

CODE SMILE PLUGIN



Indice

PROBLEM DOMAIN	3
SCENARIOS	3
Analisi sull'intero progetto	3
Analisi sul file corrente	4
Analisi su più file scelti dall'utente	4
Analisi real-time	5
FUNCTIONAL REQUIREMENTS	5
NONFUNCTIONAL REQUIREMENTS	5
TARGET ENVIRONMENT	6
Compatibilità e Piattaforme Supportate	7
Architettura e Comunicazione	7
Use Cases	7
UC1 Analisi del progetto	7
UC2 Analisi sul file corrente	8
UC3 Analisi su determinati file	9
UC4 Analisi real-time	10
SUBSYSTEMS DECOMPOSITION	12
OBJECT DESIGN & CLASS DIAGRAM	15
DESIGN PATTERNS	18

PROBLEM DOMAIN

I code smells rappresentano pratiche di implementazioni subottimali in progetti software che possono compromettere le prestazioni, l'affidabilità e la manutenibilità del sistema. Non si tratta di veri e propri bug, ma di pratiche scorrette che possono portare alla nascita degli stessi e compromettere la qualità del progetto.

Con la nascita dell'intelligenza artificiale e la sua integrazione nei progetti software standard (i.e. progetti che fanno uso di Machine Learning, Deep Learning) si sono venuti a creare i cosiddetti ML-CSs che sta per "Machine Learning specific code smells", ovvero code smells specifici a Machine Learning pipeline.

Se i code smells tradizionali sono legati a determinati anti-patterns riguardante il codice, con l'introduzione dell'IA in questi progetti che hanno i dati come componente core, i ML-CSs si riferiscono a problemi specifici legati a dati, modelli e codice di addestramento.

L'AI è un campo in rapida evoluzione e lo studio dei code smells in ML-enabled systems è ancora poco esplorato, ma alcuni ricercatori hanno iniziato a investigare il problema, tra cui (i) Zhang, il quale ha lavorato alla stesura di una lista di 22 code smells, e (ii) Recupito & Giordano che hanno lavorato su un tool di analisi statistica che rileva 16 su 22 dei code smells trovati in letteratura, ovvero Code Smile che è la base dello sviluppo di questo studio.

Code Smile è un tool di analisi statistica per progetti Python e identificazione di ML-CSs.

Lo scopo di questo studio è (i) rendere Code Smile un plugin per ambienti di sviluppo che supportano il linguaggio Python e (ii) di evolvere la natura statica dell'analisi del tool a quella real-time.

SCENARIOS

Per rendere chiaro cosa intende fare il plugin e il suo funzionamento, vengono proposti una serie di scenari:

Analisi sull'intero progetto

Marco, uno sviluppatore Python, sta lavorando ad un progetto di Machine Learning insieme al suo team di lavoro.

Marco all'interno del suo team si occupa delle seguenti task: pulire il dataset, restituire il dataset, validarlo prima di consegnarlo al team che si occuperà della creazione del modello.

Marco ha già una parte di codice scritta da colleghi che hanno lavorato prima di lui, lo fa partire e tutto sembra funzionare correttamente, quindi inizia a lavorarci.

Dopo giorni di lavoro e di testing sulle sue task, Marco vuole condividerle con i suoi team members, ma Marco, essendo una persona scrupolosa e perfezionista, vuole fare ottimizzazione sul suo lavoro e cerca se ci sono strumenti che lo aiutano in tal senso a dargli degli insights sul suo lavoro.

Si imbatte nel plugin CodeSmile, un plugin per la detection di code smells in progetti Python.

Marco è estasiato, lo installa ed è pronto ad usarlo.

Appena apre il plugin seleziona la voce "Analisi su tutto il progetto", il plugin fa partire l'analisi e gli segnala 4 code smells: 2 DCA (Dataframe Conversion API Misused), 1 NaN (NaN Equivalence Comparison Misused), 1 UI (Unnecessary Iteration) con tanto di indicazione in che file e che linea si trovano questi code smells e un messaggio sulla possibile causa e/o consigli su come fixarlo.

Marco prende coscienza dei messaggi del plugin, ricontrolla le linee di codice dei file segnalati e opera in modo tale per rimuoverli.

Analisi sul file corrente

Dopo una prima correzione, Marco fa ripartire il plugin e vede che è rimasto solo il code smells UI nel file xxx.py.

Di conseguenza, Marco conscio che il plugin fa selezionare in che modalità far partire l'analisi ed essendo i code smells vincolati ad un solo file, decide di concentrarsi e di far partire l'analisi da ora fino a che non rimuove il code smell solo sul file corrente, come permesso dal plugin.

