UNIVERSITÀ DEGLI STUDI
DI SALERNO

MuSe Remix Plugin

Mainteinance Report

Docente / Tutor

Andrea De Lucia

Gerardo Iuliano

Studenti

Carmine Calabrese (CC)
0522501853

Daniele Carangelo (DC)
0522501696

Data	Versione	Autori
25/08/2025	1.0	DC, CC

Table of Contents

1 Introduzione **2**

2 Reverse Engineering **2**
2.1 System and Object Design 2
2.1.1 Moduli (Packages) 2
2.1.2 Componenti Off-The-Shelf 2
2.2 Requirements Analysis 3
2.2.1 Requisiti Funzionali 3
2.2.2 Requisiti Non Funzionali 4
2.2.3 Use Cases 4

3 Forward Engineering **8**
3.1 Change Requests 9
3.2 Impact Analysis 9
3.3 Requirements Analysis 10
3.3.1 Requisiti Funzionali 10
3.3.2 Requisiti Non Funzionali 10
3.3.3 Use Cases 13
3.3.4 Sequence Diagram 14
3.4 System and Object Design 16
3.4.1 Sottosistemi 16
3.4.2 Architettura 16
3.5 Testing 17
3.6 Installazione del Tool 17

1 Introduzione

MuSe (Mutation Seeding tool) è uno strumento basato su tecniche di mutation testing progettato per generare **benchmark di sicurezza** in ambito blockchain. Il tool è basato sul framework SuMo (SOlidity MUtator) e mira ad introdurre vulnerabilità note in smart contract scritti in solidity. MuSe sfrutta un motore di parsing AST per individuare pattern all'interno del codice e inserire specifiche mutazioni. Attualmente il tool è utilizzabile esclusivamente tramite interfaccia a **riga di comando (CLI)** in ambiente locale.

2 Reverse Engineering

Per la fase di **Reverse Engineering**, data la natura del progetto, è stato adottato un approccio di tipo **Black-box RE**, basato sull'osservazione del comportamento del tool tramite l'esecuzione con input differenti. Questa tecnica, che *non richiede la modifica del codice sorgente*, ha consentito di ricostruire la **struttura modulare** di **SuMo** e **MuSe**, evidenziando le dipendenze tra i componenti principali. Inoltre, ha permesso di identificare in modo preciso le parti del sistema con cui il plugin interagirà, fornendo una base solida per il processo di **integrazione** e **manutenzione** all'interno dell'IDE.

2.1 System and Object Design

Il tool **MuSe** rappresenta un'estensione del tool **SuMo**, progettata per introdurre una nuova categoria di mutanti *security-oriented*.

2.1.1 Moduli (Packages)

- **MutationGenerator:** gestisce abilitazione e disabilitazione dei mutanti;
- **Mutation:** si occupa dell'identificazione e applicazione delle mutazioni;
- **MutationRunner:** implementa le funzioni principali di *SuMo* usabili da riga di comando;
- **Operators:** implementa la logica di tutti i mutanti implementati in *SuMo* e *MuSe*;
- **TestingInterface:** gestisce la rete di blockchain locale e si occupa di instanziare il compilatore solidity durante la fase di testing;
- **Reporter:** si occupa di stampare i report durante le fasi di esecuzioni del tool;
- **Utils:** modulo contenente funzioni generiche utilizzate dagli altri moduli.

2.1.2 Componenti Off-The-Shelf

Per l'esecuzione dei test, MuSe si integra con framework consolidati per il testing di smart contract **Truffle, Hardhat, Brownie** e **Forge (Foundry)**.

