UNIVERSITÀ
di VERONA

Word Automa

Ingegneria del Software
Documentazione progetto

Anno Accademico: 2023-2024

Studenti

Casalini Marco
VR490461

Pancheri Ermes
VR487531

Docenti

Carlo Combi

Pietro Sala

Indice

Pagina

1 Requisiti

In questa sezione andremo ad elencare i requisiti richiesti dal compito e i requisiti che ci siamo imposti dopo aver analizzato il problema richiesto, basandoci sulla nostra esperienza:

1.1 Requisiti funzionali

Il sistema deve soddisfare i seguenti requisiti funzionali:

- consentire all'utente di inserire una parola e avviare una ricerca all'interno di un grafo;
- visualizzare i nodi attraversati e gli archi corrispondenti durante la ricerca della parola nel grafo;
- indicare se la parola è accettata o rifiutata, basandosi su un nodo finale del grafo;
- permettere all'utente di creare grafi e visualizzare le connessioni fra i nodi.

1.2 Requisiti non funzionali

Oltre ai requisiti funzionali, il programma deve soddisfare i seguenti requisiti non funzionali:

- garantire una risposta rapida per la ricerca di parole;
- essere facile da usare, con un'interfaccia utente intuitiva;
- garantire la compatibilità con diversi formati di input per i grafi.

2 Casi d'uso

breve descrizione di cosa sono i casi d'uso...

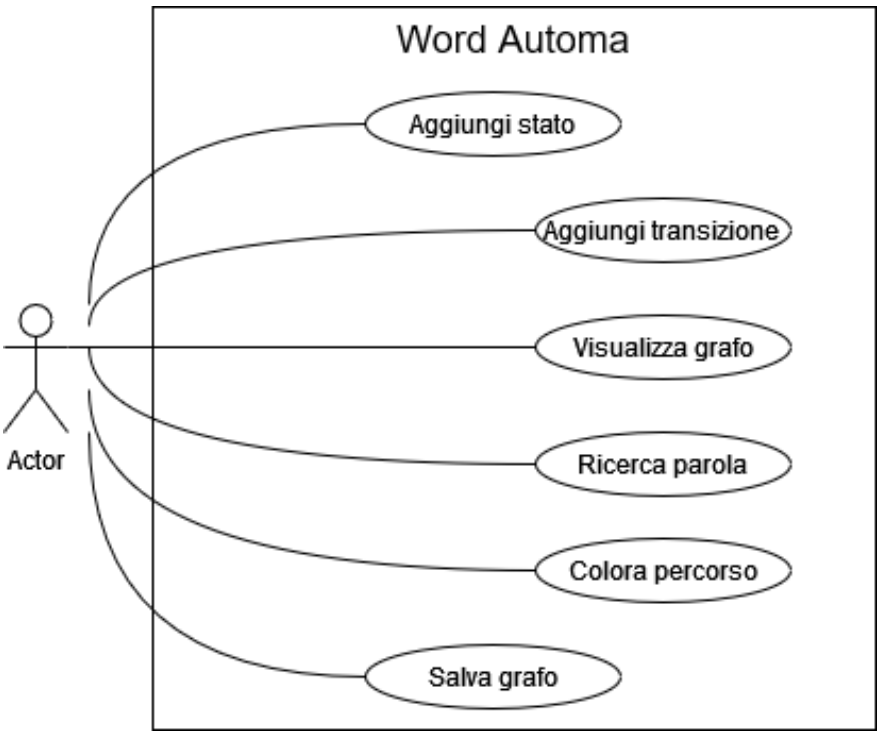

Figura 1: Casi d'uso

2.1 Caso d'uso: Aggiungi stato

- **Attori principali**: Utente
- **Pre-condizioni**: Il programma è in esecuzione.
- **Sequenza degli eventi**:
  1. L'attore seleziona l'opzione per aggiungere uno stato e inserisce le informazioni.
  2. Il sistema controlla se la stringa aggiunta è accettablie.
- **Post-condizioni**: L'utente conoscerà l'esito dell'operazione.
- **Descrizione**: L'utente aggiunge uno stato all'automa.

2.2 Caso d'uso: Ricerca di una parola nel grafo

- **Attori principali**: Utente
- **Pre-condizioni**: L'utente ha caricato correttamente un grafo e ha accesso al sistema.
- **Sequenza degli eventi**:
  1. L'utente scrive una parola nel campo di ricerca e invia conferma.
  2. In caso di successo mostra il precorso attraversato con nodi e valori.