Analisi su più file scelti dall'utente

Marco, soddisfatto dall'esperienza per ora avuta usando il plugin, decide di rispolverare un suo vecchio progetto e di applicare l'analisi solo su determinati files su cui ha l'impressione, ma non è perfettamente sicuro di aver introdotto alcuni code smells.

Anche in questo il plugin si rileva uno strumento di grande aiuto che lo aiuta a trovare ben 17 code smells in quei file.

Marco è sempre più soddisfatto e dell'aiuto che lo strumento che gli sta dando, ma trova scoccante dover cliccare sempre il tasto e cambiare la modalità

Analisi real-time





Pertanto, Marco decide esclusivamente per questo progetto di attivare la modalità real-time che gli segnala i code smells mentre lui modifica il codice a proprio piacimento.

Questo lo aiuta perché, essendo molti code smells e dovendo aggiungere nuove funzionalità e nuovi files, riesce a tenerli sott'occhio tutti contemporaneamente e a prevenire la creazione di una versione di file con code smells già integrati.

Questi scenari dimostrano come il plugin CodeSmile può essere utile in diverse situazioni di sviluppo, migliorando la qualità del codice in progetti di Machine Learning.




FUNCTIONAL REQUIREMENTS

I requisiti funzionali per lo sviluppo di questo plugin individuati sono i seguenti:

- **FR1:** Il plugin deve fornire analisi di progetti Python e detection di code smells, usando il tool già esistente CodeSmile
Priority 
- **FR2:** Il plugin deve fornire un'interfaccia per scegliere il tipo di analisi
Priority 
- **FR3:** Il plugin deve supportare real-time detection
Priority 
- **FR4:** Deve fornire feedback visivo direttamente nell'editor.
Priority 

NONFUNCTIONAL REQUIREMENTS

I requisiti non funzionali per lo sviluppo di questo plugin individuati sono i seguenti:

- **NFR1:** Il plugin deve essere performante, senza rallentare PyCharm.
Priority 
- **NFR2:** Deve **essere scalabile**, per future integrazioni con AI.
Priority 
- **NFR3:** Deve funzionare su **Windows, Linux e macOS**.
Priority 

Legenda priorità

Alta 

Media 

Bassa 

TARGET ENVIRONMENT

Il plugin **CodeSmile** verrà sviluppato per l'ambiente di sviluppo **JetBrains PyCharm**, che è ampiamente utilizzato nella comunità Python, in particolare da sviluppatori e ricercatori nel campo del **Machine Learning e Data Science**.

Per la realizzazione del plugin, verranno utilizzate le seguenti tecnologie e strumenti:

1. JetBrains IntelliJ Platform Plugin SDK (DevKit)

- Il plugin sarà costruito utilizzando il framework **IntelliJ Platform**, che permette di estendere le funzionalità degli IDE JetBrains (come PyCharm).
- **Motivazione:** Questa scelta garantisce una **perfetta integrazione con PyCharm**, permettendo di sfruttare API native per l'analisi del codice, la gestione dei file e l'interfaccia utente.

2. Kotlin per l'implementazione principale del plugin

- Kotlin è il linguaggio ufficiale per lo sviluppo di plugin IntelliJ ed è altamente interoperabile con Java.
- **Motivazione:** È più conciso e sicuro rispetto a Java, riducendo il rischio di errori e migliorando la leggibilità del codice.

3. Gradle per la gestione delle dipendenze e della build

- Gradle è lo strumento di build utilizzato per i plugin IntelliJ.
- **Motivazione:** Facilita l'integrazione delle librerie necessarie e semplifica il processo di distribuzione del plugin.

4. Python come linguaggio di riferimento per CodeSmile

- CodeSmile è uno strumento scritto in **Python**, e il plugin dovrà essere in grado di comunicare con esso.
- **Motivazione:** Poiché l'analisi dei code smells si basa su CodeSmile, il plugin dovrà invocare lo script Python in background o interagire con esso tramite **chiamate di sistema o un'API locale**.

5. File di configurazione XML/YAML

- I plugin IntelliJ utilizzano file **XML** per definire la configurazione del plugin (registrazione delle azioni, menu, dipendenze).
- YAML potrebbe essere utilizzato per memorizzare preferenze o configurazioni degli utenti.

Compatibilità e Piattaforme Supportate

PyCharm Community & Professional Edition (versione minima 2021.3, o superiore per garantire compatibilità con le API più recenti).

Sistemi Operativi supportati:

- **Windows**
- **Linux**
- **macOS**

Architettura e Comunicazione

Il plugin sarà progettato con un'architettura modulare per facilitare futuri aggiornamenti e integrazioni. Le principali modalità di comunicazione tra il plugin e CodeSmile saranno:

1. **Chiamate dirette a CodeSmile (CLI)** → Il plugin eseguirà CodeSmile come processo separato e interpreterà l'output JSON.
2. **API locale** → CodeSmile verrà eseguito in modalità **server locale**, consentendo al plugin di inviare richieste HTTP per ottenere i risultati dell'analisi.