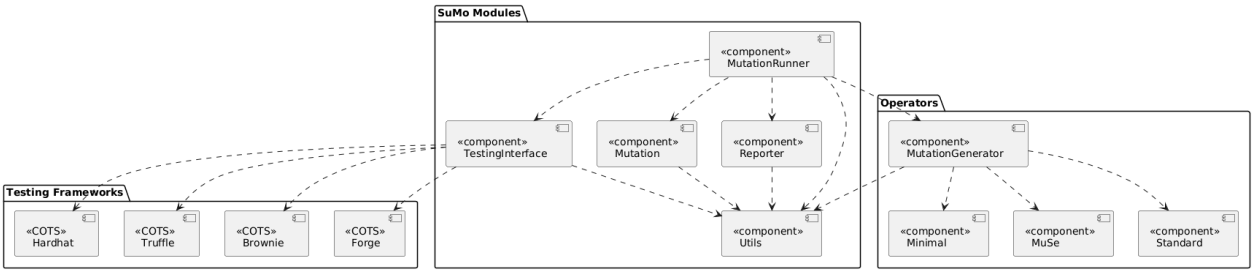

Figure 1: Component Diagram MuSe

2.2 Requirements Analysis

2.2.1 Requisiti Funzionali

Table 1: Requisiti funzionali del sistema MuSe

CODICE	NOME_REQUISITO	DESCRIZIONE
RF_1	Generazione contratti mutati	Il sistema deve permettere la creazione di versioni mutate dei contratti originali per finalità di testing.
RF_2	Esecuzione pre-test	Il sistema deve consentire l'esecuzione di test sul contratto originale fornito in input, per verificarne il corretto funzionamento.
RF_3	Esecuzione test mutanti	Il sistema deve permettere il testing dei contratti mutati generati, valutando la correttezza e il comportamento rispetto al contratto originale.
RF_4	Identificazione mutanti	Il sistema deve consentire di individuare i mutanti possibili applicabili al contratto in input, secondo le regole definite dagli operatori.
RF_5	Abilitazione operatori	Il sistema deve permettere di abilitare operatori di mutazione, rendendoli attivi per l'applicazione sui contratti.
RF_6	Disabilitazione operatori	Il sistema deve permettere di disabilitare operatori di mutazione, escludendoli dal processo di generazione dei mutanti.

Continua nella pagina successiva

Tabella continuata dalla pagina precedente

CODICE	NOME_REQUISITO	DESCRIZIONE
RF_7	Generazione report risultati	Il sistema deve consentire la creazione di report dettagliati contenenti i risultati dei test sui contratti originali e mutati.
RF_8	Stampa degli operatori abilitati	Il sistema deve permettere la stampa degli operatori abilitati dall'utente.

2.2.2 Requisiti Non Funzionali

Dall'analisi di **Reverse Engineering** svolta, e dall'esame del **codice sorgente**, non è stato possibile individuare un insieme definito di **requisiti non funzionali**. L'analisi ha comunque evidenziato che il **tool** risulta essere *complesso da utilizzare ed installare* per chi non possiede conoscenze pregresse sull'esecuzione di tool tramite **CLI (Node)**.

2.2.3 Use Cases

UC_01 Lookup Smart Contract	
Descrizione	Lo UC permette all'utente di individuare quali mutanti sono compatibili con lo smart contract inserito
Attore	Utente
Entry Condition	L'utente ha inserito uno *smart contract* nella cartella **contracts**
Exit Condition (Success)	Il sistema ha generato il report *mutations.json* con una lista dettagliata degli operatori
Exit Condition (Error)	Il sistema non genera il report *mutations.json*
FLUSSO DI EVENTI	
1 / Utente	L'utente esegue il comando `npx sumo lookup`
2 / Sistema	Il sistema genera il report `mutations.json` con la lista degli operatori compatibili

UC_02 Abilitazione Mutanti	
Descrizione	Lo UC permette all'utente di abilitare uno o più operatori da usare durante le mutazioni
Attore	Utente
Entry Condition	L'utente vuole abilitare uno o più operatori
Exit Condition (Success)	Il sistema abilita gli operatori e stampa a console una conferma
Exit Condition (Error)	Il sistema stampa un messaggio di errore
FLUSSO DI EVENTI	
1 / Utente	L'utente esegue il comando `npx sumo enable BOR`
2 / Sistema	Il sistema stampa un messaggio di conferma `BOR enabled`
FLUSSO DI EVENTI ALTERNATIVO (NO PARAMS)	
1.1.1 / Utente	L'utente non inserisce parametri
1.1.2 / Sistema	Il sistema stampa un messaggio di conferma `All mutation operators enabled.`
FLUSSO DI EVENTI ALTERNATIVO (ERROR)	
1.2.1 / Utente	L'utente inserisce un operatore non esistente `FDS`
1.2.2 / Sistema	Il sistema stampa un messaggio di errore `Error:FDS does not exist.`

