

Scuola di Scienze e Tecnologie

ESECUZIONE DI SMART CONTRACT DIMAMICI

NELL’AMBITO DELLA BLOCKCHAIN

L-31 - Classe delle lauree in Scienze e tecnologie informatiche

*Grammatico Rosario Matteo – Mtr. 095314*

*Pagan Denise – Mtr. 110677*

*Toma Matteo – Mtr. 116781*

Sommario

[Sommario 1](#_Toc92445860)

[*1.* Introduzione 1](#_Toc92445861)

[1.1 La flexchain 1](#_Toc92445862)

[*2.* Obiettivi del project 2](#_Toc92445863)

[*2.1* Tecnologie utilizzate 3](#_Toc92445865)

[2.1.1 Blockchain 3](#_Toc92445866)

[2.1.2 Smart Contract 4](#_Toc92445867)

[2.1.3 Docker 6](#_Toc92445868)

[2.1.4 Xming X Server per Windows 6](#_Toc92445869)

[2.1.5 JavaFX 6](#_Toc92445870)

[2.1.6 Drools 6](#_Toc92445871)

[2.1.7 Maven 7](#_Toc92445872)

[*3.* Architettura 8](#_Toc92445873)

[4. Implementazione 13](#_Toc92445874)

[4.1 Dockerizzazione attraverso il Dockerfile e DockerCompose 13](#_Toc92445875)

[*4.2* Script 17](#_Toc92445876)

[*4.3* Xming 18](#_Toc92445877)

[*4.4* User Interface 19](#_Toc92445878)

[*5.* Utilizzo del programma DockerFlexchain 23](#_Toc92445879)

[*5.1* Le risorse necessarie all’uso della FlexChain 23](#_Toc92445880)

[*6.* Conclusioni 24](#_Toc92445881)

# Introduzione

Negli ultimi anni si è diffusa sempre di più l’esecuzione di processi business multi-party all’interno della Blockchain.

Tali processi necessitano di una fiducia delle parti concorrenti, che non può essere data per scontata. Questa tecnologia è impiegata in tali processi poiché:

1. La natura immutabile della Blockchain: se un record di transazione contiene un errore, dovrà essere aggiunta una nuova transazione per correggere l’errore ed entrambe le transazioni saranno poi visibili.
2. La fiducia tra persone poiché non richiede intermediari nelle transazioni. Tutti sulla blockchain possono visualizzare e validare le transazioni e i dati, creando trasparenza e fiducia.

Anche se queste caratteristiche le rendono una tecnologia adatta alle esecuzioni di tali processi, dall’altra però la sua rigidità intrinseca influisce sui dati scritti e sugli *smart* *contract* in essa contenuti, rendendoli non modificabili dinamicamente in fase di esecuzione.

Ciò denota una mancanza di flessibilità, una proprietà desiderabile soprattutto se si vuole far fronte a fattori esogeni ed endogeni che potrebbero influire sui processi.

Ed è proprio da questa esigenza che nasce il progetto della *Flexchain*.

## 1.1 La flexchain

Nel progetto Flexchain si propone una soluzione model-driven per l'esecuzione flessibile di processi di business multi-party sulla blockchain.

In tale approccio si usufruisce di una blockchain pubblica e senza permessi, la quale è la più utilizzata per guadagnare fiducia in contesti non particolarmente favorevoli.

L'approccio proposto utilizza la notazione BPMN ed in particolare i modelli di coreografie.

Grazie ad esso, è possibile esprimere le interazioni tra diverse parti senza rivelare il loro comportamento interno. Esso consente di descrivere dunque un qualsiasi processo poi che produrrà uno smart contract nella Blockchain.

Un modello coreografico viene automaticamente tradotto in uno smart contract on-chain, rappresentante lo stato di esecuzione corrente, e un programma basato su regole definite con il motore Drools, corrispondente alla logica del modello coreografico, eseguita sul processore off-chain.

Tale procedimento porta diversi vantaggi:

* Per prima cosa, scinde la logica di business del modello coreografico dal suo stato di esecuzione, per consentire modifiche a run-time senza interrompere l'esecuzione.
* Secondo, l'esecuzione in off-chain della logica del modello coreografico riduce i costi e risolve alcuni problemi legati alla scalabilità.
* Infine, l'implementazione basata sulle regole permette di raggiungere una modulazione del codice, che è particolarmente adatta alla logica di business.

Successivamente alla creazione dell’on-chain e del programma off-chain, gli utenti possono proporre modifiche fornendo un ulteriore modello coreografico. Questo passaggio attiva una procedura automatica che confronta la nuova versione con la versione precedente, individuando e applicando le modifiche.

Quindi l’esecuzione automatica è suddivisa in tre passaggi:

1. Caricamento della coreografia nel framework della Flexchain, che viene poi tradotta in uno smart contract e in una raccolta di regole.
2. Esecuzione dell’istanza coreografica attraverso scambi di messaggi che seguono le regole memorizzate nel contratto.
3. Ed infine aggiornamento dell’istanza coreografica. Questo passaggio inizia con la proposta di un modello coreografico da una parte. Questo modello viene tradotto in regole queste vengono confrontate con le regole esistenti nella Blockchain. Il risultato di questo confronto è una sequenza di operazioni (aggiungere/aggiornare/cancellare regole) per adeguare la logica della coreografia.

# Obiettivi del project

Partendo dal progetto appena descritto, abbiamo scelto di implementare il relativo programma (DockerFlexchain) per renderlo più scalabile (grazie all’uso di Docker, quindi alla possibilità di essere usato da ogni sistema operativo) e di facile utilizzo per qualsiasi utente, ad esempio suddiviso in sezioni dedicate ad ogni funzionalità che il programma offre.

Nello specifico, gli obiettivi principali prefissi per tale implementazione sono stati:

1. Dockerizzare l’applicazione
2. Migliorare l’interfaccia utente
3. Aumentare il numero delle funzionalità dell’interfaccia, proponendo all’utente maggiori servizi. In particolare, aggiungere la possibilità di visualizzare il contratto caricato su Ethercan e la possibilità di visualizzare a video eventuali modifiche dopo l’update di un contratto.

Tale implementazione manterrà l’obiettivo di rendere flessibile la Blockchain.



## Tecnologie utilizzate

### 2.1.1 Blockchain

Con il termine blockchain identifichiamo un insieme di tecnologie che permettono di mantenere un registro distribuito di dati, chiamato Distributed Ledger, strutturato come una catena di blocchi contenenti le transazioni (e non solo).

Qualsiasi record o transazione aggiunta alla blockchain non può essere modificata o alterata, il che significa che le transazioni sono al sicuro dall'hacking. Un blocco è l'unità più piccola di una blockchain ed è un contenitore che contiene tutti i dettagli della transazione. Un blocco ha quattro campi o attributi primari:

Hash precedente: questo attributo memorizza il valore dell'hash del blocco precedente ed è così che i blocchi sono collegati tra loro.

Dati: questo è l'insieme aggregato di transazioni incluse in questo blocco, ovvero l'insieme di transazioni che sono state estratte, convalidate e incluse nel blocco.

Nonce: In un algoritmo di consenso "proof of work", che bitcoin utilizza, il nonce è un valore casuale utilizzato per variare l'output del valore hash. Ogni blocco dovrebbe generare un valore hash e il nonce è il parametro utilizzato per generare quel valore hash. La proof of work è il processo di verifica delle transazioni fatto in blockchain.

Hash: Questo è il valore ottenuto passando il valore hash precedente, i dati e il nonce attraverso l'algoritmo SHA-256; è la firma digitale del blocco.

**Mining**

È il processo mediante il quale le transazioni bitcoin vengono convalidate digitalmente sulla rete bitcoin e aggiunte al registro della blockchain. Viene fatto risolvendo complessi enigmi hash crittografici per verificare blocchi di transazioni che vengono aggiornati sul registro blockchain decentralizzato. Risolvere questi enigmi richiede una potente potenza di calcolo e attrezzature sofisticate. In cambio, i minatori vengono ricompensati con criptovalute, che viene poi rilasciato in circolazione.