Questa architettura garantirà **scalabilità**, permettendo in futuro di integrare **un motore AI per il rilevamento avanzato di code smells**.

Use Cases

UC1 Analisi del progetto

UC1	Analisi del progetto
-----	----------------------

Attori	Utente
Entry condition	L'utente si trova in Pycharm, ha un progetto aperto e si trova nella view del plugin
Flusso degli eventi	<p>1. L'utente ha cliccato la voce "Analisi del progetto intero"</p> <p>2. Il plugin fa partire l'analisi e produce i risultati</p>
Exit condition	Il plugin ha rilevato e segnalato all'utente i code smells trovati

Flusso alternativo/Eccezioni: Al punto 2, se il plugin non trova nessuno smell nell'analisi mostra la scritta "Nessuno smell trovato"

UC2 Analisi sul file corrente

UC2	Analisi sul file corrente
Attori	Utente
Entry condition	L'utente si trova in Pycharm, ha un progetto aperto, ha almeno un file aperto nella tab dell'IDE e si trova nella view del plugin
Flusso degli eventi	<p>1. L'utente seleziona la voce "Analizza file corrente"</p> <p>2. Il plugin analizza il file e produce i risultati dell'analisi e li mostra all'utente</p>
Exit condition	Il plugin ha rilevato e segnalato all'utente i code smells

trovati

Flussi alternativi/Eccezioni:

- Se al punto 2 il plugin rileva che il file non è un file con estensione .py, viene mostrato un messaggio d'errore

UC3 Analisi su determinati file

UC3	Analisi su determinati file
Attori	Utente
Entry condition	L'utente si trova in Pycharm, ha un progetto aperto e si trova nella view del plugin
Flusso degli eventi	<div>1. L'utente clicca la voce "Analisi file multipla"</div> <div>2. Il plugin mostra all'utente una lista e chiede di spuntare quali file analizzare.</div> <div>3. L'utente sceglie i file che vuole analizzare</div> <div>4. Il plugin chiede conferma all'utente, chiedendogli di cliccare il bottone con scritto "Avvia Analisi".</div> <div>5. L'utente conferma l'opzione.</div> <div>6. Il sistema analizza i file e produce i risultati.</div>
Exit condition	Il plugin ha rilevato e segnalato all'utente i code smells trovati rispetto ai file indicati

Flussi alternativi/Eccezioni:

- Al punto 6 se i file non hanno estensione .py non vengono analizzati, mentre gli altri sì.

UC4 Analisi real-time

UC4	Analisi real-time
Attori	Utente
Entry condition	L'utente si trova in Pycharm, ha un progetto aperto e si trova nella view del plugin
Flusso degli eventi	<p>1. L'utente clicca la voce "Analisi real-time"</p> <p>2. Il plugin avvisa l'utente che da ora comincia la sessione di real time code smells detection</p> <p>3. L'utente crea, scrive, modifica file</p> <p>4. Il plugin effettua analisi real-time e nella sua view mostra i risultati della detection</p>
Exit condition	Il plugin rileva e segnala all'utente i code smells trovati in real-time

Flussi alternativi/Eccezioni:

- Se l'utente decide di cliccare il pulsante "Stop", chiude il plugin o esce da Pycharm, la detection viene stoppata e la sessione cancellata **[UC5 Stop Analisi Real-Time]**

