo calcolare successivamente (semplifica l'operazione di reperimento dell'hash di un certo BSP).
- **LeafCalendar**: corrisponde a una foglia dell'albero.

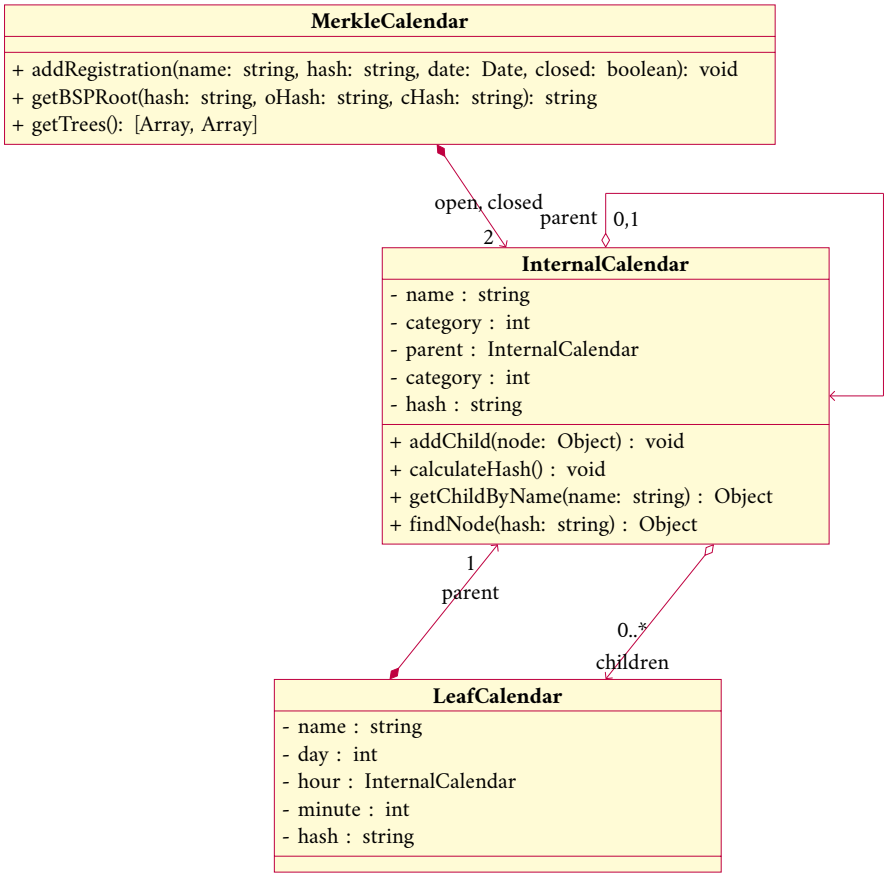

Figura 4.3: Rappresentazione UML delle classi descritte

5. Dimostrazioni d'uso per il fine preposto

6. Conclusioni e Sviluppi Futuri

7. Bibliografia