

Relazione Progetto Ingegneria del Software - PetriHub

Alessio Vivaldelli 

Antonio Iovine 

Davide Sala 

Luglio 2025

Contents

1 Requisiti e Interazioni Utente-Sistema	3
1.1 Casi d'uso per l'admin	3
1.1.1 Creazione di una PN	3
1.1.2 Gestione delle computazioni	5
1.2 Casi d'uso generali	5
1.2.1 Sottoscrizione a una PN	7
1.2.2 Inizio di una computazione	7
1.2.3 Esecuzione di una transizione	8
1.3 Diagrammi di stato	9
2 Sviluppo	12
2.1 Note sullo sviluppo	12
2.2 Stack Tecnologico e Metodologie di Sviluppo	12
2.2.1 Controllo di Versione e Flusso di Lavoro	12
2.2.2 Gestione del Progetto e delle Dipendenze	12
2.2.3 Framework e Librerie Principali	12
2.3 Serializzazione e deserializzazione delle PN	14
2.3.1 Integrazione dello Standard PNML (ISO/IEC 15909-2)	14
2.3.2 Utilizzo e Adattamento di PNML nel Progetto	14
2.3.3 Estensione dello Standard per Attributi Personalizzati	14
2.4 Elenco dei Design Pattern utilizzati	15
2.5 Diagrammi delle classi	16
2.5.1 Database	16
2.5.2 Petri net	19
2.5.3 Tabelle	23
2.6 Diagrammi delle sequenze	26
2.6.1 Petri net	26
2.6.2 Gestione delle reti	36
2.6.3 Creazione delle Tabelle	42
3 Validazione e test	45
3.1 Test interni	45
3.2 Test esterni	45
3.2.1 Metodologia di Test	45
3.2.2 Risultati Emersi	46
3.3 Unit Testing	47

1 Requisiti e Interazioni Utente-Sistema

Il sistema prevede la creazione, sottoscrizione e gestione di reti di Petri(che chiameremo PN per semplicità) da parte di diversi utenti. Un utente può essere user normale o admin, quest'ultimo con la facoltà di poter creare delle proprie PN.

1.1 Casi d'uso per l'admin

Dopo aver effettuato l'autenticazione, all'utente Admin vengono rese disponibili tutte le azioni necessarie per la creazione, condivisione e gestione delle varie PN.

1.1.1 Creazione di una PN

Attraverso il pulsante ”new net” l'Admin può generare una nuova PN, prima inserendo un nome non in uso e crearla poi mediante un'interfaccia. Il sistema effettua automaticamente dei controlli per verificare la che rete creata sia una rete ”valida”.

mediante un'interfaccia per la reali

Attore: Admin

Precondizioni: utente Admin autenticato

Passi:

1. l'admin accede al sistema
2. l'admin seleziona ”new net”
3. l'admin procede alla creazione della rete attraverso ap-
posta interfaccia

Postcondizioni: l'admin visualizza un messaggio di con-
ferma di salvataggio della rete e ritorna alla homepage.

La rete creata, se valida, verrà automaticamente salvata nel database.