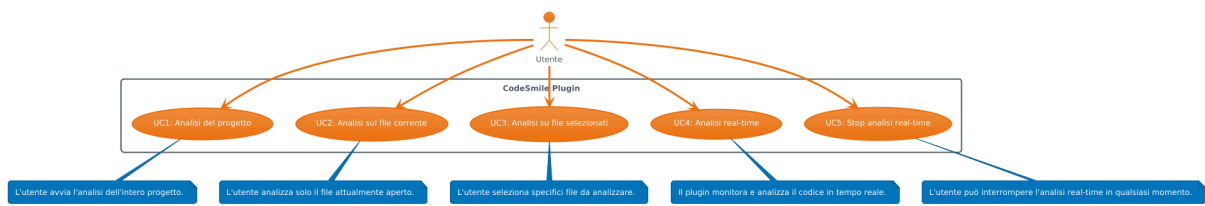
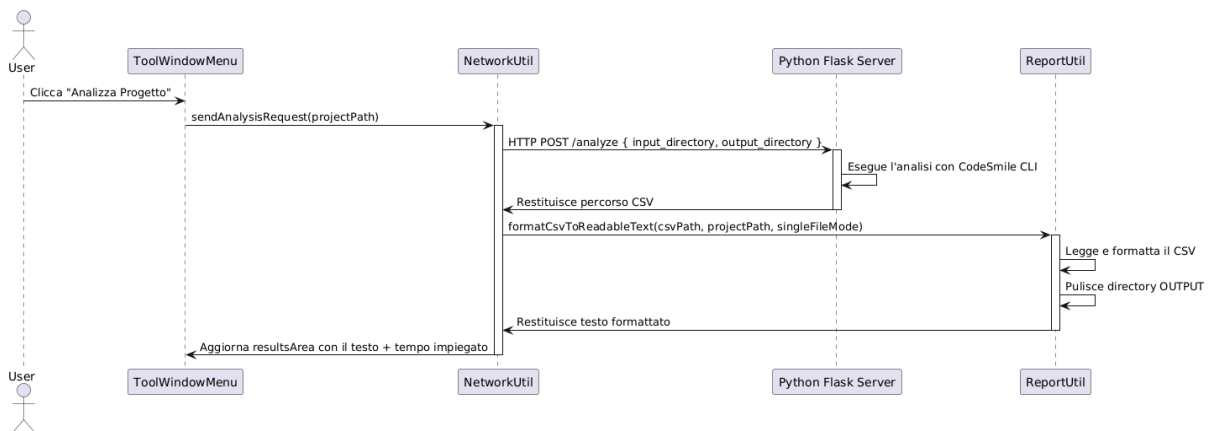


