Assignment 2

Giangiulli Chiara (1189567; chiara.giangiulli@studio.unibo.it)
Shtini Dilaver (1189997; dilaver.shtini@studio.unibo.it)
Terenzi Mirco (1193420; mirco.terenzi@studio.unibo.it)

12 Maggio 2025

Analisi del problema

Il progetto ha come obiettivo l'analisi delle dipendenze tra classi, interfacce e package in un'applicazione Java, ovvero identificare da quali elementi del codice dipendono le porzioni analizzate.

Lo sviluppo prevede due soluzioni distinte, ciascuna basata su un approccio differente:

- **Asincrono**: è richiesta la realizzazione di una libreria in grado di operare a tre diverse profondità: sull'intero progetto, su un singolo package o su una specifica classe, con l'obiettivo di identificare le relative dipendenze. Le operazioni su ciascuno di questi livelli devono essere eseguite in modo indipendente e non bloccante.
- **Reattivo**: è previsto lo sviluppo di un programma GUI-based che permetta all'utente di avviare l'analisi delle dipendenze e visualizzare in modo dinamico e incrementale i risultati ottenuti per il progetto selezionato. Le relazioni devono essere mostrate sotto forma di grafo, eventualmente raggruppando le classi nei rispettivi package. I componenti devono reagire automaticamente a eventi esterni o modifiche di stato. In questo modello, i risultati si propagano come un flusso di valori nel tempo, anziché in un unico momento, permettendo all'utente finale di visualizzare l'avanzamento in tempo reale. L'interfaccia deve includere due pulsanti (uno per selezionare la *root folder* del progetto e uno per avviare l'analisi), una sezione per la visualizzazione del grafo e due contatori (uno per il numero di classi analizzate e uno per il numero di dipendenze individuate).

L'analisi delle dipendenze è un'operazione potenzialmente costosa in termini computazionali, soprattutto su repository di grandi dimensioni. La difficoltà non riguarda solo l'analisi dei singoli file, ma anche la combinazione dei risultati in modo coerente. Inoltre è necessario gestire correttamente gli errori, come file non trovati o errori di parsing, e garantire una visualizzazione dei risultati chiara e informativa per l'utente.

Design e Architettura

Versione asincrona

Il progetto è basato sull'utilizzo del framework Vert.x, che fornisce un'infrastruttura asincrona basata su un event loop interno. Questo permette di eseguire operazioni (come lettura file e parsing) in modo non-bloccante, senza dover gestire direttamente thread o concorrenza. La libreria `DependencyAnalyserLib` espone tre metodi principali:

- `getClassDependencies(File classSrcFile): Future<ClassDepsReport>` fornisce un report contenente tutte le dipendenze della classe specificata tramite il path, passato come parametro;
- `getPackageDependencies(File packageSrcFolder): Future<PackageDepsReport>` analizza le dipendenze del package in ingresso, in particolare processando ogni classe a esso appartenente;

- `getProjectDependencies(File projectSrcFolder): Future<ProjectDepsReport>` restituisce una lista di dipendenze per ogni classe all'interno di un determinato progetto.

Tutti i metodi ritornano oggetti *Future*: i risultati delle analisi vengono gestiti tramite callback (*onSuccess*, *onFailure*, *onComplete*) che vengono eseguite solo al termine dell'elaborazione, senza bloccare l'esecuzione del programma.

I risultati sono modellati da tre classi (`ClassDepsReport`, `PackageDepsReport`, `ProjectDepsReport`) che estendono una classe astratta generica `DepsReport<T>`. Questo design permette una composizione strutturata dei report: un report di progetto aggrega report di package, e ciascun report di package aggrega quelli delle classi.

Infine, l'utilizzo della libreria JavaParser consente di analizzare il codice sorgente Java tramite Abstract Syntax Tree (AST), per la ricerca delle dipendenze.

Versione reattiva

L'architettura è basata sul pattern Model-View-Controller (MVC), che separa la logica dell'applicazione dalla presentazione. La parte di analisi delle dipendenze è implementata nella classe `DependencyAnalyser`, che fornisce le funzionalità di analisi, mentre la parte di interfaccia grafica è gestita dalla classe `GUI`, realizzata utilizzando la libreria Swing di Java.