Chi risolve i complessi enigmi è chiamato miner: l’obiettivo principale del minatore è determinare il valore nonce e quel valore nonce è il puzzle matematico che i minatori sono tenuti a risolvere per generare un hash inferiore all'obiettivo definito dalla rete per un particolare blocco.

**Ethereum**

Ethereum è una piattaforma decentralizzata (cioè nessuno può controllarla, censurarla) del Web 3.0 per la creazione e pubblicazione peer-to-peer di smart contracts creati in un linguaggio di programmazione Turing-completo. Dunque, Ethereum è:

* sicuro perché nessuno può controllarlo o manipolarlo a proprio vantaggio
* affidabile perché non c’è un solo garante ma è la rete composta da migliaia di utenti che validano le operazioni che vengono effettuate.
* è trasparente perché tutte le operazioni sono pubbliche e verificabili da chiunque.

Proprio per questi motivi, in questo progetto si è infatti usata la rete Rinkeby testnet di Ethereum.

### Smart Contract

Gli smart contract basati sulla blockchain sono contratti che possono essere stipulati e imposti senza la necessità di un'interazione umana. Sono protocolli informatici che facilitano, verificano, o fanno rispettare, la negoziazione o l'esecuzione di un contratto, e alcune implementazioni della blockchain potrebbero consentire la codifica di contratti che verranno eseguiti quando sono soddisfatte determinate condizioni. Di solito hanno anche un'interfaccia utente e spesso simulano la logica delle clausole contrattuali.

Lo smart contract sintetizzabile nello schema di funzioni tipicamente condizionali “if/then” incorporate in software o protocolli informatici, può dare esecuzione esclusivamente a ciò per cui è stato preventivamente predisposto, nella fase di programmazione del codice, dal programmatore. Pertanto, si presenta come uno strumento che da esecuzione ad una volontà precedentemente elaborata dalle parti.

Lo smart contract, qualora inserito all’interno di una blockchain, produce:

1. Automatizzazione e certezza giuridica dell’esecuzione di obbligazioni contrattuali visibili a tutti i partecipanti della rete e non solo alle parti coinvolte;
2. Trasparenza delle obbligazioni contrattuali e dei loro risultati e risvolti tali da essere preimpostati e quindi “pre-compresi” da tutti i partecipanti della blockchain;
3. Immutabilità delle transazioni registrate e quindi l’impossibilità a modificare o annullare il contratto;
4. Possibilità di trovare un accordo in assenza di fiducia.

Grazie a queste caratteristiche, permettono di risparmiare un considerevole quantitativo di risorse nelle fasi della negoziazione e di esecuzione contrattuale, accelerare le prestazioni e diminuire notevolmente la probabilità che possano verificarsi controversie e dispute tra le parti.

Tuttavia, tra i punti di maggiore criticità degli smart contract occorre considerare la loro irrevocabilità, che scatta nel momento in cui gli stessi sono inseriti all’interno della piattaforma blockchain, da cui segue l’automatica esecuzione delle prestazioni che rende inutilizzabile l’eccezione di inadempimento.

Per programmare gli Smart contract si utilizzano Solidity, che è un linguaggio di programmazione orientato agli oggetti, e Remix, un IDE che consente di sviluppare, distribuire e amministrare smart contract per la blockchain Ethereum.

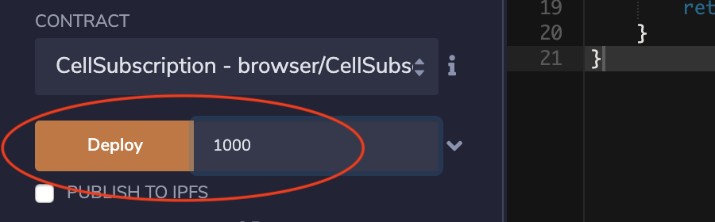
Un esempio di smart contract è questo:

Immagine che contiene testo

Descrizione generata automaticamente

Il deploy viene fatto attraverso Remix, dopo aver impostato un account con gli Ether a disposizione e un gas limit.

Si definisce successivamente un costo da assegnare al costruttore e infine si clicca su Deploy.



Dopo il Deploy si possono vedere le transazioni su [http://www.etherscan.io](http://www.etherscan.io/) attraverso l’indirizzo dello smart contract.

### 2.1.3 Docker

Docker è un framework che permette di creare, testare e distribuire applicazioni con la massima rapidità.

Docker raccoglie il software in unità standardizzate chiamate container.

Un container Docker è un formato di creazione dei pacchetti, eseguibile, leggero e autonomo che include tutto il necessario per l'esecuzione di un'applicazione, comprese le librerie, gli strumenti di sistema, il codice e il run-time.

Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi sistema operativo, e grazie ai container, qualsiasi evento non indesiderato avviene solo dentro al container senza danneggiare il sistema operativo o occupare memoria in esso. Permette quindi di tenere le risorse sotto controllo.  
Inoltre permette di distribuire il codice più rapidamente, standardizzare il funzionamento delle applicazioni, trasferire il codice in modo ottimizzato e risparmiare denaro migliorando l'utilizzo delle risorse.

### 2.1.4 Xming X Server per Windows

Xming è il principale server di sistema X Window per Microsoft Windows.

È completo di tutte le funzionalità, semplice da installare, ed è, è facilmente trasportabile, poiché è Windows nativo.

### 2.1.5 JavaFX

JavaFX è una famiglia di [software applicativi](https://it.wikipedia.org/wiki/Applicazione_(informatica)), basati sulla [piattaforma Java](https://it.wikipedia.org/wiki/Piattaforma_Java), che consente la creazione e la distribuzione di applicazioni per computer, [cellulari](https://it.wikipedia.org/wiki/Telefono_cellulare), [dispositivi portatili](https://it.wikipedia.org/wiki/Dispositivo_mobile) di vario genere, e altri tipi di piattaforme.

Esso consta di una raccolta di librerie per funzionalità grafiche, includendo un vero e proprio [linguaggio di programmazione](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione) indipendente da Java, chiamato "JavaFX script", un [linguaggio di scripting](https://it.wikipedia.org/wiki/Linguaggio_di_scripting) dichiarativo e [staticamente tipizzato](https://it.wikipedia.org/wiki/Tipizzazione_statica) fortemente orientato alla programmazione grafica.

Combinando l’utilizzo di JavaFx e di un file FXML, ovvero un formato di file che JavaFX utilizza per creare il layout delle schermate, è stato possibile creare un’interfaccia utente.

### 2.1.6 Drools

Drools è un popolare Business Rule Management System (BRMS). Con questa definizione si intende un sistema in grado di definire, eseguire, monitorare e mantenere la varietà e la complessità della logica decisionale utilizzata all'interno di un'organizzazione. Tale logica è rappresentata dalle cosiddette business rules.

È una componente principale della Flexchain, ed è stato utilizzato per i seguenti vantaggi:

* Programmazione dichiarativa: Le regole facilitano l'espressione di soluzioni a problemi difficili e verificano anche le soluzioni. Le regole sono scritte in un linguaggio meno complesso, quindi sono facilmente leggibili e verificabili.
* Separazione logica e dati: i dati risiedono negli oggetti dominio e la logica aziendale si trova nelle regole. A seconda del tipo di progetto, questo tipo di separazione può essere molto vantaggioso.
* Velocità e scalabilità: L'algoritmo di Rete Object Oriented su cui è scritto Drools è già un algoritmo testato. Quindi un’applicazione diventa molto scalabile: se ci sono frequenti richieste di modifica, è possibile aggiungere nuove regole senza dover modificare le regole esistenti.
* Centralizzazione della conoscenza: Usando le regole, si crea una *knowledge base* (una repository di conoscenza) che è eseguibile. Idealmente, le regole sono così leggibili da poter essere utilizzate anche come documentazione.
* Explanation Facility: I sistemi di regole forniscono effettivamente una "struttura di spiegazione" essendo in grado di registrare le decisioni prese dal motore di regole insieme al motivo per cui sono state prese le decisioni.
* Regole comprensibili: Creando modelli di oggetti e, facoltativamente, lingue specifiche del dominio che modellano il dominio del problema, si possono scrivere regole molto vicine al linguaggio naturale.