UC_03 Mutazione Smart Contract	
Descrizione	Lo UC permette all'utente di generare contratti mutati
Attore	Utente
Entry Condition	L'utente ha inserito un contratto nella cartella *contracts* **AND** ha completato l'UC_02 (abilitazione mutanti)
Exit Condition (Success)	Il sistema genera i contratti mutati nella cartella */sumo/results/mutants*
FLUSSO DI EVENTI	
1 / Utente	L'utente esegue il comando `npx sumo mutate`
2 / Sistema	Il sistema genera eventuali mutanti risultanti e li salva nella cartella `/sumo/results/mutants` e stampa a console un messaggio di conferma `"Mutants saved to /sumo/results/mutants"`
FLUSSO DI EVENTI ALTERNATIVO (NO MUTANTS GENERATED)	
1.1.1 / Utente	L'utente ha abilitato operatori non compatibili
1.1.2 / Sistema	Il sistema non genera mutanti

UC_04 Pre-Test Smart Contract	
Descrizione	Lo UC permette all'utente di eseguire la test-suite solo sul contratto originale
Attore	Utente
Entry Condition	L'utente ha inserito un contratto nella cartella *contracts* **AND** ha inserito dei file di test all'interno della cartella *tests* **AND** ha modificato il valore di testingFramework all'interno di `sumo-config.js`
Exit Condition (Success)	Il sistema esegue con successo i test e genera un report `sumo-log.txt`
Exit Condition (Error)	Il sistema non esegue con successo i test
FLUSSO DI EVENTI	
1 / Utente	L'utente esegue il comando `npx sumo pretest`
2 / Sistema	Il sistema esegue i test e stampa un messaggio di conferma `Pre-test OK.`
FLUSSO DI EVENTI ALTERNATIVO (FRAMEWORK NON VALIDO)	
1.1.1 / Utente	L'utente ha inserito un framework non esistente
1.1.2 / Sistema	Il sistema stampa un messaggio di errore `Error: The specified testing framework is not valid.`
FLUSSO DI EVENTI ALTERNATIVO (ERRORE)	
1.2.1 / Sistema	Il sistema stampa un messaggio di errore `Error: Pre-test failed - original tests should pass.`

UC_05 Test Mutated Smart Contract	
Descrizione	Lo UC permette all'utente di eseguire la test-suite sui contratti mutati generati
Attore	Utente
Entry Condition	L'utente ha inserito un contratto nella cartella *contracts* **AND** ha inserito dei file di test all'interno della cartella *tests* **AND** ha modificato il testingFramework all'interno di `sumo-config.js` **AND** ha completato l'UC_02 (abilitazione mutanti)
Exit Condition (Success)	Il sistema esegue con successo i test e genera un report `sumo-log.txt`
Exit Condition (Error)	Il sistema non esegue con successo i test
FLUSSO DI EVENTI	
1 / Utente	L'utente esegue il comando `npx sumo test`
2 / Sistema	Il sistema genera il report `sumo-log.txt` e stampa un messaggio di conferma `Mutation Testing Completed...`
FLUSSO DI EVENTI ALTERNATIVO (FRAMEWORK NON VALIDO)	
1.1.1 / Utente	L'utente ha inserito un framework non esistente
1.1.2 / Sistema	Il sistema stampa un messaggio di errore `Error: The specified testing framework is not valid.`

3 Forward Engineering

Nel seguente capitolo verrà documentato il processo di *Forward Engineering* adottato per l'implementazione delle tre Change Request proposte per il tool **MuSe**.

3.1 Change Requests

ID	NOME	DESCRIZIONE	TIPOLOGIA	PRIORITÀ
CR_01	Wrapping MuSe tool	Integrazione diretta del tool, sotto-forma di plugin, con l'IDE di Smart Contract Remix tramite l'utilizzo di API	Adattiva	Alta
CR_02	Implementazione UI	Implementazione di una UI che permetta agli utenti di interagire direttamente con il tool tramite le API implementate.	Additiva	Alta
CR_03	Pipeline CI/CD	Implementazione di una pipeline tramite Github Actions per automatizzare il processo di deploy (testing, docker image e github pages)	Perfettiva	Media

3.2 Impact Analysis

L'analisi dell'impatto della manutenzione sulle componenti del tool **MuSe** ha evidenziato che le modifiche introdotte sono **non invasive** rispetto al sistema originale.