Il progetto utilizza il framework RxJava, libreria per la programmazione reattiva in Java, che consente di lavorare con flussi di dati asincroni (come input dell'utente, aggiornamenti dell'interfaccia grafica e ricezione dei risultati) e di gestire eventi in modo reattivo. La libreria fornisce un'implementazione del pattern Observer, che permette di osservare e reagire a eventi o cambiamenti di stato in modo semplice ed efficiente.

All'interno del programma, è implementata una semplice gestione della backpressure, che consente di limitare il numero di eventi emessi in un dato momento, evitando sovraccarichi e garantendo una reattività fluida. Questo è realizzato tramite la definizione di una dimensione massima per il buffer, che controlla quanti eventi (le classi da analizzare) possono essere emessi prima di essere elaborati. In futuro, il progetto potrebbe essere esteso per includere una gestione più avanzata della backpressure, ad esempio gestendo la frequenza di emissione degli eventi.

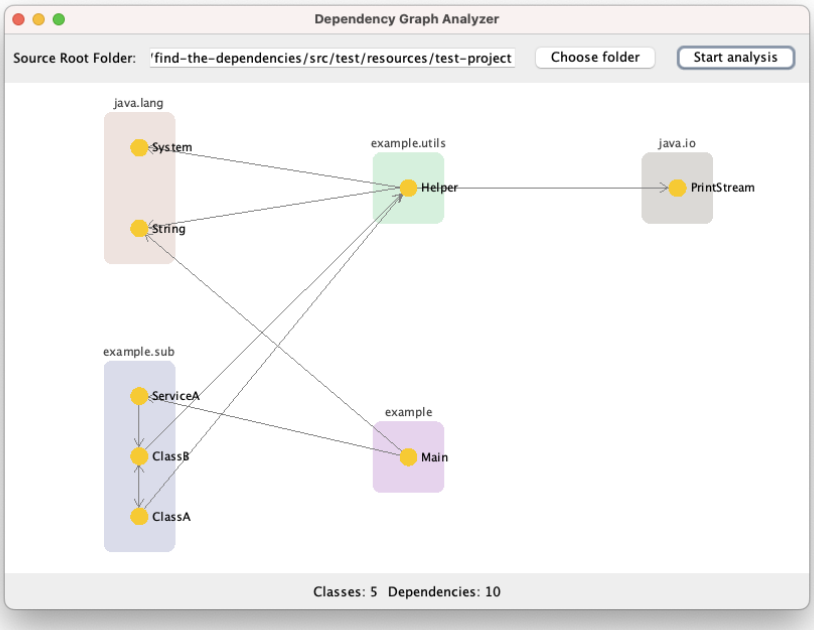

Figura 1: Rappresentazione dell'interfaccia grafica del sistema reattivo.

La GUI è progettata per essere reattiva, consentendo all'utente di selezionare un progetto e visualizzare le dipendenze in modo dinamico, man mano che le varie classi vengono analizzate. La disposizione dei nodi nel grafo è gestita raggruppando le classi in package, in modo da rendere più chiara la visualizzazione delle dipendenze.

Comportamento del sistema

Versione asincrona

Le tre reti di Petri illustrate nella fig. 2 rappresentano rispettivamente i tre metodi della libreria: `getClassDependencies`, `getPackageDependencies` e `getProjectDependencies`. Per ciascuna rete sono evidenziati i principali controlli, necessari per garantire il corretto funzionamento dell'analisi delle dipendenze. Ogni passaggio rappresenta una composizione asincrona che può portare al fallimento o al successo del processo.

In particolare, viene identificato con:

- n il numero di classi nel package;
- m il numero di package nel progetto.