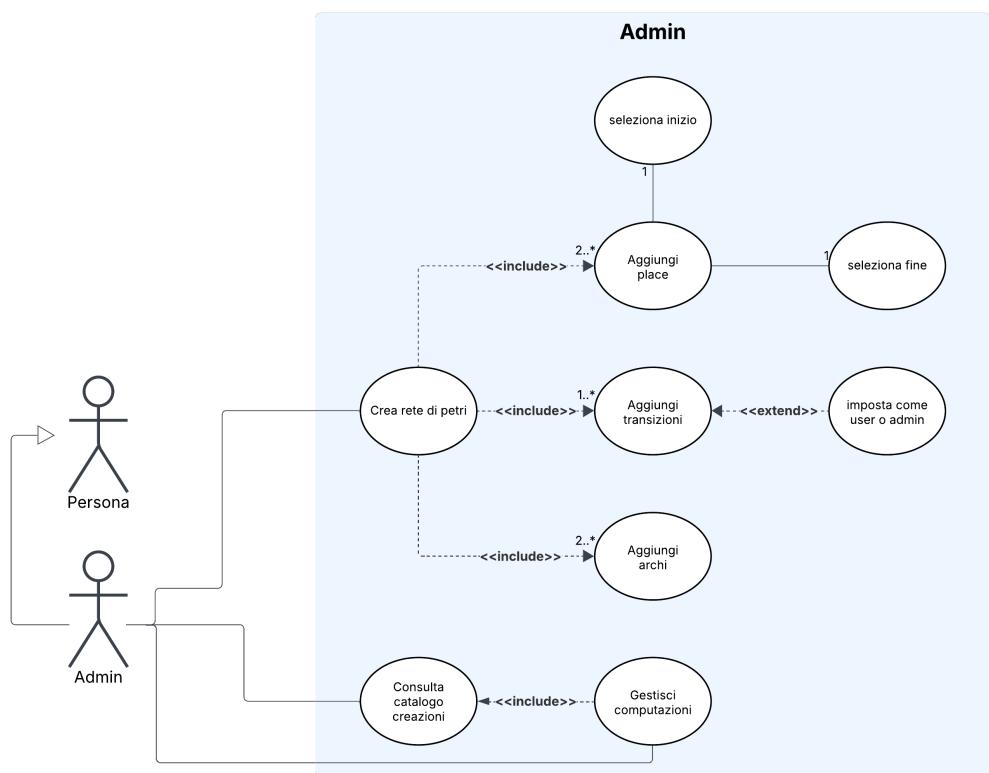


Figure 1: Casi d'uso per l'Admin

1.1.2 Gestione delle computazioni

Cliccando sulla sezione "my nets", apposita per l'Admin, è possibile visualizzare informazioni relative alle reti create e ad eventuali computazioni iniziate.

Attore: Admin

Precondizioni: utente Admin autenticato

Passi:

1. l'admin accede al sistema
2. l'admin seleziona "my nets"
3. il sistema mostra le reti create dall'utente e, selezionando ognuna di queste, la lista di utenti sottoscritti
4. l'utente accede la rete di cui vuole modificare la computazione

Postcondizioni: l'utente visualizzerà lo stato attuale della computazione, con la possibilità di cancellarla o proseguirla

1.2 Casi d'uso generali

Così come per l'Admin, all'utente standard che si è autenticato verrà mostrata la stessa interfaccia, ma con delle differenze nel numero di operazioni possibili.