- **Post-condizioni**: Il sistema restituisce il risultato della ricerca e visualizza i nodi e gli archi attraversati.
- **Descrizione**: L'utente inserisce una parola nel campo di ricerca. Il sistema analizza la parola carattere per carattere, verifica se esiste un percorso nel grafo e alla fine visualizza se la parola è accettata o rifiutata, con una lista dei nodi e degli archi percorsi.

3 Diagrammi di attività

Il seguente diagramma di attività illustra il flusso dell'azione nel sistema nell'aggiunta di uno stato nel grafo con la possibilità di assegnare l'etichetta di nodo iniziale o nodo finale:

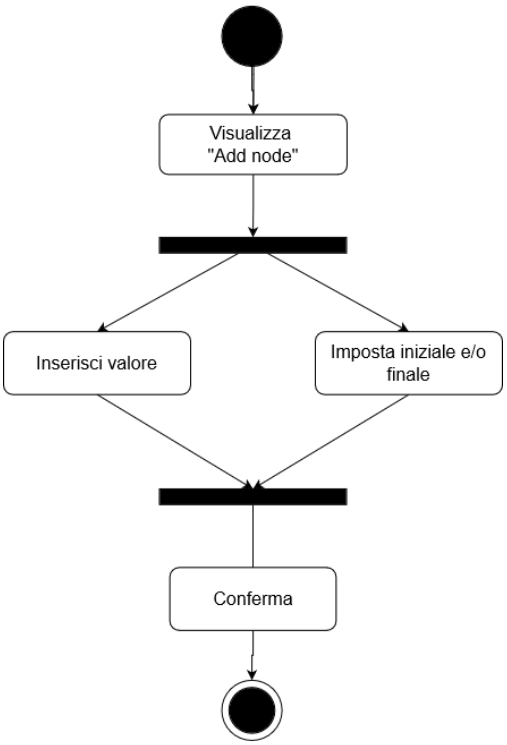

Figura 2: Diagramma di attività: Aggiungi stato.

Il seguente diagramma di attività illustra il flusso dell'azione nel sistema durante la ricerca di una parola nel grafo:

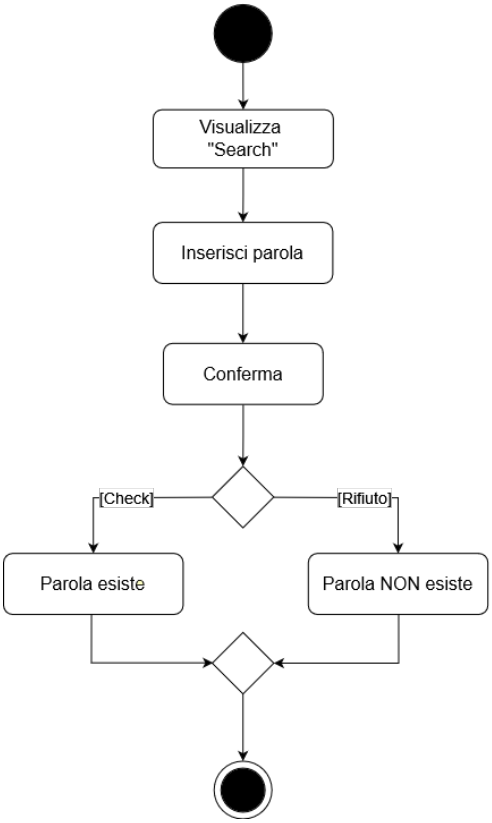

Figura 3: Diagramma di attività: Ricerca nel grafo.

Nelle pagine che seguono vengono riportate le principali classi con i loro campi e metodi:

