Relazione

-Virtual Thread:

La strategia risolutiva adottata si basa sull'utilizzo della programmazione asincrona e su un'architettura che sfrutta i thread virtuali per gestire in modo efficiente la concorrenza.

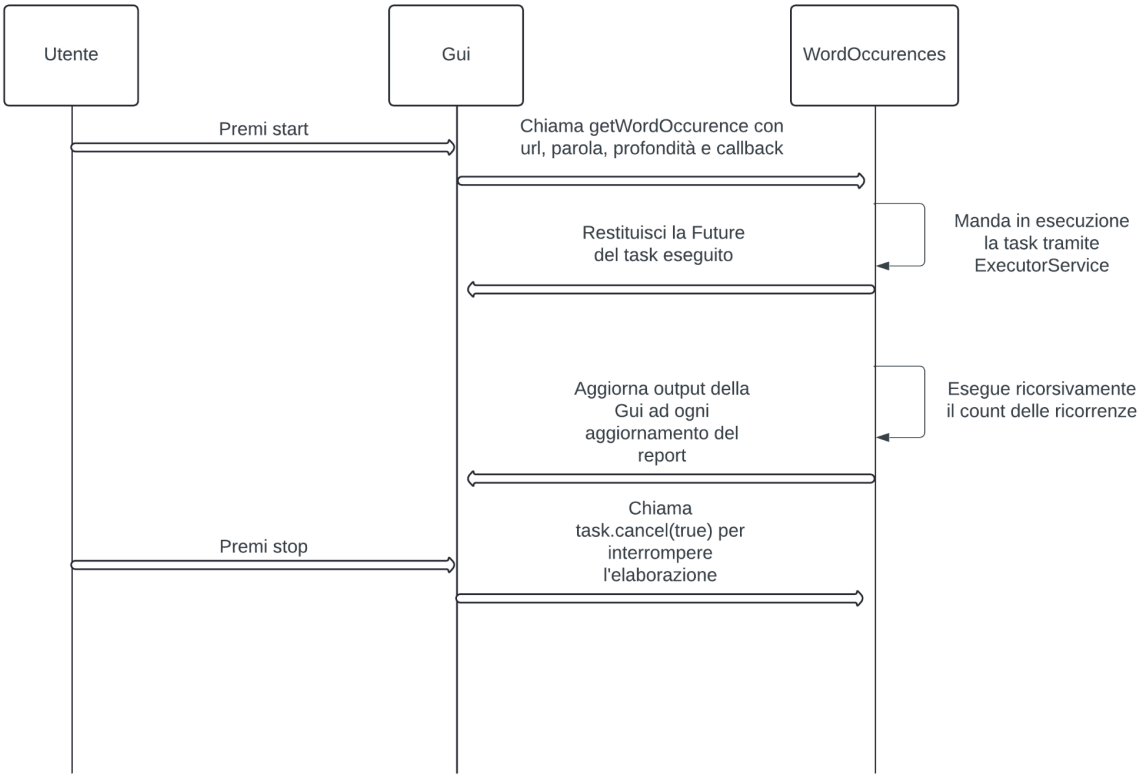
- Utilizzo della programmazione asincrona: La libreria WordOccurrences offre un'interfaccia asincrona per eseguire la ricerca delle ricorrenze delle parole nelle pagine web. Questo permette di avviare più operazioni di ricerca contemporaneamente senza bloccare il thread principale dell'applicazione.

- ThreadPool di thread virtuali: La classe utilizza un ExecutorService per gestire l'esecuzione delle operazioni asincrone. In particolare, viene utilizzato un ExecutorService basato su thread virtuali (newVirtualThreadPerTaskExecutor()) per sfruttare l'efficiente gestione dei thread virtuali introdotta da Java 17.

- Approccio ricorsivo: La ricerca delle occorrenze delle parole nelle pagine web avviene in modo ricorsivo. Ciò significa che, partendo dall'URL di partenza, vengono esplorate tutte le pagine collegate fino a una certa profondità specificata. Questo approccio consente di esaminare un ampio insieme di pagine in modo efficiente, sfruttando al meglio le capacità concorrenti del sistema.

- Callback per gli aggiornamenti del report: Durante l'elaborazione, vengono chiamati i metodi della callback ReportCallback per aggiornare l'interfaccia utente con i progressi dell'operazione. Questo permette all'utente di visualizzare

incrementalmente gli aggiornamenti del report mentre l'elaborazione è in corso.