In particolare:

- Le funzionalità core di **MuSe** rimangono **inalterate**; le *API* e il *plugin* Remix si interfacciano con il tool senza modificarne la logica interna.
- L'implementazione del *plugin* e delle *API* introduce un **layer aggiuntivo** per la comunicazione e la gestione dell'interfaccia utente, garantendo la compatibilità con l'IDE **Remix**.
- Non sono state apportate modifiche ai dati esistenti né ai processi di generazione dei mutanti e dei report, preservando la **stabilità** del tool originale.

In sintesi, l'analisi conferma che l'evoluzione del sistema tramite le change request proposte **non comporta rischi significativi** per la piattaforma **MuSe** esistente, ma ne estende le capacità in modo **modulare** e **sicuro**.

3.3 Requirements Analysis

3.3.1 Requisiti Funzionali

Table 2: Requisiti funzionali del plugin MuSe per Remix

CODICE	NOME_REQUISITO	DESCRIZIONE
RF_9	Lettura file da Remix	Il sistema deve permettere la lettura di file presenti nell'IDE Remix.
RF_10	Scrittura file su Remix	Il sistema deve permettere la scrittura di file presenti nell'IDE Remix.
RF_11	Esecuzione comandi da Remix	Il plugin deve consentire l'esecuzione dei comandi principali di MuSe direttamente dall'ambiente Remix.
RF_12	Visualizzazione output	Il sistema deve visualizzare nel plugin i messaggi ricevuti dal backend, come output, errori o conferme.
RF_13	Selezione del contratto	Il plugin deve permettere all'utente di selezionare uno o più smart contract (presenti nella cartella *contacts*) da inviare al backend per l'elaborazione.
RF_14	Visualizzazione dei report	Il sistema deve generare un report (in formato HTML) dove vengono esposti in maniera chiara i risultati del MUtation Testing.

3.3.2 Requisiti Non Funzionali

Durante la fase di Reverse Engineering non è stato possibile estrarre i requisiti non funzionali del tool MuSe, ma durante la fase di manutenzione c'è stata la possibilità di introdurre nuovi requisiti non funzionali:

Table 3: Benefici qualitativi introdotti dalla manutenzione del plugin MuSe per Remix

ID	ASPETTO MIGLIORATO	DESCRIZIONE
RNF_1	Usabilità	L'uso del plugin in Remix IDE consente di interagire con MuSe tramite un'interfaccia grafica, eliminando la necessità di utilizzare la riga di comando.
RNF_2	Accessibilità multipiattaforma	L'integrazione con Docker e Remix consente di utilizzare MuSe anche su Windows e MacOS, superando la precedente limitazione ai sistemi Linux.
RNF_3	Facilità di installazione	L'uso di una Docker image e l'hosting mediante *GitHub Pages* ha semplificato il processo di installazione del tool, rendendolo immediato e ripetibile.
RNF_4	Automazione del flusso di lavoro	L'utente può ora eseguire tutte le funzionalità principali direttamente dall'IDE, senza necessità di comandi esterni o setup delle librerie esterne usate dal tool.
RNF_5	Riduzione degli errori utente	L'interfaccia guida l'utente attraverso le operazioni disponibili, riducendo il rischio di errori di sintassi o uso scorretto della CLI.
RNF_6	Estendibilità futura	La comunicazione tramite API REST tra plugin e backend consente di aggiungere nuove funzionalità in modo modulare e scalabile.

3.3.3 Use Cases

UC-M_01 Mutazione Smart Contract	
Descrizione	Lo UC permette all'utente di scegliere il contratto da mutare, selezionare gli operatori ed effettuare la mutazione
Attore	Utente
Entry Condition	Il Plugin è caricato in **Remix IDE** e l'utente ha inserito uno *smart contract* nella cartella **contracts** in **Remix IDE**
Exit Condition (Success)	Il sistema ha generato i contratti mutati nella cartella *MuSe/results/mutants*
Exit Condition (Error)	Il sistema non genera i contratti mutati ed avvisa dell'esito tramite un messaggio in console
FLUSSO DI EVENTI	
1 / Utente	L'utente sceglie un contratto dal menu Dropdown
2 / Utente	L'utente seleziona uno o più operatori dai Dropdown menu presenti
3 / Utente	L'utente clicca il tasto **Mutate** per avviare il processo di mutazione
4 / System	Il sistema genera i contratti mutati nella cartella *MuSe/results/mutants* in Remix IDE e avvisa in console con "File saved successfully"
FLUSSO DI EVENTI ALTERNATIVO (Nessun contratto selezionato)	
1.2.1 / Utente	L'utente non ha selezionato contratti da mutare
1.2.2 / Sistema	Il sistema stampa un messaggio in console: "Please select a contract first."
FLUSSO DI EVENTI ALTERNATIVO (Nessun mutante generato)	
2.1.1 / Utente	L'utente ha selezionato operatori che non generano mutanti
2.1.2 / Sistema	Il sistema stampa un messaggio in console: "No mutants generated ..."
FLUSSO DI EVENTI ALTERNATIVO (Nessun operatore selezionato)	
2.2.1 / Utente	L'utente non ha selezionato operatori
2.2.2 / Sistema	Il sistema stampa un messaggio in console: "Please select at least one mutation operator."