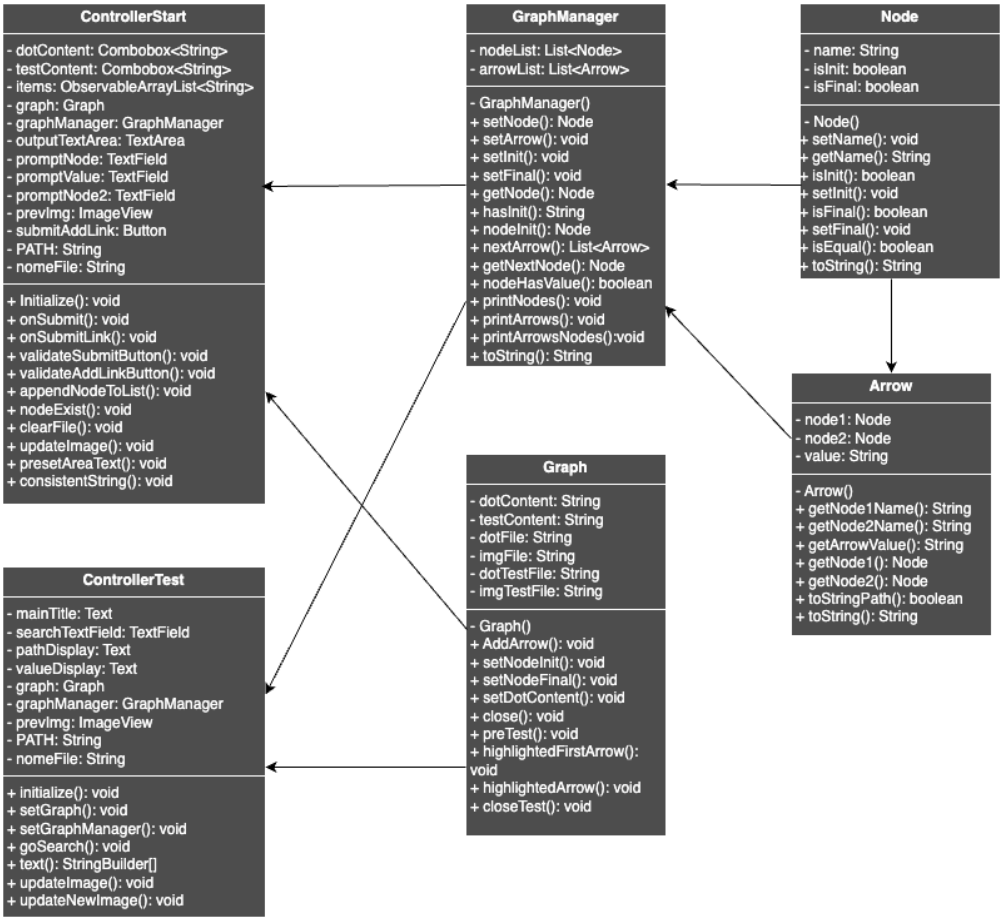

Figura 4: Diagrammi delle classi.

4 Sviluppo: progetto dell'architettura ed implementazione del sistema

4.1 Note sul processo di sviluppo

Lo sviluppo del progetto ha combinato metodologie plan-driven e agile, adottando un approccio incrementale. Il codice è stato scritto principalmente tramite il pair programming, favorendo una maggiore qualità del software e promuovendo la condivisione delle competenze tra i membri del team.

La fase iniziale di progettazione ha comportato la definizione di requisiti minimi, basati sulle specifiche fornite, che l'applicazione avrebbe dovuto soddisfare. Questi requisiti, suddivisi in funzionali e non funzionali, sono stati utilizzati per creare un piano di sviluppo che includeva la scelta dell'architettura software. I requisiti raccolti sono stati poi trasformati in specifiche tecniche dettagliate, che hanno guidato lo sviluppo del codice. La distribuzione dei compiti è avvenuta inizialmente per macro-aree, come front-end e back-end, e successivamente in base a unità di lavoro specifiche, permettendo al team di lavorare su più aspetti contemporaneamente, migliorando così l'efficienza e la capacità di adattamento ai cambiamenti.

La gestione del codice è stata semplificata dall'uso di un sistema di versionamento distribuito, grazie a GitHub, che ha consentito ai membri del team di lavorare in parallelo sulla stessa base di codice, garantendo la tracciabilità delle modifiche attraverso le varie versioni. Il codice è stato sviluppato prevalentemente in linguaggio Java, utilizzando JavaFX per l'interfaccia grafica e IntelliJ come ambiente di sviluppo (IDE).

Parallelamente allo sviluppo, sono stati condotti test sui singoli componenti per assicurare la correttezza del codice e la conformità alle specifiche. Raggiunto un livello soddisfacente di qualità e funzionalità, il team ha eseguito operazioni di refactoring e ottimizzazione del codice.

La documentazione finale è stata completata alla fine del processo di sviluppo, per garantire la coerenza tra il codice e le informazioni riportate, evitando ridondanze e assicurando la correttezza dei contenuti.

4.2 Progettazione e pattern architetturali

Durante lo sviluppo, sono state adottate diverse tecniche di progettazione per garantire la modularità e la manutenibilità del sistema. Si è optato per l'uso di un pattern *Model-View-Controller* (MVC), che ha permesso di separare chiaramente la logica di controllo, il modello dati e la visualizzazione.