### 2.1.7 Maven

Maven è un tool di build automatizzato, usato principalmente nei progetti Java.

Maven si occupa principalmente di due aspetti principali nella compilazione del software: come il software viene compilato e le sue dipendenze.

Al contrario dei tool precedenti come Apache Ant, Maven usa le sue proprie procedure per il processo di compilazione.

**2.1.8 Infura**

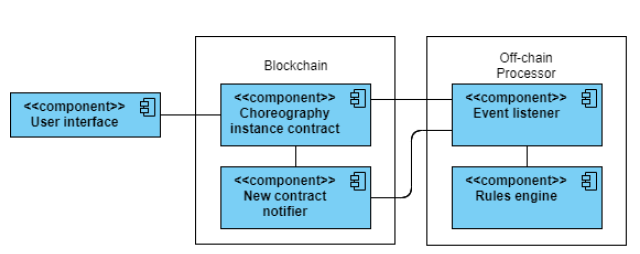
Infura: è un sistema che permette alle App di processare informazioni sulla blockchain di Ethereum senza avere un full node installato sul proprio dispositivo.

Fornisce accesso istantaneo HTTPS e WebSocket alle reti Ethereum e IPFS. Ci si può connettere al Web 3.0 senza dover avviare e mantenere la tua infrastruttura.

È stato utilizzato per poter effettuare le transazioni tramite il wallet della rete Testnet Rinkeby di Ethereum quando si utilizza la Flexchain.

# Architettura

Il programma *Dockerflexchain* è strutturato con la seguente architettura:



* Blockchain: ha invece il compito di memorizzare solo lo stato di un’istanza.

Ogni istanza della coreografia corrisponde ad uno smart contract nella Blockchain che ha lo scopo di mantenere lo stato di esecuzione di quella stessa istanza.

Gli smart contract sono automaticamente completati dopo il caricamento dei corrispettivi modelli coreografici.

La memorizzazione riguarda in particolar modo:

* il set di regole della coreografia
* il set di variabili del modello (string, uint, bool)
* il set degli stati degli elementi rappresentati da valori enum (disabled, enabled, completed).

New contract notifier: questo componente viene utilizzato per stabilire la connessione tra un nuovo contratto intelligente distribuito nella blockchain e il processore off-chain. Questo è uno smart contract a sua volta, che ha il compito di pubblicare un evento contenente l'indirizzo di un contratto appena generato.

* Off-chain Processor: comprende l’event listener e usa il Rules engine (una componente stateless che valuta una regola sulla base delle informazioni fornite dell’event listener)

Si occupa principalmente di recuperare le regole da un contratto e di valutarle.

Per farlo, sfrutta una componente chiamata *event listener*, altro non è che un componente che rimane sempre in ascolto, aspettando l’occorrenza di qualche evento.

Una volta ricevuto un evento, recupera dallo smart contract la regola associata al messaggio da eseguire, e chiede al *rule engine* di valutarla.

Il *rule engine* a sua volta è un componente stateless, che serve a valutare una regola sulla base delle informazioni fornite dall’*event listener*. La valutazione della regola può aggiornare lo stato del contratto. In questo caso abbiamo usato *Drools Engine*.

* La UI e il Translator fanno parte dello stesso macro-componente:

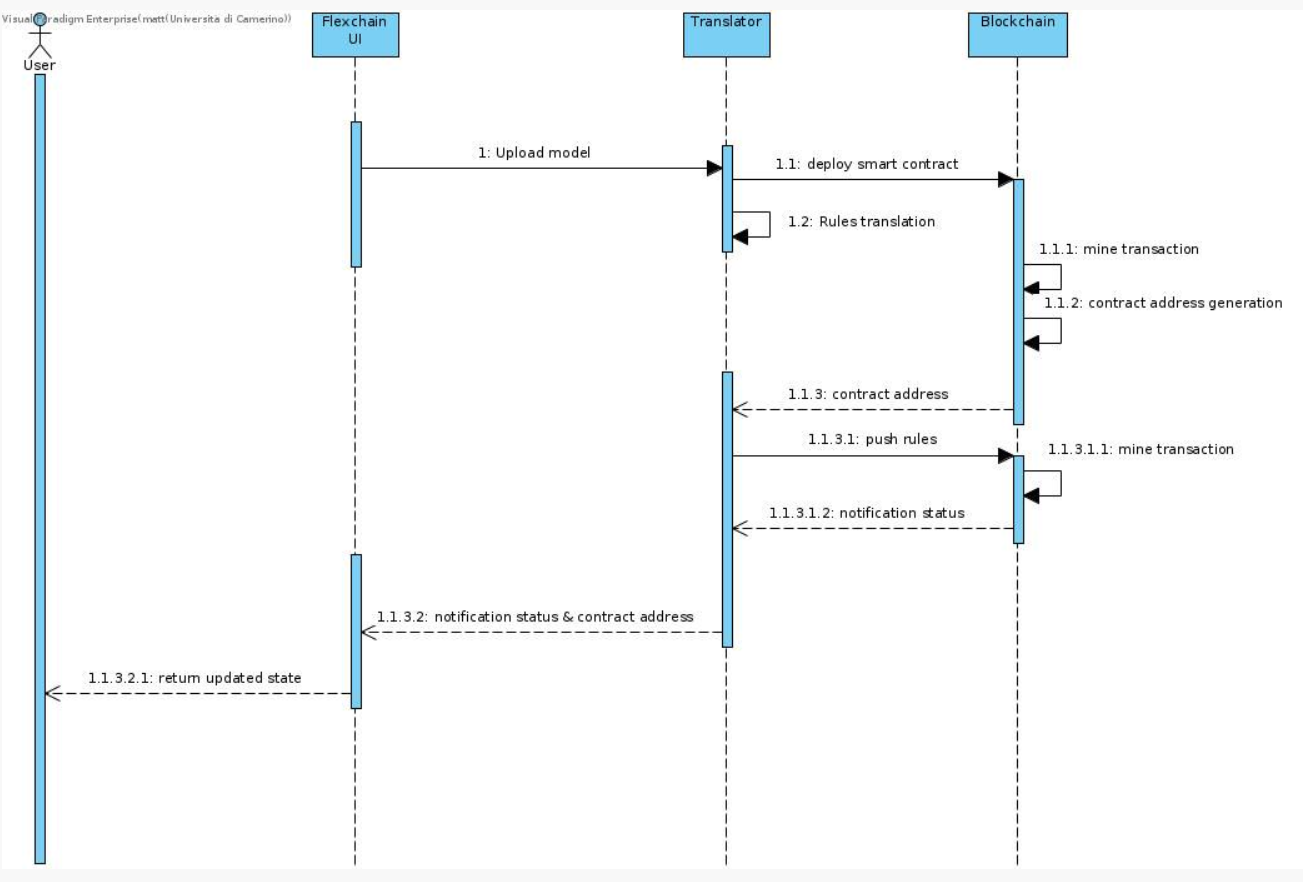
User interface (UI): un’interfaccia grafica che fornisce delle funzionalità all'utente per istanziare, eseguire e aggiornare una coreografia. Permette all’utente di interagire con l’engine e la conoscenza delle informazioni.

Istanzia automaticamente gli smart contract dopo il caricamento dei modelli corrispondenti di coreografia. Genera le regole (una regola per ogni messaggio) rappresentanti nella logica di una coreografia e aggiunge le regole nello smart contract e lo distribuisce:

* Confrontare le regole;
* Aggiorna le regole;
* Carica le regole.