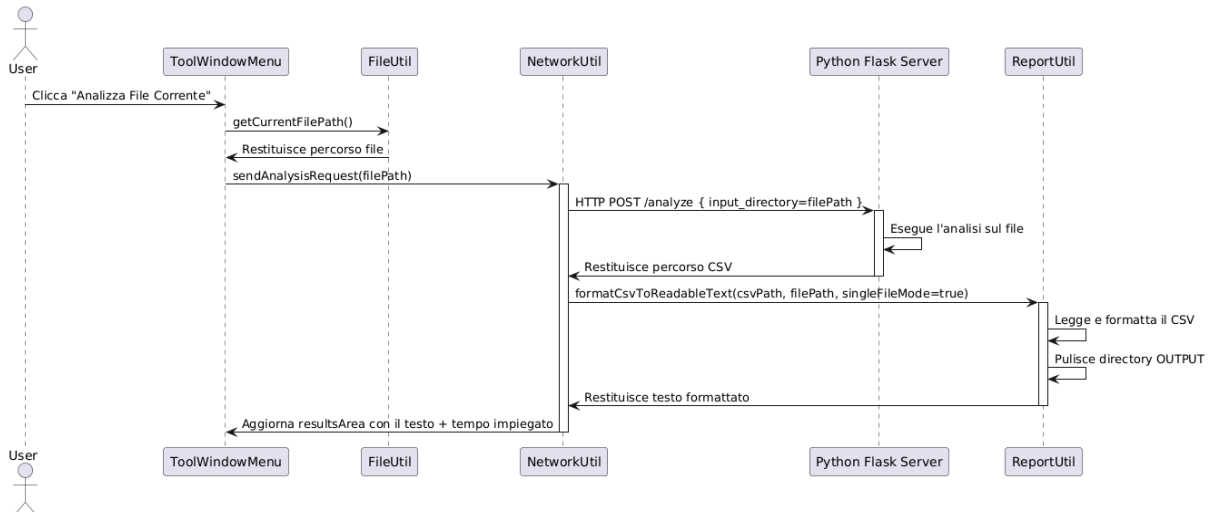
Fig. 1: UML Use Case Diagram

SEQUENCE DIAGRAMS

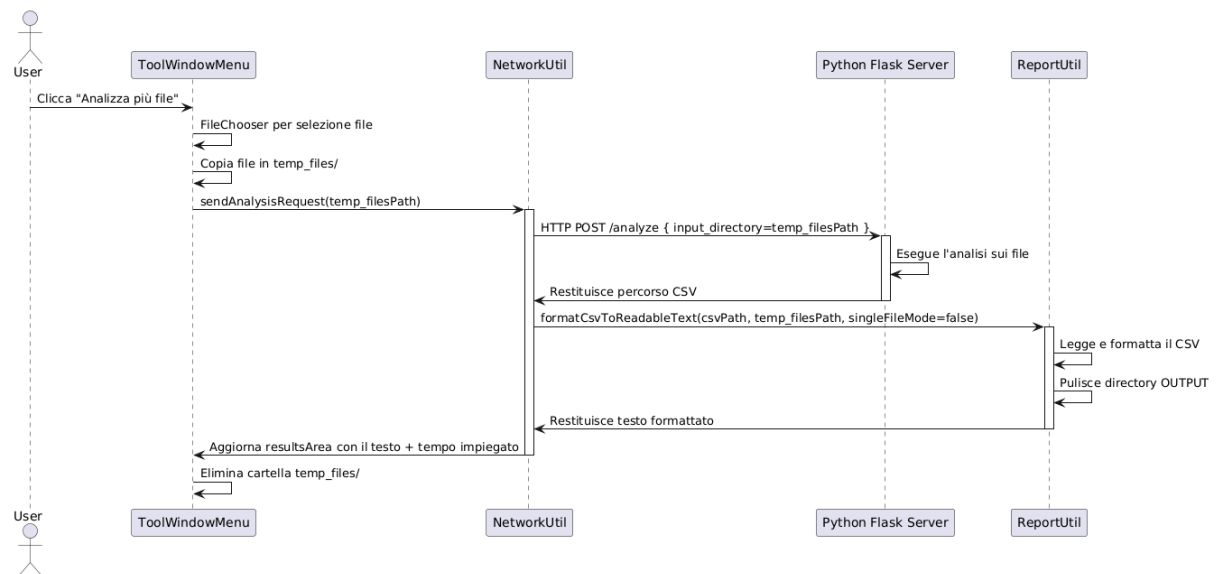
UC1



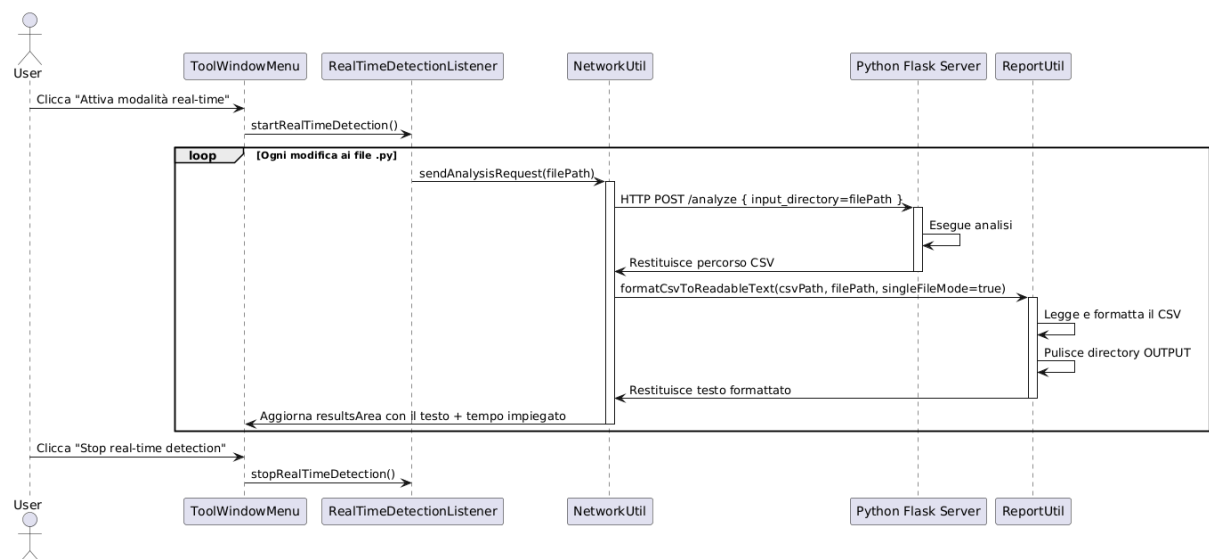
UC2



UC3



UC4



SUBSYSTEMS DECOMPOSITION

Il plugin **CodeSmile** è strutturato in più sottosistemi collaborativi, ognuno dei quali gestisce una parte distinta del ciclo di vita dell'analisi dei code smells nei progetti Python. Questa architettura consente di mantenere separate le responsabilità, migliorando la manutenzione e l'estendibilità del sistema.

Interfaccia Utente e Gestione dell'Interazione

Il primo sottosistema è rappresentato dalla **Tool Window Menu**, l'interfaccia grafica che espone le funzionalità del plugin all'utente. Qui vengono presentati all'utente diversi pulsanti che permettono di eseguire analisi su:

- L'intero progetto aperto.

- Il file Python attualmente selezionato nell'editor.
- Un set multiplo di file selezionati manualmente.

In aggiunta, la stessa interfaccia permette di attivare e disattivare la **modalità di detection real-time**, che monitora le modifiche sui file Python e avvia automaticamente nuove analisi senza richiedere ulteriori interazioni da parte dell'utente.

La Tool Window ospita anche un'area testuale che funge da terminale di output, nella quale vengono mostrati i risultati formattati delle analisi effettuate.

Gestione del Server Python

Un ulteriore componente critico è quello della **gestione del server Python** che esegue CodeSmile. Questa responsabilità è gestita direttamente nel plugin, che si occupa di:

- Terminare eventuali istanze già in esecuzione del server.
- Avviare una nuova sessione del server Flask sulla porta 5000, assicurandosi che sia pronto a ricevere richieste di analisi.