Considerati i requisiti del sistema, principalmente quelli relativi alla gestione e visualizzazione dell'automa, oltre alla necessità di separare i dati dalla logica di presentazione e controllo, il team ha deciso di adottare un pattern architetturale di tipo **Model View Controller**. Il pattern MVC è implicitamente utilizzato attraverso la separazione della logica di business (gestione del grafo) e l'interfaccia utente (gestita dai controller). I controller come ControllerTest e ControllerStart fungono da intermediari tra il modello (i dati del grafo) e la vista (la UI fornita dai file FXML). Questo pattern è stato applicato per garantire un'interazione chiara tra le componenti e una facile manutenzione del codice. Si è optato per suddividere le classi operative del programma in tre pacchetti distinti: uno dedicato alla parte grafica, uno relativo al modello e un altro per la logica di controllo. In dettaglio:

- **Model**, composto dale classi `Graph` e `GraphManager`.
- **View**, composto dai file `FXML` per la visualizzazione dall'interfaccia grafica.
- **Controller**, composto dalle classi `ControllerStart` e dal `ControllerTest` per l'interazione con l'utente.

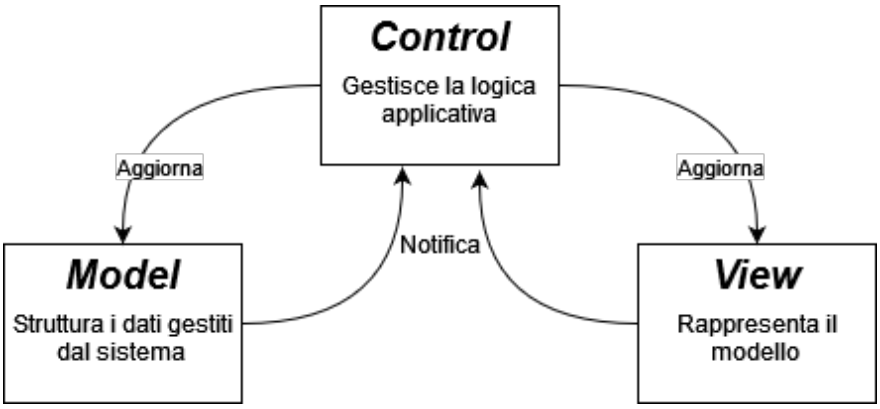

Figura 5: Pattern architetturale Model-View-Controller.

4.3 Diagrammi di sequenza del software implementato

Il seguente diagramma di sequenza mostra come avviene l'interazione durante una ricerca nel grafo.

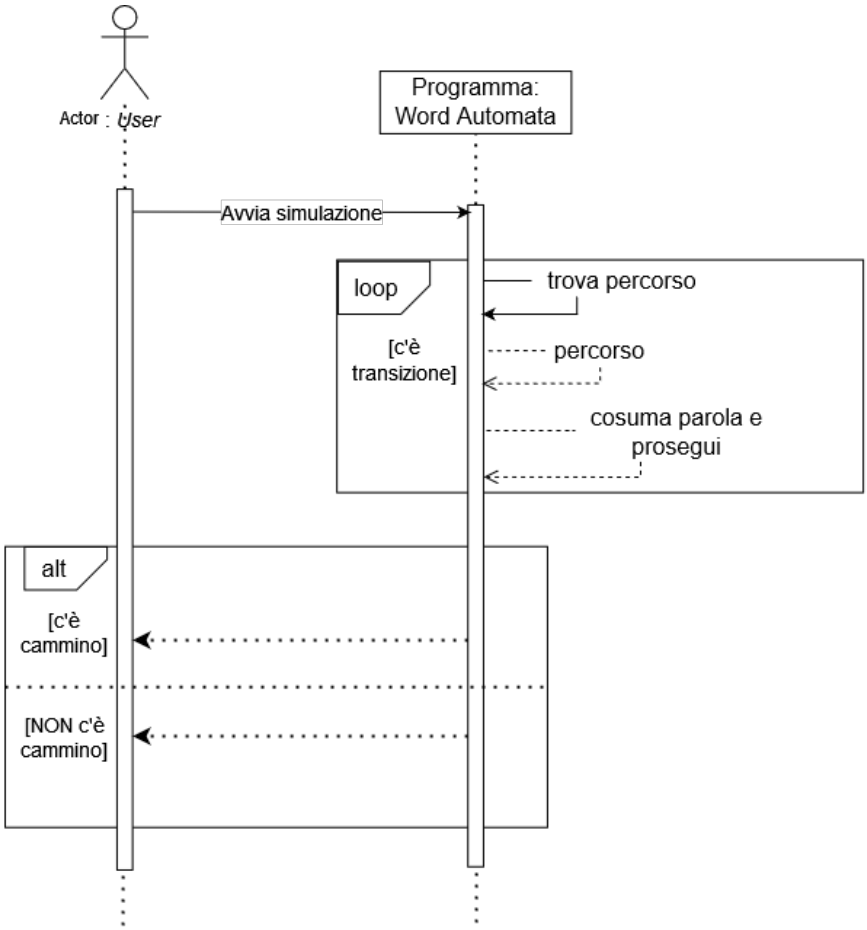

Figura 6: Diagramma di sequenza: Avvio ricerca.