**Diagrammi**

**Upload Model**

****

Questo diagramma rappresenta gli step necessari per il caricamento di un modello di coreografia.  
La prima cosa che l’utente dovrà fare sarà quella di cliccare sul tasto *Load Custom Model.*

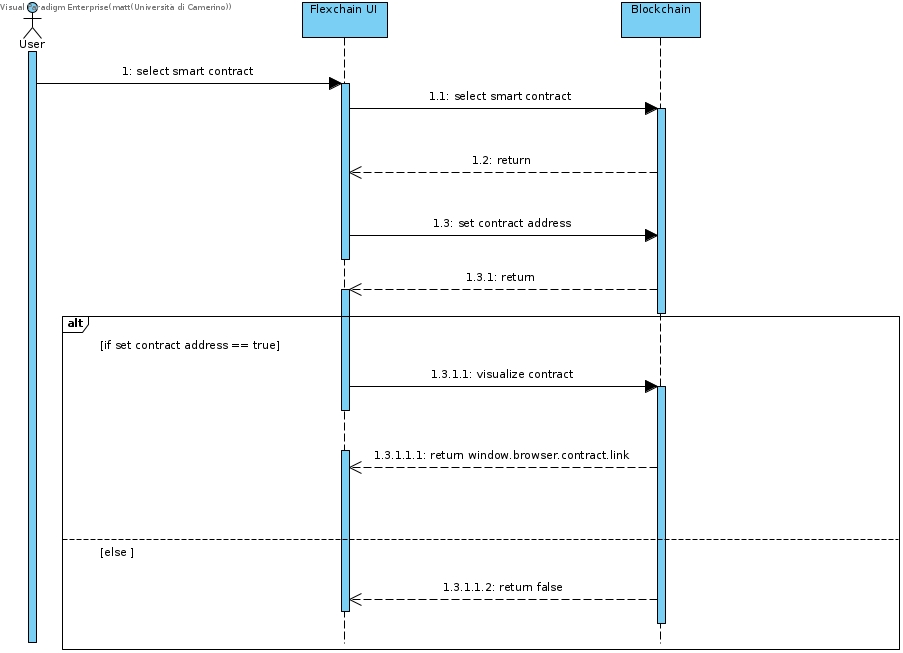
A quel punto, si aprirà un’interfaccia che permetterà di scegliere che modello caricare.

Una volta scelto viene data la conferma e il modello viene caricato.

Una volta caricato il modello esso viene tradotto regola per regola, in uno smart contract, che verrà poi inviato alla Blockchain.

Quest’ultima, si occupa di minare la transazione e generare l’indirizzo del contratto, che sarà successivamente notificato all’utente insieme allo stato del caricamento delle regole.

**Visualize Contract**



Viene data l’opportunità di visualizzare su [***https://rinkeby.etherscan.io/***](https://rinkeby.etherscan.io/)il proprio contratto.

L’utente ha la possibilità di inserire il contratto specificando l’indirizzo dello stesso oppure sfruttando il load model.

Il contratto viene visualizzato dopo il controllo da parte della Blockchain sulla validità dell’indirizzo.

**Execute Message**

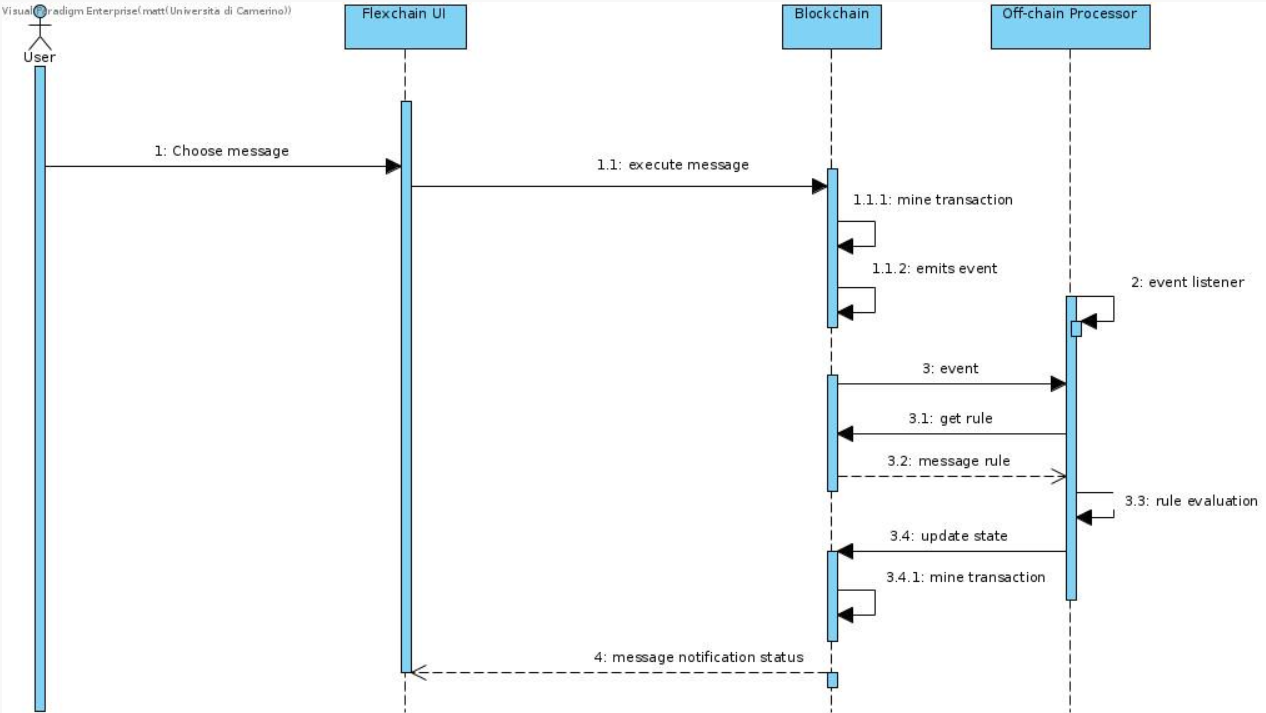
L’utente seleziona un messaggio da eseguire e definisce gli input corrispondenti da passare allo smart contract.

Quest’ultimo, dopo aver salvato le informazioni sugli input, emette l’evento corrispondente al messaggio da eseguire.

L’off-chain processor, recupera prima di tutto la regola corrispondente dallo smart contract e, poi, se tutte le condizioni sono state soddisfatte, ne esegue la logica.

I risultati dell’esecuzione sono memorizzati sullo smart contract, evolvendo lo stato della corrispettiva istanza di coreografia.

L’utente riceverà un avviso di completamento grazie all’evoluzione dello stato di esecuzione.