Questa logica viene integrata all'avvio della Tool Window per garantire che il backend Python sia sempre operativo e accessibile.

Comunicazione di Rete e Invio delle Richieste di Analisi

Il **Network Communication Subsystem** si occupa di inviare le richieste di analisi al server Flask, includendo nel payload il percorso del progetto o dei file da analizzare e la directory di output nella quale il server salverà i risultati.

Questo sottosistema è anche responsabile della ricezione della risposta dal server e del calcolo del **tempo effettivo di detection**, che include il tempo di invio della richiesta, elaborazione sul server e ricezione della risposta.

La gestione della comunicazione di rete è completamente astratta rispetto alla UI, permettendo di isolare il canale di comunicazione HTTP dai dettagli dell'interfaccia utente e delle altre funzionalità.

Formattazione dei Risultati

Una volta ricevuto il percorso al file di output generato dal server, il plugin passa i dati al **Result Formatting Subsystem**. Questo modulo si occupa di:

- Caricare e leggere il file CSV generato dal server.
- Estrarre i dati rilevanti come il nome del file, la funzione, il tipo di code smell e la descrizione.
- Formattare i dati in un testo leggibile e ordinato da mostrare nell'area dei risultati della Tool Window.

Il modulo include anche la gestione della pulizia della directory di output per evitare l'accumulo di file temporanei.

Monitoraggio in Tempo Reale delle Modifiche

La funzionalità di **Real-Time Detection** viene gestita da un listener che monitora le modifiche ai file Python nel progetto. Quando viene rilevata una modifica:

- Viene immediatamente inviata una nuova richiesta di analisi al server.
- I risultati vengono aggiornati in tempo reale nell'area di output della Tool Window.

Questa modalità garantisce che lo sviluppatore riceva feedback continuo mentre scrive codice, senza dover avviare manualmente l'analisi ogni volta.

Estensioni Future

Il design prevede una possibile estensione future:

- **Motore AI per Analisi Avanzata:** per introdurre una strategia di analisi basata su tecniche di intelligenza artificiale, in grado di migliorare l'efficacia e la precisione nel rilevamento dei code smells.