UC-M_02 Test Smart Contract	
Descrizione	Lo UC permette all'utente di eseguire in maniera rapida il mutation testing sul contratto
Attore	Utente
Entry Condition	L'utente ha completato l'UC-M_01 (Mutazione smart contract) **AND** ha inserito i test nella cartella **tests** in **Remix IDE**
Exit Condition (Success)	Il sistema esegue con successo i test e genera un report `report.html` nella cartella `MuSe/results`
Exit Condition (Error)	Il sistema non esegue con successo i test e avvisa con un messaggio in console
FLUSSO DI EVENTI	
1 / Utente	L'utente sceglie i parametri di testing: TestingFramework (Dropdown menù), TestingTimeout (Input Field)
2 / Sistema	Il sistema esegue i test, genera il report `report.html` e stampa un messaggio di conferma sulla console `Mutation Testing Completed...`
FLUSSO DI EVENTI ALTERNATIVO (TEST NON COMPATIBILI)	
1.1.1 / Utente	L'utente ha inserito file di testing non compatibili col framework scelto
1.1.2 / Sistema	Il sistema stampa un messaggio di errore `No test file found...`
FLUSSO DI EVENTI ALTERNATIVO (ERRORE)	
6.1.1 / Sistema	Il sistema non riesce ad eseguire con successo i test
6.1.2 / Sistema	Il sistema stampa un messaggio di errore `Error: Pre-test failed - original tests should pass.`

3.3.4 Sequence Diagram

In questa sezione sono riportati due Sequence Diagram che illustrano, a livello alto, le interazioni tra i diversi attori del plugin durante l'esecuzione delle due operazioni principali: Mutation e Testing.