-Reactive:

La strategia risolutiva si basa sull'utilizzo di un approccio reattivo per gestire in modo efficiente il conteggio delle occorrenze delle parole nelle pagine web, mantenendo un'interfaccia utente reattiva e scalabile. L'architettura proposta si compone di tre classi principali: Gui, WorkerTask e WordOccurences, che collaborano per consentire all'utente di avviare e interrompere il conteggio delle occorrenze delle parole in modo semplice ed efficiente.

Descrizione dell'architettura:

- Gui: Questa classe rappresenta l'interfaccia utente dell'applicazione. Consente all'utente di inserire l'URL del sito web da esaminare, la parola da cercare e la profondità di ricerca. Utilizzando i pulsanti "Start" e "Stop", l'utente può

avviare e interrompere il conteggio delle occorrenze delle parole. L'interfaccia utente rimane reattiva durante l'elaborazione, consentendo all'utente di interagire con l'applicazione senza blocchi.

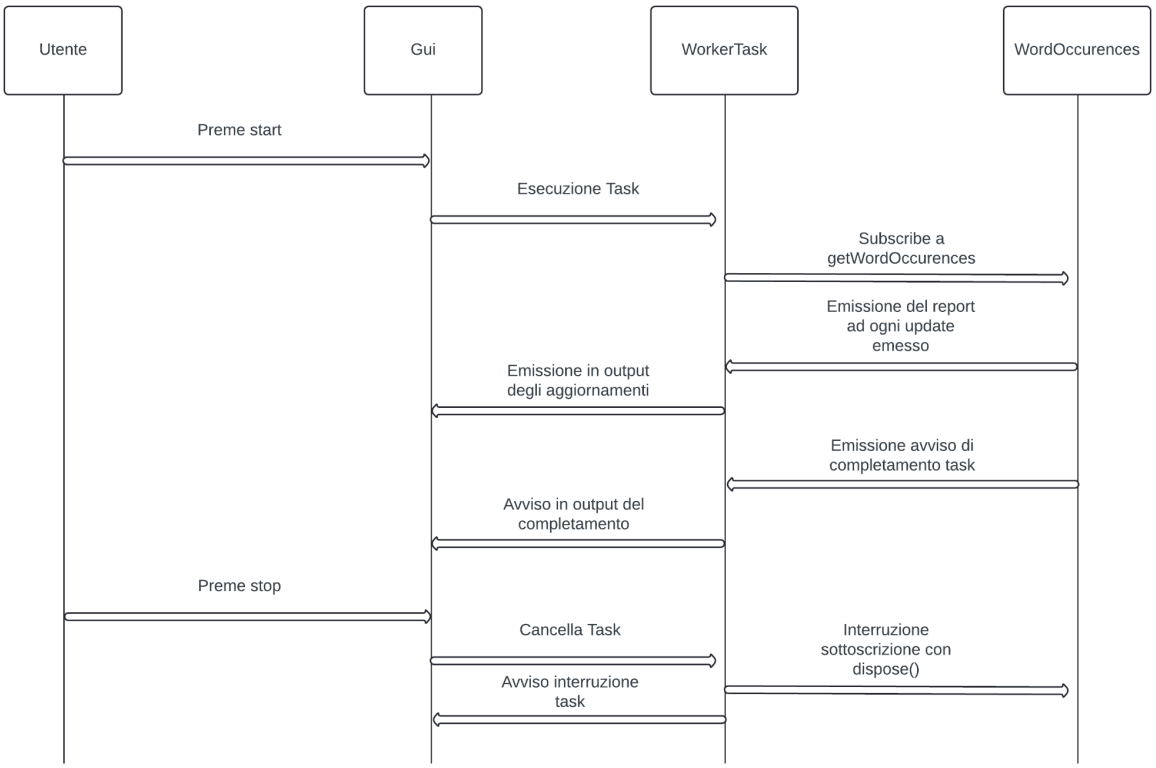
- WorkerTask: Questa classe estende SwingWorker e rappresenta il task che esegue il conteggio delle occorrenze delle parole in background. Utilizza un approccio reattivo utilizzando RxJava per gestire la sottoscrizione ai risultati del conteggio. Quando avviato, comunica con la classe WordOccurences per avviare il conteggio delle occorrenze. Riceve i risultati parziali e li aggiorna nell'interfaccia utente tramite l'oggetto outputArea. È in grado di interrompere il conteggio delle occorrenze quando richiesto dall'utente.