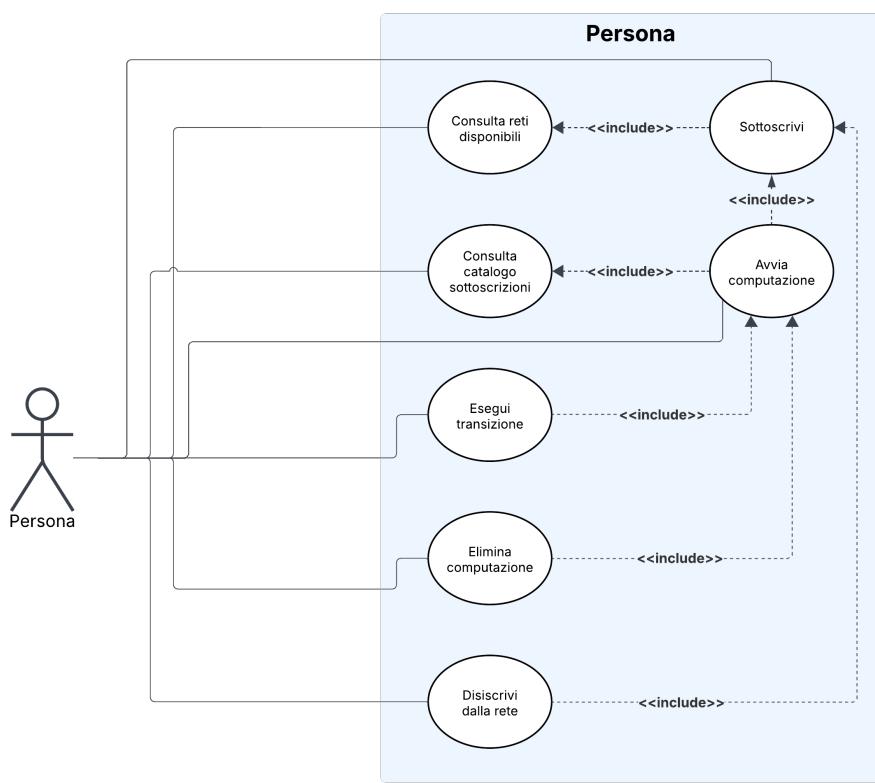


Figure 2: Casi d'uso User

1.2.1 Sottoscrizione a una PN

Per iscriversi a una nuova PN, un utente (oppure un Admin, a meno che non si tratti del creatore) può ricercare una rete di interesse nella sezione Discover della Home.

Attore: User o Admin

Precondizioni: utente autenticato e non creatore della rete
Passi:

1. l'utente accede al sistema
2. l'utente seleziona "discover"
3. il sistema mostra le reti disponibili all'utente, ossia quelle a cui non è già sottoscritto e di cui non è il creatore
4. l'utente sceglie la rete a cui sottoscriversi
5. nell'interfaccia che apparirà, l'utente può iscriversi premendo "subscribe"

Postcondizioni: l'utente sottoscritto alla rete, rivedrà la stessa nella sezione "my subs"

A seguito della sottoscrizione, l'utente può procedere all'inizializzazione della computazione (spiegata di seguito) o ritornare alla schermata Home.

1.2.2 Inizio di una computazione

Per le PN cui è stata già eseguita la sottoscrizione, l'utente sottoscritto può iniziare la computazione per quella rete da apposita interfaccia.

Attore: User o Admin

Precondizioni: utente autenticato e non creatore della rete

Passi:

1. l'utente accede al sistema
2. l'utente seleziona "my subs"
3. il sistema mostra le reti a cui l'utente si è sottoscritto,
4. l'utente sceglie la rete di cui iniziare la computazione
5. nell'interfaccia che apparirà, l'utente può premere il pulsante d'avvio sopra l'immagine per iniziare la computazione

Postcondizioni: all'utente verrà mostrato quali sono le transizioni che sono eseguibili in quel momento.

1.2.3 Esecuzione di una transizione

Solo dopo aver dato il via alla computazione, l'interfaccia fornirà, all'utente sottoscritto, la possibilità di eseguire gli step di computazione (solo quelli che lo riguardano) disponibili.

Attore: User o Admin

Precondizioni: utente autenticato e creatore di quella rete o sottoscritto ad essa, computazione per quella rete avviata

Passi:

1. l'utente accede al sistema
2. l'utente seleziona "my nets" (o "my subs" in base ai casi)
3. selezionando la rete d'interesse, il sistema mostrerà gli step eseguibili dal punto attuale, attraverso apposita interfaccia
4. l'utente sceglie e seleziona la transizione da sparare

Postcondizioni: l'immagine del progresso della rete, assieme alla lista dei marking place e le transizioni che sono ora disponibili, viene aggiornata.

Qualora dovesse ripresentarsi in futuro la possibilità per l'utente di eseguire delle transizioni proprie, riceverà in-app una notifica riconducibile alla specifica rete. Dovesse terminare la computazione, verrà mostrato a video un messaggio informativo.