5 Attività di test e validazione

5.1 Breve commento sulle interfaccia

L'interfaccia utente è semplice e intuitiva, con un campo di input per la ricerca della parola e un'area di visualizzazione del percorso nel grafo. Vengono visualizzati in tempo reale i nodi e gli archi attraversati.

5.2 Ispezione codice e documentazione

Il codice è stato sottoposto a ispezioni regolari per garantire la qualità e la conformità agli standard di programmazione.

```
Graph configuration:

Nodes:
  Node{name='q0', isInit=true, isFinal=false}
  Node{name='q1', isInit=false, isFinal=false}
  Node{name='q2', isInit=false, isFinal=false}
  Node{name='q3', isInit=false, isFinal=true}
  Node{name='q4', isInit=false, isFinal=true}

Arrows:
  Arrow{from=q0, value='aab', to=q1}
  Arrow{from=q0, value='abbb', to=q2}
  Arrow{from=q2, value='aba', to=q1}
  Arrow{from=q1, value='b', to=q2}
  Arrow{from=q1, value='a', to=q3}
  Arrow{from=q2, value='b', to=q3}
  Arrow{from=q3, value='bb', to=q4}
  Arrow{from=q4, value='b', to=q2}
  Arrow{from=q4, value='a', to=q1}
  Arrow{from=q3, value='abb', to=q3}

Testing word: 'aababbbb'
Starting pathSelection with key: aababbbb
Word accepted with finale node: q3
    Path 'abbbabaaba' accepted. Reached node: q3

Testing word: 'abbbaba'
Starting pathSelection with key: abbbaba
Final node is NOT an accepting state.
    Path 'abbbabaaba' NOT accepted.

Testing word: 'abbbabaaba'
Starting pathSelection with key: abbbabaaba
No matching arrow found for remaining key: ba
    Path 'abbbabaaba' NOT accepted.
```

Figura 7: Ispezione: Test automatico.

5.3 Test degli sviluppatori

Sono stati eseguiti test di integrazione per verificare che i vari componenti del sistema interagiscano correttamente. Questi test hanno coinvolto la ricerca di parole con diverse lunghezze e complessità.

5.4 Test utente generico

Il software è stato sottoposto a test da parte di alcuni utenti con limitate competenze informatiche e completamente estranei allo sviluppo. In questa fase, non si è voluto in alcun modo guidare l'esperienza, per evitare di influenzare i risultati. Gli utenti sono stati lasciati liberi di esplorare il sistema autonomamente fornendo solo una generica descrizione degli

obiettivi sull'uso del software. L'obiettivo principale del test era identificare eventuali errori non evidenti agli sviluppatori.

6 Possibili aggiunte future

Successivamente ai test abbiamo notato, sia noi sviluppatori che i tester di terze parti, la mancanza di alcune possibili funzioni. In un aggiornamento futuro dell'applicazione sarebbero possibili diverse implementazioni di funzionalità:

- Salvataggio file: l'implementazione di poter nominare e salvare i file può essere molto utile se si lavora con più automi, dando così la possibilità di riprenderne di vecchi e crearne di nuovi. Una possibile implementazione alla quale abbiamo pensato sarebbe quella di aggiungere un pagina iniziale: contenente un bottone per creare un nuovo file, con annesso pop-up per aggiungerne il nome, per poi passare alla gestione del grafo; una lista contenente i file già creati, dopo averne selezionato uno si passerà alla sua gestione;
- Eliminazione di nodi e stati: per dare più agilità all'utente, nel caso sbagli o voglia cambiare l'alfabeto o i suoi stati;
- Bottone per tornare indietro: un bottone per tornare dalla pagina di test a quella dell'impostazione dell'automa sarebbe sicuramente un ulteriore funzionalità per dare all'utente la libertà di muoversi nel sistema, in caso dopo i test voglia aggiungere, togliere o cambiare qualcosa.