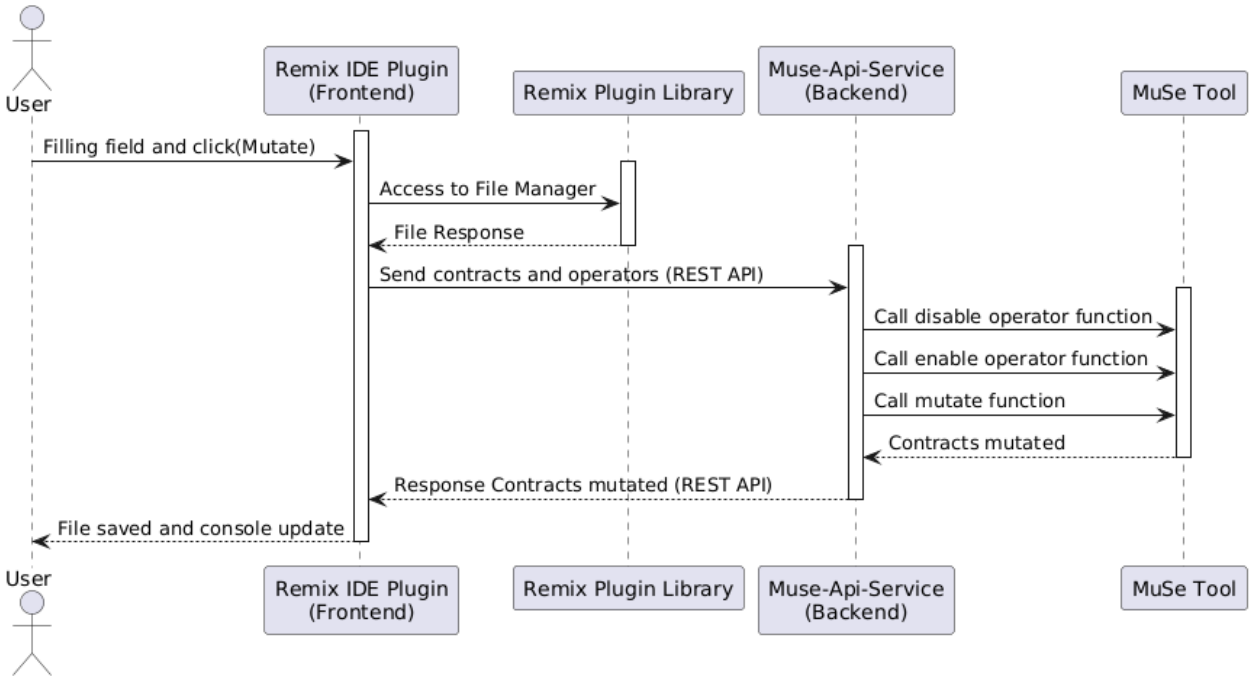

Figure 2: Sequence Diagram Mutation (SD_01)

La precondizione indicata nel Sequence Diagram SD_02 (Figura 3) specifica che, per poter eseguire la fase di testing, è necessario completare preventivamente il Sequence Diagram SD_01 (Figura 2), al fine di generare i mutanti da utilizzare durante il testing.

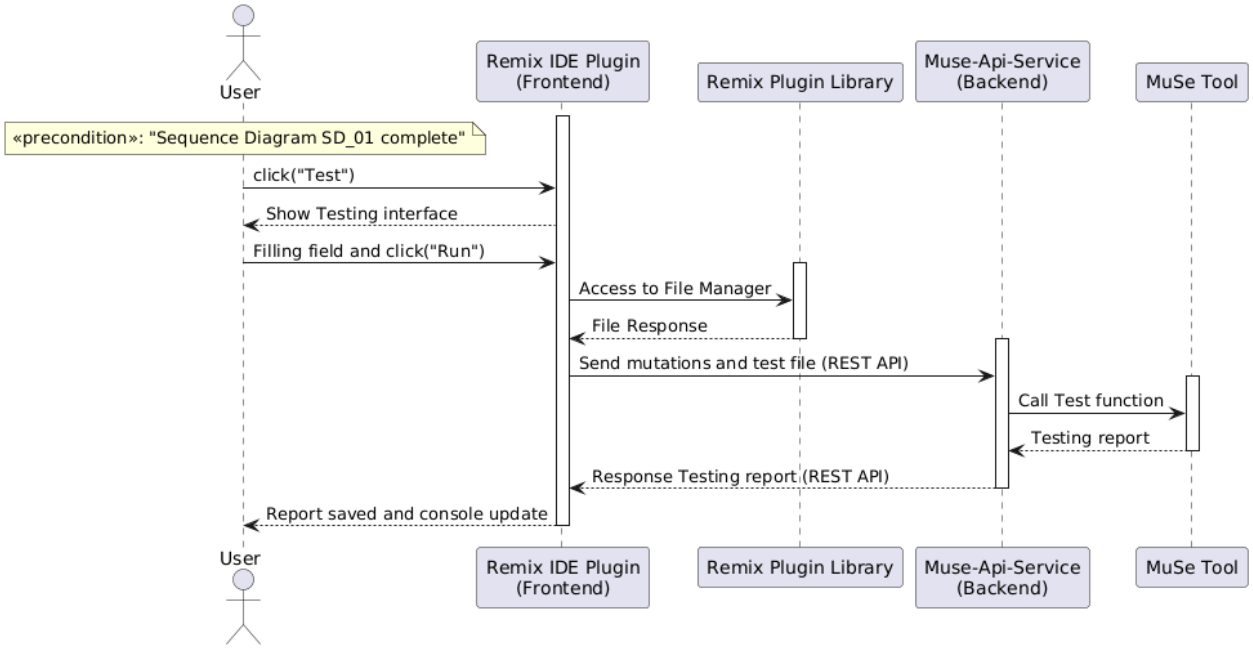

Figure 3: Sequence Diagram Testing (SD_02)

3.4 System and Object Design

3.4.1 Sottosistemi

Nella versione precedente del tool erano previsti principalmente due sottosistemi:

- **SuMo:** contenente l'implementazione dei metodi principali usati da MuSe.
- **Operators:** contenente la logica dei singoli operatori implementati

Nella versione attuale, sono stati previsti tre sottosistemi principali (Figura 4)

- **MuSe:** contenente tutta la logica di SuMo e Operators (Figura 1)
- **MuSe API Service:** sottosistema che espone delle REST API per poter comunicare con il tool MuSe.
- **MuSe Remix Plugin:** sottosistema che gestisce il front-end e la connessione tra IDE ed API.