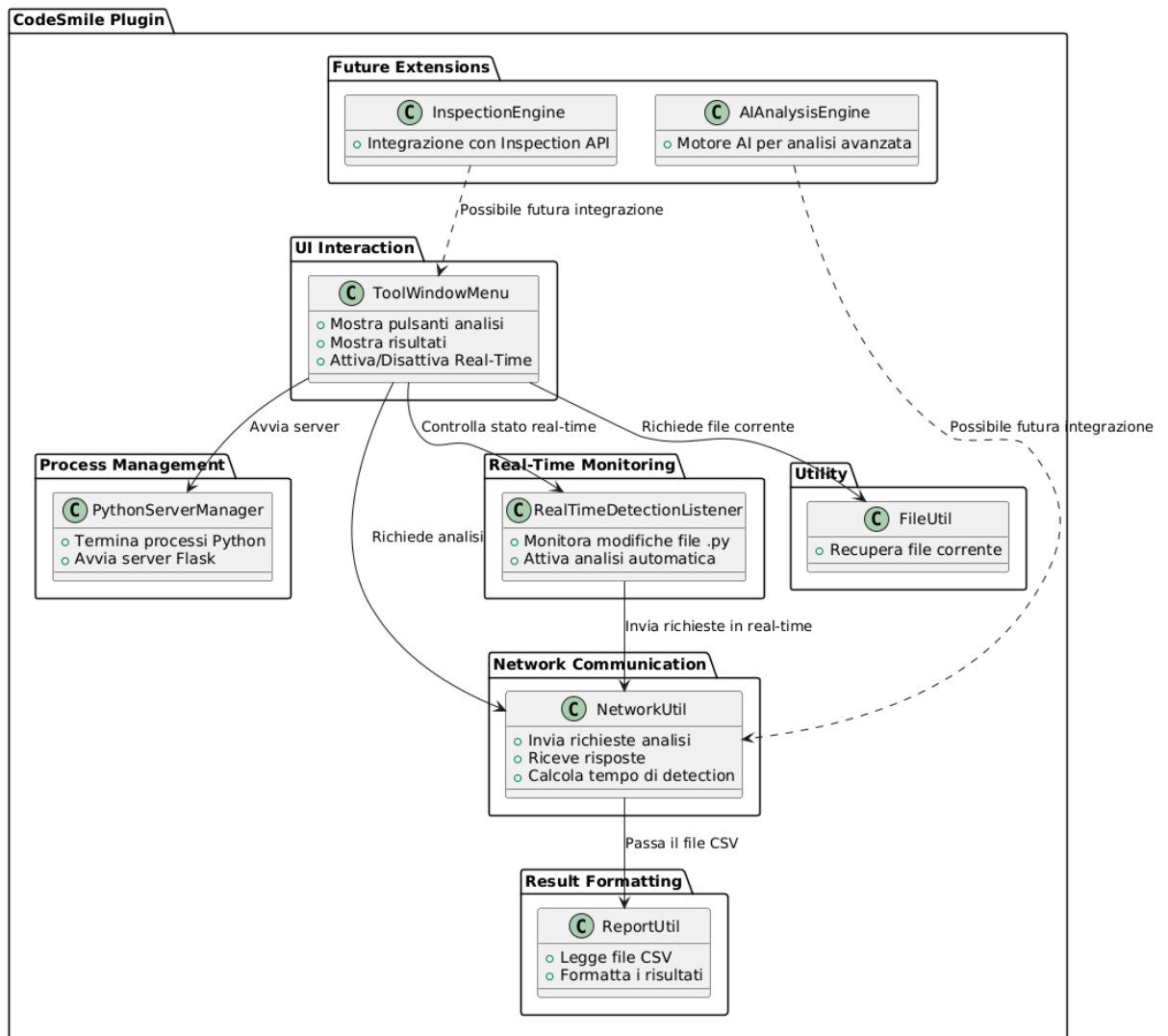


Fig 2: UML Component Diagram of the Code Smile Plugin subsystems decomposition

OBJECT DESIGN & CLASS DIAGRAM

L'architettura del plugin **CodeSmile** è suddivisa in classi che rappresentano in modo chiaro le responsabilità identificate nei vari sottosistemi. Ogni classe svolge un ruolo specifico nel ciclo di vita dell'analisi, dalla gestione dell'interfaccia utente alla comunicazione di rete, dalla formattazione dei risultati fino al monitoraggio real-time dei file.

ToolWindowMenu

Questa classe rappresenta il **cuore dell'interfaccia utente del plugin**. Espone:

- I pulsanti per avviare le analisi su progetto, file singolo o selezione multipla.
- I controlli per attivare e disattivare la modalità di detection in tempo reale.
- L'area testuale per la visualizzazione dei risultati.

Inoltre, contiene metodi per:

- Terminare eventuali processi Python già attivi.
- Avviare una nuova istanza del server Flask.

Questa classe funge da orchestratore principale del flusso di interazione tra utente e sistema.

NetworkUtil

Responsabile della **comunicazione con il server Python**, questa classe si occupa di:

- Preparare le richieste di analisi, includendo percorso di input e output.
- Inviare le richieste HTTP al server.
- Gestire la ricezione e il parsing della risposta JSON.
- Calcolare e riportare il **tempo effettivo di detection**.

I risultati vengono poi passati a ReportUtil per la formattazione finale.

ReportUtil

Questa classe riceve il percorso al file CSV generato dal server e si occupa di:

- Leggere e interpretare le righe del CSV.
- Formattare i risultati in un testo chiaro e leggibile, evidenziando file, funzione, tipo di code smell e descrizione.
- Pulire la directory di output dopo aver processato i risultati, per evitare accumulo di file temporanei.

NOTA: Non calcola il tempo di detection, concentrandosi esclusivamente sulla presentazione dei dati.

FileUtil

Questa classe fornisce una semplice utilità per:

- Recuperare il percorso del **file attualmente aperto** nell'editor di PyCharm, utilizzato nei flussi di analisi su file singolo.

RealTimeDetectionListener

Implementa la gestione della **modalità di detection real-time**, monitorando le modifiche ai file .py nel progetto. Le sue funzioni includono:

- Rilevare modifiche ai file.
- Innescare l'analisi automatica tramite NetworkUtil.
- Gestire l'attivazione e disattivazione della modalità real-time.

Collaborazioni tra le classi

- **ToolWindowMenu** collabora con **NetworkUtil** per avviare le analisi e con **FileUtil** per recuperare il file corrente.
- **NetworkUtil** si affida a **ReportUtil** per la formattazione dei risultati ricevuti dal server.
- **RealTimeDetectionListener** richiama **NetworkUtil** per avviare analisi automatiche in risposta alle modifiche dei file.

Questa suddivisione garantisce una separazione netta tra:

- **Interfaccia utente e gestione dei processi** (ToolWindowMenu).
- **Comunicazione di rete e gestione delle richieste** (NetworkUtil).
- **Formattazione dei risultati** (ReportUtil).
- **Supporto alla selezione del file corrente** (FileUtil).
- **Monitoraggio real-time dei file** (RealTimeDetectionListener).