- WordOccurences: Questa classe gestisce effettivamente il conteggio delle occorrenze delle parole. Offre un metodo statico getWordOccurrences che restituisce un oggetto Observable, rappresentando il flusso di risultati reattivi. Utilizza un approccio ricorsivo per esaminare il contenuto delle pagine web e aggiornare il report delle occorrenze. Il conteggio delle occorrenze viene eseguito in modo asincrono su thread virtuali, consentendo all'applicazione di rimanere reattiva durante l'elaborazione.

Descrizione del comportamento del sistema:

Quando l'utente avvia il conteggio delle occorrenze delle parole tramite l'interfaccia utente, il task WorkerTask viene eseguito in background. Questo task sfrutta l'approccio reattivo per sottoscriversi al flusso di dati emessi da WordOccurences. Mentre WordOccurences esamina il contenuto delle pagine web, emette risultati parziali tramite oggetti Observable, che vengono ricevuti da WorkerTask. Quest'ultimo aggiorna l'output nell'interfaccia utente con i risultati parziali, consentendo all'utente di monitorare il progresso del conteggio delle occorrenze. L'utente ha anche la possibilità di interrompere il conteggio in qualsiasi momento tramite l'interfaccia utente.

Nel contesto dell'architettura reattiva proposta, l'oggetto Disposable viene utilizzato per gestire la sottoscrizione al flusso di dati emesso da un observable e per disporre delle risorse associare a questa sottoscrizione quando non sono più necessarie.
Nel nostro caso, l'oggetto Disposable viene utilizzato all'interno della classe WorkerTask per gestire la sottoscrizione al flusso di dati emesso da WordOccurences. Quando WorkerTask si sottoscrive a WordOccurences.getWordOccurrences(), riceve un oggetto Disposable che rappresenta la sottoscrizione al flusso di dati.
Quando l'operazione di conteggio delle occorrenze viene interrotta dall'utente (tramite il pulsante "Stop" nell'interfaccia utente), l'oggetto Disposable viene utilizzato per interrompere la sottoscrizione al flusso di dati. Questo viene fatto chiamando il metodo dispose() sull'oggetto Disposable. Questa azione permette di liberare le risorse associate alla sottoscrizione e interrompere eventuali operazioni in corso.



-Event-Loop:

La strategia risolutiva e l'architettura proposta si concentrano sull'utilizzo della programmazione asincrona basata su event loop, con l'obiettivo di

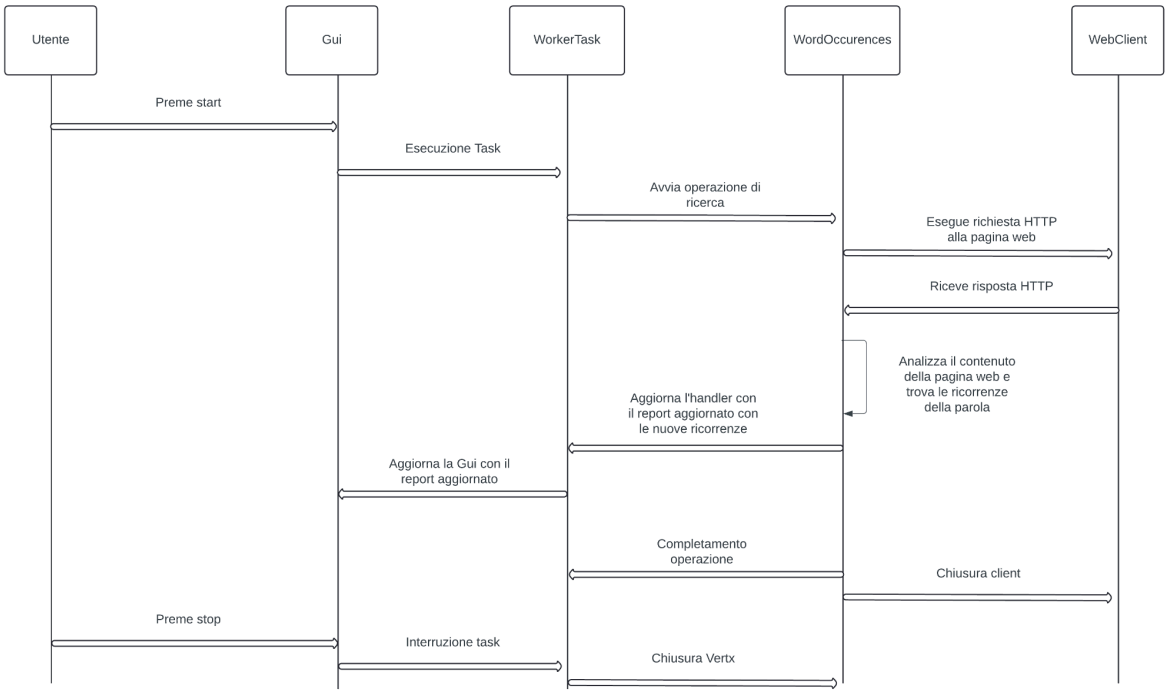
gestire le operazioni di rete in modo efficiente e non bloccante. Qui di seguito una descrizione dettagliata:

Strategia risolutiva:

- Programmazione asincrona: L'architettura si basa sull'approccio asincrono, in cui le operazioni di I/O, come le richieste HTTP per ottenere il contenuto delle pagine web, vengono gestite in modo non bloccante. Ciò consente al sistema di continuare a elaborare altre richieste e operazioni senza dover attendere il completamento delle operazioni di rete.
- Utilizzo di Vert.x: Il framework Vert.x viene utilizzato per implementare l'event loop e gestire le operazioni asincrone. Vert.x offre un'architettura ad eventi che permette di gestire concorrentemente molteplici operazioni di I/O su un singolo thread.
- Gestione dei thread: Grazie all'event loop di Vert.x, non è necessario gestire direttamente i thread. Le operazioni di I/O vengono eseguite in modo concorrente su un singolo thread, evitando il sovraccarico associato alla creazione e gestione di molti thread.

Architettura proposta:

- WordOccurences: Questa classe gestisce le operazioni di rete per ottenere il contenuto delle pagine web e trovare le occorrenze della parola desiderata. Utilizza Vert.x per gestire le richieste HTTP in modo non bloccante e asincrono.
- Gui: Rappresenta l'interfaccia utente dell'applicazione. L'utente può inserire l'URL della pagina web, la parola da cercare e la profondità di ricerca. Premendo il pulsante "Start", viene avviato un task in background per trovare le occorrenze della parola. L'utente può interrompere l'operazione premendo il pulsante "Stop".
- WorkerTask: Questa classe estende SwingWorker e rappresenta il task in background avviato dall'interfaccia utente. Utilizza Vert.x per comunicare con WordOccurences. Una volta ricevuti i risultati, aggiorna l'interfaccia utente con le occorrenze trovate.



Analisi delle prestazioni delle architetture basate su Event-Loop, Reactive e Virtual-Thread

Event-Loop:

- Numero di richieste: 500
- Utilizzo medio della CPU: 28.6%
- Thread nell'Event-Loop: 15
- Tempo totale: 11428 millisecondi
- Throughput: 43 richieste al secondo

Reactive:

- Numero di richieste: 500
- Utilizzo medio della CPU: 19%
- Utilizzo dei thread: 500 RxCachedThreadScheduler
- Tempo totale: 5143 millisecondi
- Throughput: 97.2 richieste al secondo

Virtual-Thread:

- Numero di richieste: 500
- Tempo totale: 3758 millisecondi
- Utilizzo medio della CPU: 57.6%
- Thread: ForkJoinPoolWorker 8
- Throughput: 133.04 richieste al secondo

Considerazioni:

- L'architettura basata su Event-Loop mostra un throughput più basso rispetto alle altre due architetture, con un utilizzo medio della CPU significativamente più alto e un tempo totale maggiore per elaborare tutte le richieste. Questo potrebbe essere dovuto al maggior numero di thread nell'Event-Loop e alla maggiore complessità nell'elaborare le richieste in modo sincrono.

- L'architettura reattiva mostra un throughput superiore rispetto all'Event-Loop, con un utilizzo medio della CPU inferiore e un tempo totale inferiore per elaborare le stesse richieste. Questo suggerisce che l'approccio reattivo potrebbe essere più efficiente nell'elaborare un gran numero di richieste in modo asincrono e non bloccante.

- L'architettura basata su Virtual-Thread mostra il miglior throughput e il tempo totale più basso per elaborare le richieste. Anche se l'utilizzo medio della CPU è più alto rispetto alle altre due architetture, il tempo di esecuzione più breve suggerisce che l'approccio basato su Virtual-Thread può gestire un carico di lavoro più elevato in modo più efficiente.

In base a queste osservazioni, l'architettura basata su Virtual-Thread sembra offrire le migliori prestazioni complessive in termini di throughput e tempo di risposta, seguita dall'architettura reattiva e infine dall'architettura basata su Event-Loop.