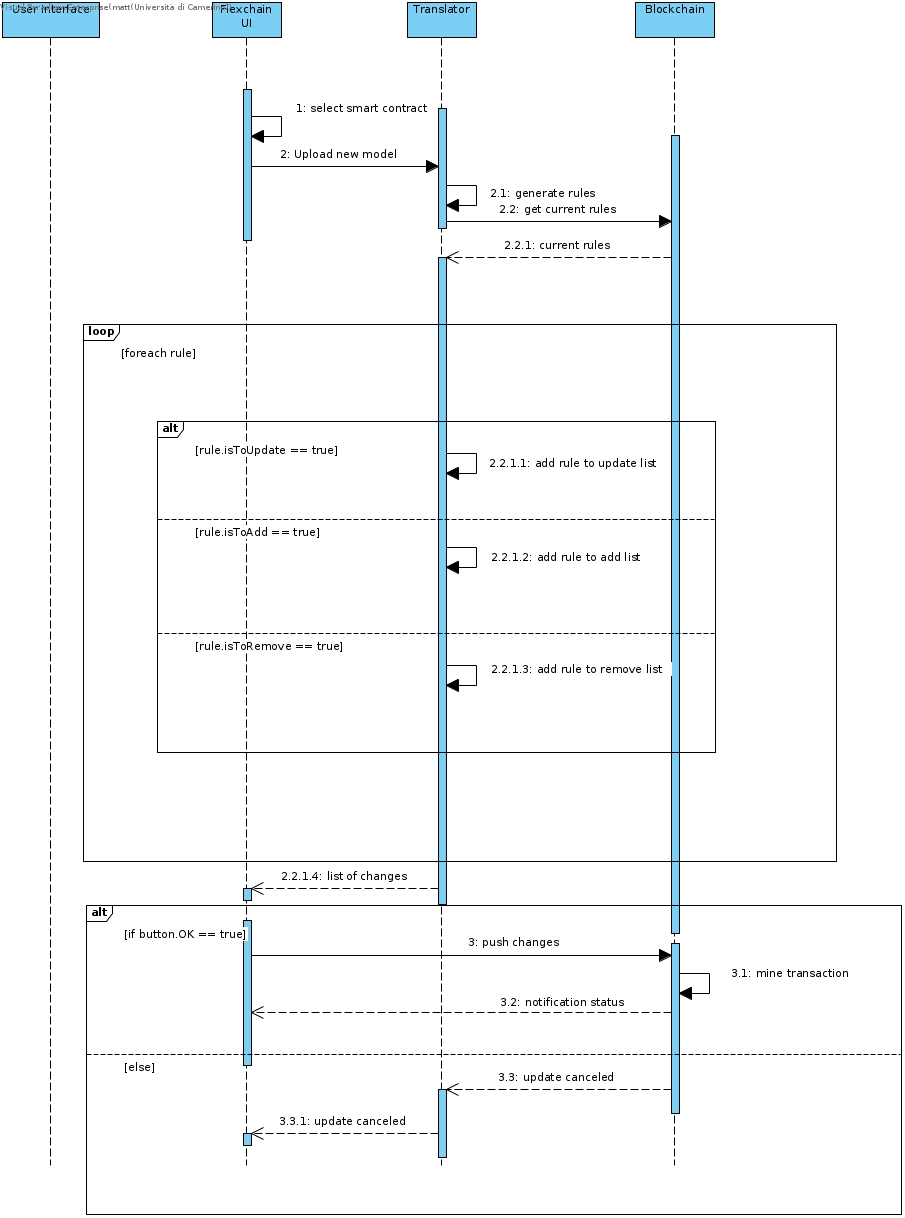
**UPDATE MODEL**

Con l’update model l’utente seleziona il modello aggiornato da caricare.

Una volta caricato viene tradotto e trasferito alla Blockchain dal Translator, nella stessa modalità del *load model.*

Le regole vengono confrontate quelle vecchie con quelle nuove e vengono mostrate a video sia quelle modificate, sia quelle aggiunte.

Una volta confermate le modifiche da parte dell’utente le regole verranno immesse nella Blockchain, la quale darà conferma dell’avvenuto cambiamento.



# Implementazione

Per Implementare la Flexchain, si è lavorato su più livelli: prima l’architettura, quindi il diagramma delle classi e i casi d’uso, poi la Dockerizzazione della Flexchain e infine l’interfaccia.

Per poter compilare ed eseguire le classi, l’interfaccia e il processore off-chain, si è utilizzato il comando

*mvn clean package assembly:single*

che crea i Jar che potranno essere eseguiti tramite il comando

*java -jar file.jar*

I Jar che verranno eseguiti sono quelli per aprire l’interfaccia grafica e l’off-chain processor, rispettivamente

*translator-1.0-SNAPSHOT-jar-with-dependencies.jar*

*offchainprocessor-1.0-SNAPSHOT-jar-with-dependencies.jar*

## Dockerizzazione attraverso il Dockerfile e DockerCompose

Per poter Dockerizzare la Flexchain, si è scelto di eseguire, in due servizi separati del file compose, i jar sopracitati (il processore off chain e il translator insieme alla gui).

In questo modo, nei due container creati dal comando *docker compose up,* l’esecuzione dei jar compilati da Java fa eseguire in modo completo la Flexchain.

una volta eseguiti i comandi di compose si otterranno due container chiamati

- OffChainProcessor

- Dockerflexchain

che corrispondono all’interfaccia comprendente il translator e all’off chain processor.

Compose è uno strumento per la definizione e l'esecuzione di applicazioni Docker multi-container. Con Compose, si utilizza un file YAML per configurare i servizi dell'applicazione.

Quindi, con un singolo comando, si creano e si avviano tutti i servizi dalla configurazione.

Grazie al Dockerfile è possibile riprodurre l'ambiente di una qualsiasi app ovunque.

I comandi principali utilizzati per Dockerizzare la nostra applicazione sono stati:

1. FROM: Inizializza una nuova fase di compilazione e imposta l'immagine di base per le istruzioni successive. Pertanto, un Dockerfile valido deve iniziare con un'istruzione FROM.
2. ENV: L'istruzione ENV imposta la variabile d'ambiente. Questo valore sarà nell'ambiente per tutte le istruzioni successive nella fase di compilazione e può essere sostituito anche successivamente. Il valore verrà interpretato per altre variabili d'ambiente.
3. RUN: L'istruzione RUN eseguirà qualsiasi comando in un nuovo livello sopra l'immagine corrente e confermerà i risultati. L'immagine di commit risultante verrà utilizzata per il passaggio successivo nel Dockerfile.  
   Si divide in due forme:
4. RUN *<command>*

(modalità shell, il comando è eseguito in una shell, che di default è /bin/sh -c su Linux or cmd /S /C su Windows)

1. RUN ["eseguibile", "param1", "param2"]

(modalità exec)

1. CMD: Lo scopo principale di un CMD è fornire le impostazioni predefinite per un contenitore in esecuzione. Questi valori predefiniti possono includere un eseguibile o possono omettere l'eseguibile.

Il contenuto del Dockerfile è il seguente:

|  |  |
| --- | --- |
| *FROM debian* | Definizione del sistema operativo |
| *RUN apt update -y* | Aggiorna la lista dei pacchetti di Debian alle ultime versioni |
| *RUN apt upgrade -y* | Aggiorna e installa i pacchetti |
| *RUN apt install sudo -y* | Consente agli utenti di eseguire programmi con i privilegi di sicurezza di un altro utente |
| *RUN apt install xdg-utils -y* | Insieme di strumenti che consente alle applicazioni di integrarsi facilmente con l'ambiente desktop dell'utente |
| *RUN apt install firefox-esr -y* | Browser per poter aprire l'indirizzo dello smart contract dall'Interfaccia della Flexchain |
| *RUN apt install software-properties-common -y* | Fornisce un'astrazione dei repository apt usati. Consente di gestire facilmente la distribuzione e le fonti software di fornitori di software indipendenti. |
| *RUN apt update -y* | Aggiorna la lista dei pacchetti da installare |
| *RUN apt install -yqq --no-install-recommends* | Apt non considera i pacchetti consigliati come una dipendenza da installare. Consente di velocizzare la build dell’immagine. |
| *RUN apt install git -y* | Installazione degli strumenti git di github per scaricare i file relativi alla flexchain |
| *RUN apt install unzip -y* | Installazione del pacchetto per aprire file zip,tar,ecc |
| *RUN apt install curl -y* | Comando per trasferire dati da o verso un server remoto |
| *RUN apt-cache search openjdk | grep 17* | Si cerca la versione 17 di Openjdk |
| *RUN apt install openjdk-17-jdk -y* | Installazione di Openjdk |
| *RUN apt install -qqy x11-apps* | Visualizzatore di applicazioni per X Windows System |
| *RUN apt install maven -y* | Installazione di Maven per compilare |
| *RUN apt install x11-xserver-utils -y* | Installazione pacchetti utili per far interagire XLaunch dell’host con il container, per poter eseguire l’Interfaccia Grafica della Flexchain nel container |
| *RUN apt install mesa-utils -y* | Libreria per il rendering della grafica |
| *RUN apt install libgl1-mesa-glx -y* | Libreria per il rendering della grafica |
| *RUN apt install libgl1-mesa-dri -y* | Libreria per il rendering della grafica |
| *RUN apt install libgl1-mesa-dev -y* | Libreria per il rendering della grafica |
| *RUN apt install libgtk-3-0 -y* | Libreria per il rendering della grafica |
| *RUN apt install libxext6 -y* | Libreria per il rendering della grafica |
| *RUN apt install libxrender1 -y* | Libreria per il rendering della grafica |
| *RUN apt install libxtst6 -y* | Libreria per il rendering della grafica |
| *RUN apt install libxi6 -y* | Libreria per il rendering della grafica |
| *RUN apt install net-tools -y* | Installazione per il toolkit del networking (permette di ottenere l'ip del container ad esempio) |
| *RUN apt install mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev -y* | Libreria per il rendering della grafica |
| *RUN apt update -y* | Aggiorna la lista dei pacchetti da installare |
| *RUN apt upgrade -y* | Aggiorna e installa i pacchetti |
| *RUN export XAUTHORITY=~/.Xauthority* | Esporta il file di configurazione di Xauth |
| *ENV DISPLAY host.docker.internal:0* | Impostazione del Display per poter visualizzare l’interfaccia Grafica usando come IP l'host di Docker |
| *RUN useradd -ms /bin/bash admin* | Per motivi di sicurezza, i prossimi pacchetti saranno installati all’interno dell’utente admin e non root |
| *RUN adduser admin sudo* | Aggiunta dell'utente admin negli utenti del sistema operativo Debian del container |
| *USER admin* | Si setta l'utente |
| *WORKDIR /home/admin* | Si comunica a Docker quale sarà la directory su cui lavorare d'ora in poi |
| *RUN git clone https://github.com/Gasu16/DockerFlexchain.git* | Clonazione delle repository necessarie per eseguire la Flexchain |
| *RUN git clone https://github.com/Gasu16/OffChainProcessor.git* | Clonazione delle repository necessarie per eseguire la Flexchain |
| *RUN mkdir OpenJFX* | Creazione cartella per installare OpenJFX |
| *WORKDIR OpenJFX/* | Si comunica a Docker quale sarà la directory su cui lavorare d'ora in poi |
| *RUN curl https://download2.gluonhq.com/openjfx/17.0.0.1/openjfx-17.0.0.1\_linux-x64\_bin-sdk.zip -o openjfx-17.0.0.1.zip* | Scaricamento del pacchetto openjfx |
| *RUN unzip openjfx-17.0.0.1.zip* | Decompressione delle risorse |
| *WORKDIR /home/admin/DockerFlexchain* | Da qui in poi questa sarà la working directory del container |
| *CMD java --module-path /home/admin/OpenJFX/javafx-sdk-17.0.0.1/lib --add-modules=javafx.controls,javafx.fxml -Dprism.order=sw -jar target/Translator-1.0-SNAPSHOT-jar-with-dependencies.jar* | Si esegue il jar file della Flexchain attraverso java, passandogli i relativi moduli controls e .fxml, disabilitando attraverso l’opzione -Dprism.order=sw l’accelerazione hardware. |