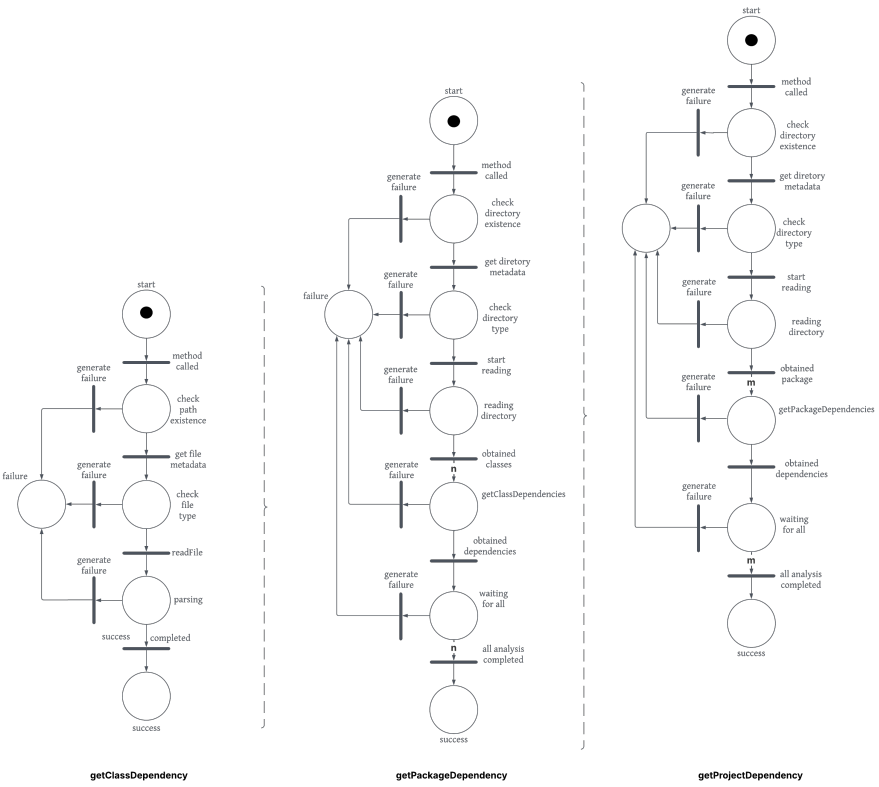

Figura 2: Rete di petri componenti asincrone.

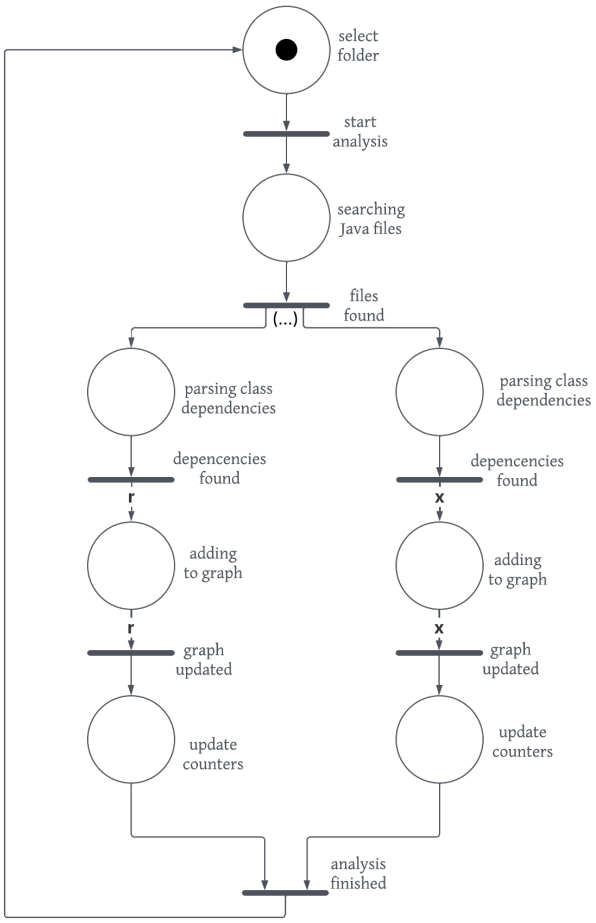

Figura 3: Rete di Petri programma reattivo.

Versione reattiva

La rete di Petri riportata in fig. 3 descrive il comportamento del programma nella sua versione reattiva, a partire dalla selezione del progetto da analizzare. L'attenzione è rivolta alla fase di individuazione delle classi Java, il cui numero, indicato con `(...)`, può variare, riflettendo la natura dinamica dei progetti analizzati. Per ciascuna viene identificato uno specifico percorso che cattura la sua autonomia nella ricerca delle rispettive dipendenze (in figura, quantificate come `r` e `x`). Solo al termine dell'analisi di tutte le classi è possibile procedere con l'analisi di altri progetti.