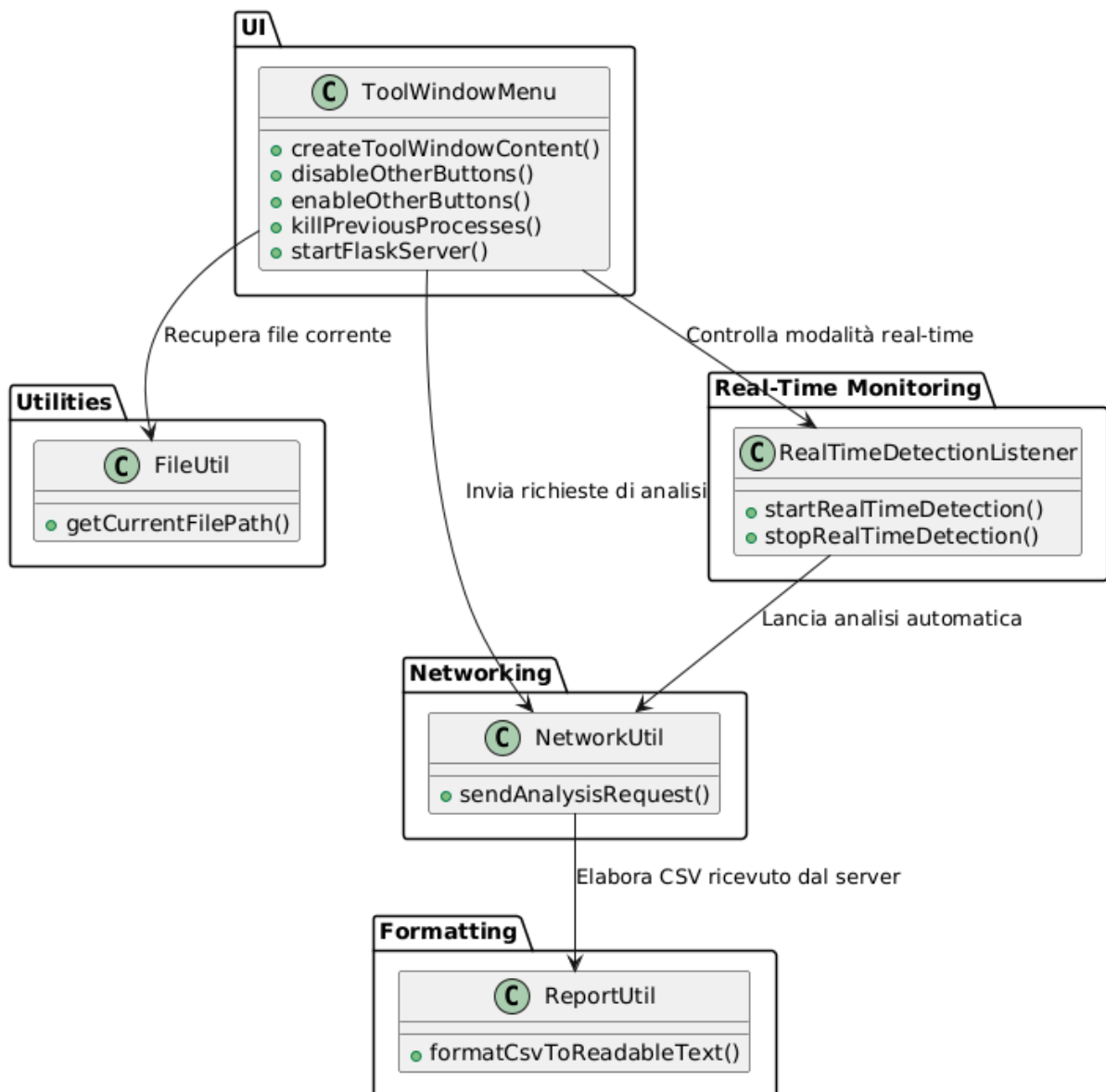


Fig 3: UML Class Diagram

DESIGN PATTERNS

Observer Pattern

Il **RealTimeDetectionListener** implementa un **Observer Pattern** per monitorare le modifiche ai file Python del progetto. Ogni variazione rilevata viene notificata al sistema, che risponde con l'avvio automatico di una nuova analisi. Questo

meccanismo permette di mantenere il plugin sempre allineato alle modifiche del codice senza intervento manuale.

Facade Pattern

Il modulo **NetworkUtil** funge da **facciata** semplificata verso la complessità della comunicazione HTTP, nascondendo la gestione del client HTTP, del payload e della risposta. La UI e i listener si interfacciano con questo modulo senza preoccuparsi dei dettagli di basso livello.

Builder / Formatter Pattern

Il modulo **ReportUtil** applica un **pattern di formattazione**, isolando la costruzione del testo leggibile dai dati strutturati forniti dal CSV. Questo approccio separa la presentazione dei dati dalla loro acquisizione o calcolo.