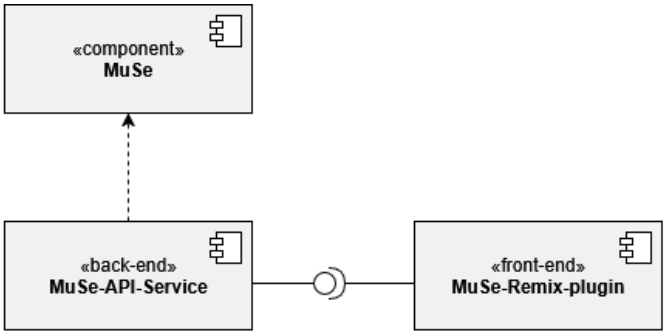

Figure 4: Component Diagram MuSe Remix Plugin

3.4.2 Architettura

L'architettura precedente prevedeva principalmente un singolo nodo, la macchina locale che eseguiva MuSe. Nell'attuale sistema si prevede un'architettura composta da tre nodi principali (Figura 5)

- **Remix IDE:** L'ambiente di sviluppo integrato per il quale è stato sviluppato il plugin, che permette agli utenti di interagire con MuSe direttamente dall'IDE.
- **Docker Container:** Contenitore locale che ospita le API, consentendo la comunicazione tra il plugin Remix e il modulo MuSe.
- **GitHub Pages:** Hosting del front-end sviluppato in React, che funge da interfaccia tra Remix IDE e le API.

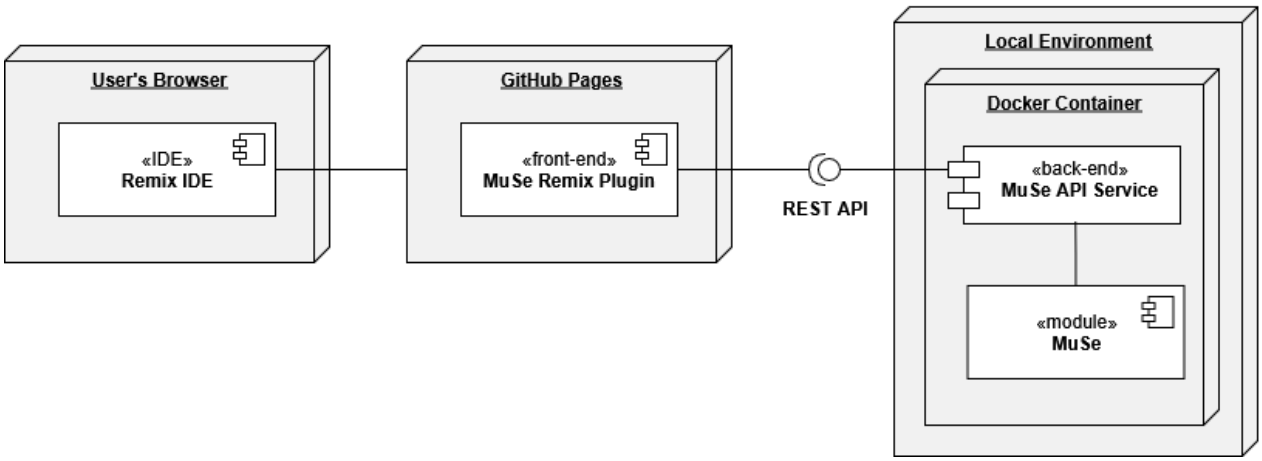

Figure 5: Deployment Diagram MuSe Remix Plugin

3.5 Testing

Le funzionalità implementate, unitamente a quelle già presenti nel tool originale MuSe (che originariamente non includeva una test-suite), sono state oggetto di attività di testing e saranno documentate nei relativi documenti di test dedicati.

3.6 Installazione del Tool

Per informazioni dettagliate sull'installazione del tool e sulla sua prima esecuzione, si rimanda al file `README` presente all'interno della repository GitHub[1].

[1] https://github.com/Carmineh/MuSe-Remix-Plugin/