Per definire servizi che compongono l’app in modo che possano essere eseguiti insieme in un ambiente isolato si usa un file chiamato docker-compose.yml.

Con docker-compose up docker andrà ad eseguire i servizi definiti nel file *docker-compose.yml*

Il contenuto del DockerCompose [per il ramo di Windows su GitHub] è il seguente:

|  |  |
| --- | --- |
| *version: '3.3'* | Versione del Docker–compose |
| *services:* | Definizione dei servizi (2 sottocontainer) da avviare |
| *docker\_bpmn:* | Primo container |
| *working\_dir: /home/admin/DockerFlexchain* | Path in cui verranno memorizzati i file scaricati durante la build del Dockerfile e del Docker Compose |
| *image: matteotoma98/dockerflexchain:latest* | Immagine che verrà creata su DockerHub |
| *offchain-processor:* | Secondo container, l’OffChain |
| *working\_dir: /OffChainProcessor/offchainprocessor* | Path in cui verranno memorizzati i file scaricati durante la build del Dockerfile e del Docker Compose |
| *container\_name: offchainprocessor* | Nome del secondo container |
| *image: matteotoma98/offchainprocessor:latest* | Immagine che verrà creata su DockerHub |
| *command: java -jar /home/admin/OffChainProcessor/target/offchainprocessor-1.0-SNAPSHOT-jar-with-dependencies.jar* | Si esegue il processore offchain tramite il jar |

## Script

Per facilitare l'utente nell’esecuzione del programma, utilizzando un solo comando, si è pensato di creare una sorta di "shortcut memory note".

L’utente in questa maniera non è costretto a scrivere i comandi uno per uno, ma potrà utilizzare semplicemente uno script che cambia sulla base del sistema operativo da lui in possesso.

**

a. Script per Windows

In questo script per windows, ci limitiamo a dare i comandi per scaricare l'immagine Docker da hub.docker.com e ad avviarlo con compose up.

*Immagine che contiene testo

Descrizione generata automaticamente*

b. Script per Linux

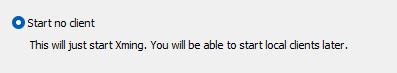
Con questo script in bash per i sistemi operativi GNU/Linux, per prima cosa ci occupiamo di garantire l'esecuzione del server grafico con xhost, successivamente, viene eseguito la build dandogli in input il Dockerfile associato, e infine vengono eseguiti le istanze dei due container con il comando finale.

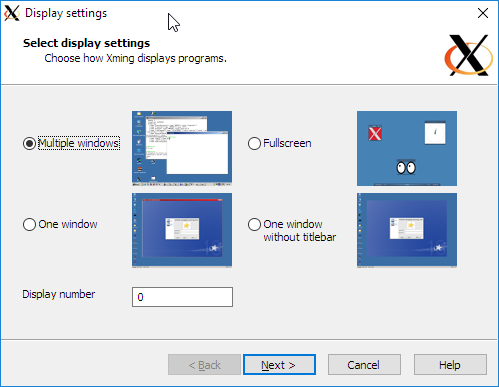
## Xming

In questo progetto è stato utilizzato Xming, che è un server di visualizzazione X11 per sistemi operativi Microsoft.

Diversamente dai sistemi GNU/Linux, i quali hanno già un server X integrato, per poter visualizzare la UI in tali sistemi è necessaria l’installazione di tale server.

Di seguito la configurazione utilizzata:

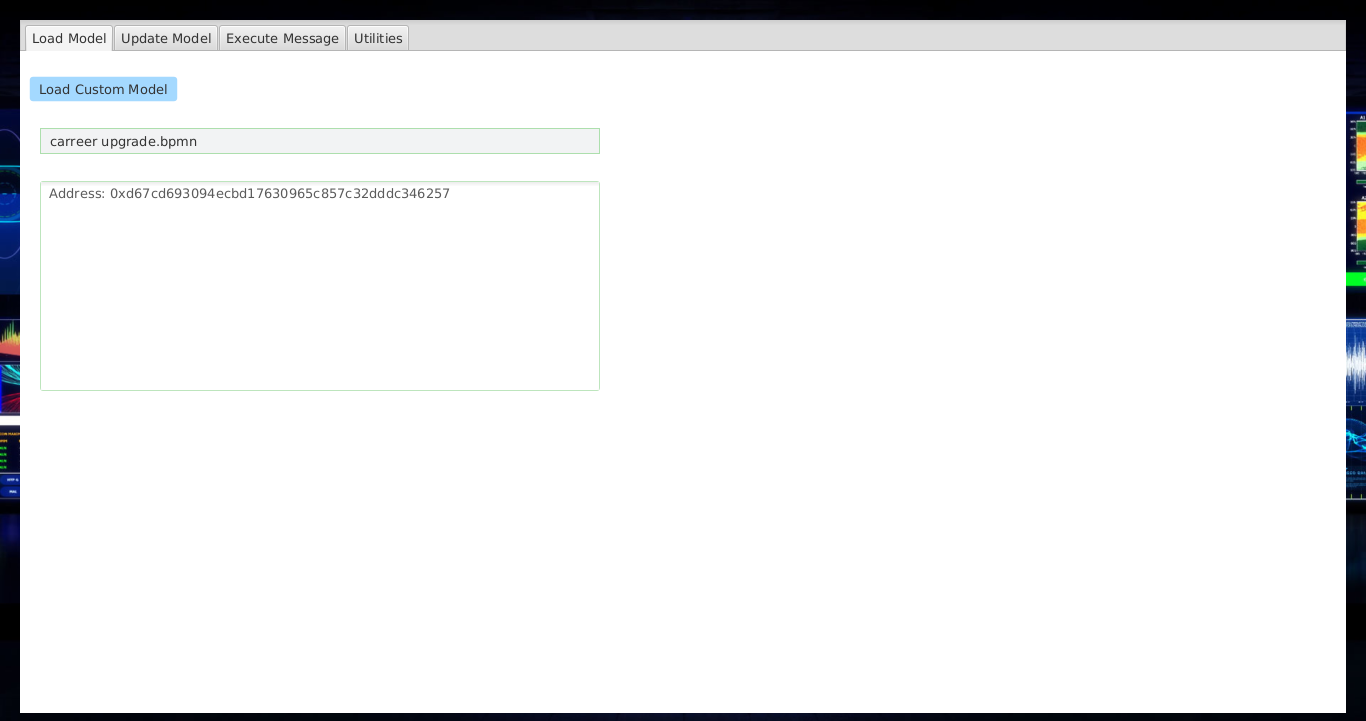


1. 
2. Immagine che contiene testo

   Descrizione generata automaticamente

## User Interface

L’interfaccia utente è stata divisa nelle quattro principali funzioni che un utente può sfruttare per utilizzare la Flexchain.

Load model

Nella finestra Load Model, l’utente ha la possibilità di caricare il modello importandolo dal proprio pc.

I file ammessi per l’inserimento sono .bpmn.

Una volta caricato il modello, viene tradotto (essendo il traduttore nello stesso componente) e deployato nella blockchain.

Inoltre, l’utente riceverà un messaggio, il quale confermerà l’avvenuto successo del caricamento del modello.

Comparirà inoltre nella text area sottostante l’indirizzo associato al modello.

Update model

Immagine che contiene testo

Descrizione generata automaticamente

Nell’Update Model, l’utente avrà la possibilità di inserire manualmente l’indirizzo di un contratto grazie al bottone “Set contract adddress”, nonché aggiornare il modello già esistente.

Messaggi di conferma/ errore verranno visualizzati nel caso in cui le operazioni dovessero andare a buon fine o meno.

L’update model prende in input il modello BPMN aggiornato e l'indirizzo di contratto che identifica l'istanza coreografica da aggiornare. A questo punto il traduttore analizza il file ed estrae le regole che verranno confrontate con quelle precedenti. A tal fine, viene chiamata una funzione getter dello smart contract per recuperare dalla blockchain gli identificatori di tutti i messaggi nella versione corrente dell'istanza coreografica. Quindi, le relative regole vengono prese dallo smart contract e infine confrontate con quelle nuove. Al termine di questo confronto vengono attivate le seguenti operazioni:

• se una nuova regola è già presente e coincide con una esistente, non si interviene sul contratto;

• se una nuova regola è già presente ma viene modificata, quella esistente viene aggiornata nel contratto;

• se una regola è completamente nuova; viene aggiunta nel contratto e lo stato dell'elemento messaggio corrispondente viene impostato su DISABLED;

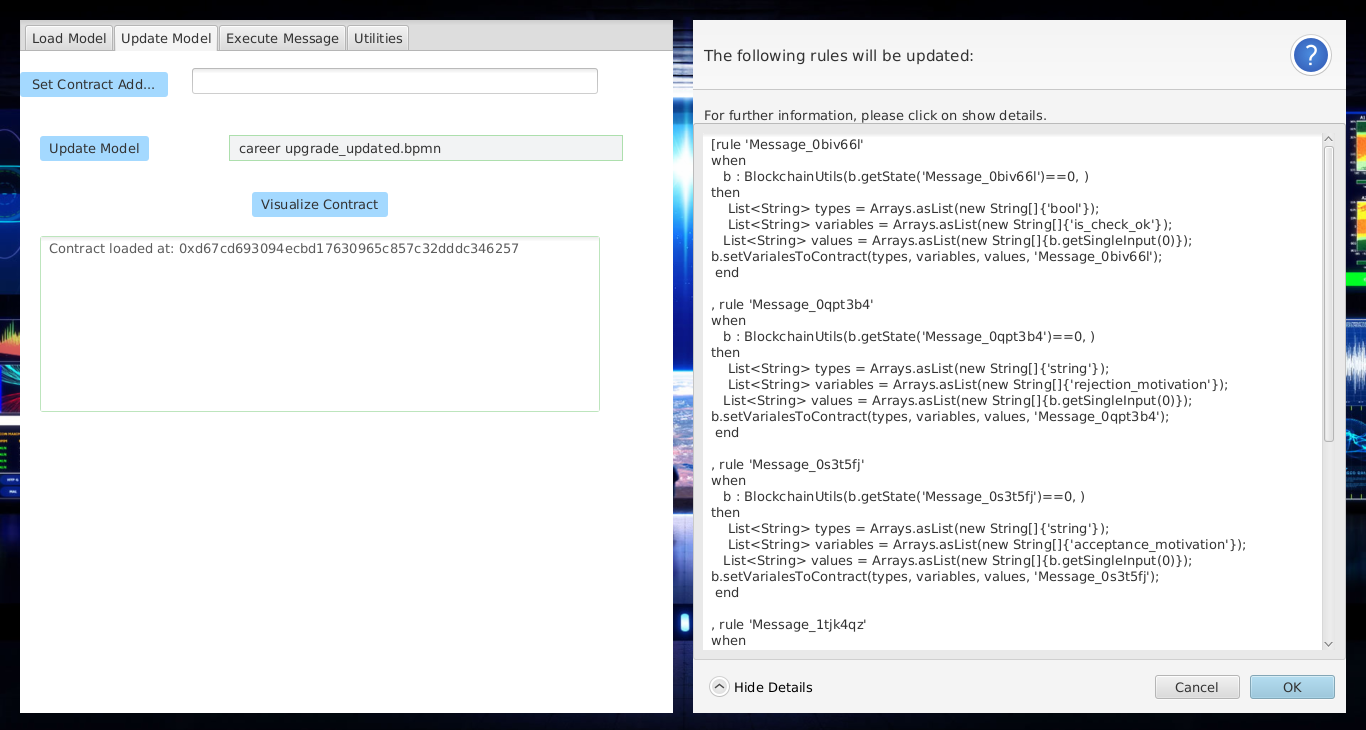
• se una vecchia regola non è più presente, viene cancellata dal contratto.

Per ampliare i servizi offerti all’utente, si è pensato fosse utile poter visualizzare il contratto. Dopo aver infatti caricato il modello è possibile visualizzarlo semplicemente cliccando il bottone “Visualize Contract”, il quale aprirà il relativo link di Etherscan con tutte le informazioni sul contratto e le relative transazioni.

Nel precedente progetto l’Update model si limitava ad aggiornare il modello con le nuove regole.

Con la nuova implementazione invece, l’utente può scegliere se rendere effettive le modifiche. Verrà infatti visualizzate a schermo tutte le regole che verranno modificate o aggiunte.

Il metodo collegato ad Update Model non fa altro che confrontare le regole del modello precedente con quello appena inserito.

Esempio di regole da aggiornare.

Execute Message

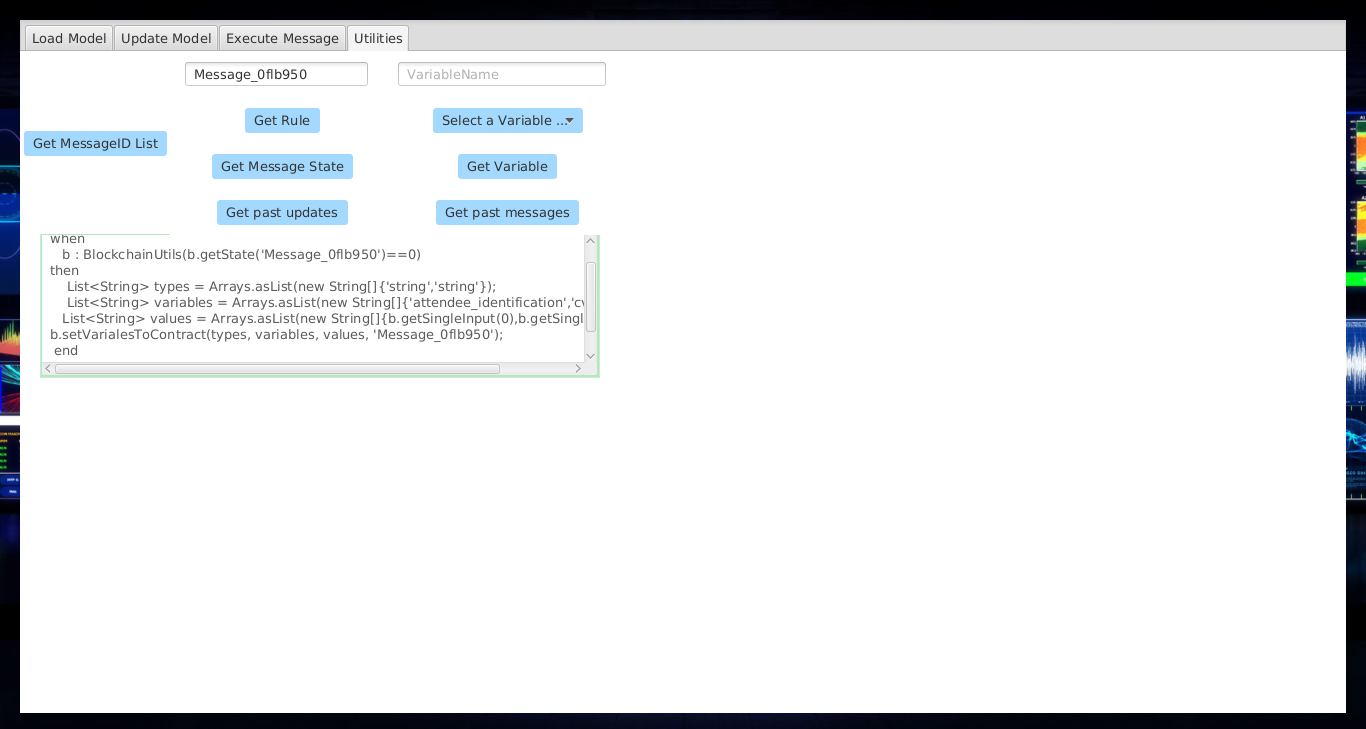
Con la finestra Execute Message l’utente sceglie il messaggio da eseguire e definisce gli input da passare allo smart contract, cliccando sull’apposito bottone “Execute message”, da cui partirà la transazione nella blockchain. (L’invio di un messaggio innesca l'esecuzione di una funzione dello smart contract,)

Immagine che contiene testo

Descrizione generata automaticamente

Quando la funzione viene richiamata, lo stato del messaggio cambia da DISABLED a ENABLED, il che significa che è attivo e può essere elaborato.

A questo punto, lo smart contract emette un evento catturato dal processore off-chain, che prima effettua una chiamata al contratto per ottenere la regola corrispondente (la stessa mostrata prima) e poi valuta la regola con il motore di Drools. Se tutte le condizioni vengono valutate con successo, gli input dell'utente vengono scritti nel contratto e sono accessibili per un controllo successivo. Quando l'esecuzione è terminata, l'elemento è finalmente impostato su COMPLETED, che registra l'esito positivo.

Utilities

In ultimo, ci sono tutte le utilities che l’utente può sfruttare.

Infatti, l’utente può:

* Visualizzare la lista degli Id dei messaggi (tasto: “Get MessageId List”);
* Inserendo il MessageId nell’apposito spazio e cliccando poi su “Get Rule” è possibile visualizzare la relativa regola.
* “Get Message State” e “Get Past Updates” ritornano rispettivamente lo stato dei messaggi e gli aggiornamenti passati.

Nello specifico lo stato dei messaggi sarà contrassegnato come:

* + ENABLE: il messaggio è attivo e può essere processato;
  + DISABLED: il messaggio non è attivo e non può essere processato;
  + COMPLETED: l’esecuzione del messaggio è terminata.
* Inserendo il nome della variabile e selezionando il relativo tipo, è possibile ricercare all’interno del contratto la variabile inserita e se essa è contenuta io meno in esso (tasto: “Get Variable”);
* Infine, con “Get past message” è possibile visualizzare i messaggi precedenti.

Come ultima cosa, a livello di codice è stata aggiunta una completa gestione degli errori, attraverso l’uso di vari try-catch. Grazie ad essi si ha un controllo sulla validità dei contratti passati come input e sulle azioni preliminari che vengono richieste per l’utilizzo di alcune funzionalità.

In tutti casi vengono segnalati agli utenti le problematiche hanno portato alla generazione dell’errore e le eventuali azioni da compiere per poter sfruttare le funzionalità desiderate.

# Utilizzo del programma DockerFlexchain

L'uso di questo programma è semplice, grazie all'utilizzo di Docker il sistema è adattabile e installabile in qualsiasi OS, che sia Windows e che sia GNU/Linux.

Come primo step, dobbiamo scaricare da Github la repository necessaria, ovvero docker\_bpmn, aprendo un terminale e dando il comando: *git clone https://github.com/Gasu16/docker\_bpmn.git .*

Questo scaricherà da Github la repository in una cartella chiamata *docker\_bpmn*.

È possibile anche specificare la cartella dove verrà scaricato.

Una volta scaricato, è sufficiente lanciare lo script per scaricare l’immagine da Dockerhub e avviare la GUI e l’off-chain processor tramite Docker compose up.

Sui sistemi GNU/Linux basta lanciare lo script run: *./run*

Sui sistemi Windows bisogna lanciare invece lo script: *run.bat*

Entrambi gli script sono semplicemente degli eseguibili che una volta aperti, provvedono automaticamente all'installazione del programma, andando a scaricare le due repository necessarie: OffChainProcessor e DockerFlexchain.

## Le risorse necessarie all’uso della FlexChain

Le risorse necessarie sono dunque:

1. Interfaccia grafica e middleware:

https://github.com/Gasu16/DockerFlexchain

1. Off-chain Processor:

https://github.com/Gasu16/OffChainProcessor

1. Docker, suddiviso in due branch:

Linux: <https://github.com/Gasu16/docker_bpmn/tree/Linux> (Utilizzabile tramite lo script run)

Windows: <https://github.com/Gasu16/docker_bpmn/tree/Windows> (Utilizzabile tramite lo script run.bat)

1. Immagini da pullare per scaricare la Flexchain su Docker:

<https://hub.docker.com/r/matteotoma98/dockerflexchain>

<https://hub.docker.com/r/40ffa173/flexchain>

<https://hub.docker.com/r/40ffa173/offchainprocessor>

# Conclusioni

Con il passare del tempo e l'evolvere delle tecnologie che riguardano la blockchain, abbiamo visto come diventa sempre più cruciale la necessità di avere flessibilità all'interno di una blockchain, specialmente negli ultimi anni in cui abbiamo visto come le situazioni possono cambiare rapidamente e c'è il bisogno di adattarsi in fretta a particolari contesti, anche inaspettati.

L'evolvere di queste situazioni in modo rapido fa sì che la flessibilità non sia più una semplice caratteristica secondaria, soprattutto con l'avvento della pandemia di covid-19, essa sta acquistando sempre più una rilevanza primaria.

Il fine ultimo del progetto Flexchain è proprio quello di estendere questa flessibilità anche alla tecnologia blockchain.

Si è data dunque la possibilità all'utente di modificare le regole già stabilite nel contratto, con nuove regole che verranno visualizzate a video su una apposita finestra, durante l'update del modello.

Questo grazie ad un’architettura separata in off-chain e on-chain.

I vantaggi portati da questa implementazione "flessibile" della Blockchain sono molteplici, in particolare la capacità delle parti di adattarsi a nuovi contesti e di aggiornare le regole con il passare del tempo, ove richiesto.

In conclusione, abbiamo dimostrato che una struttura tipicamente rigida come la Blockchain ha la possibilità di acquisire la caratteristica della flessibilità, una qualità che in futuro non potrà più essere ignorata.