1.3 Diagrammi di stato

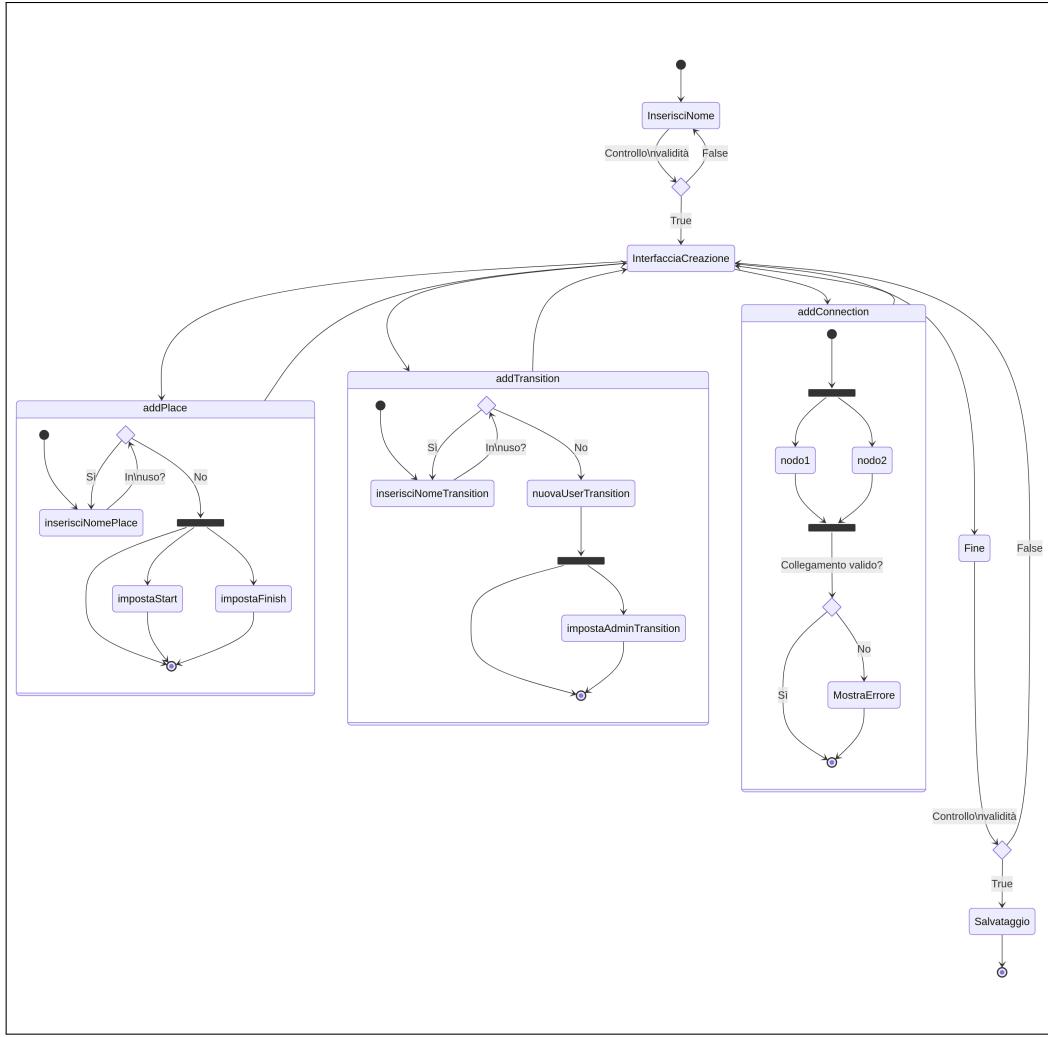


Figure 3: Diagramma di stato per creazione della rete

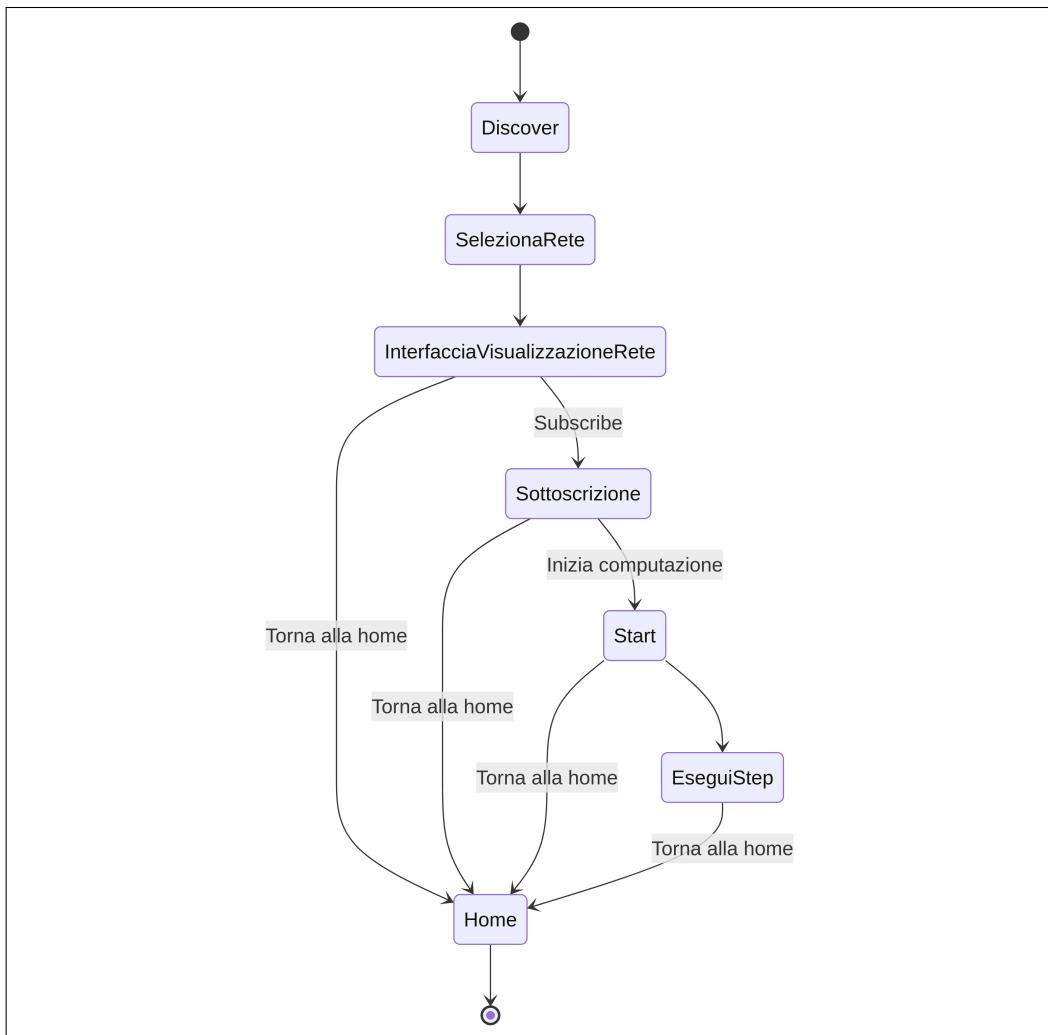


Figure 4: Diagramma di Stato per sottoscrizione

2 Sviluppo

2.1 Note sullo sviluppo

Il flusso di lavoro si è articolato in cicli brevi, all'interno dei quali abbiamo mantenuto una rigorosa sequenza di progettazione, implementazione e verifica: ogni funzione è stata prima definita a livello di requisiti, poi tradotta in specifiche tecniche e infine tradotta in codice, per garantire solidità e coerenza ad ogni rilascio.

La scrittura del codice è avvenuta prevalentemente in modalità pair programming: due sviluppatori a turno si sono occupati dello stesso modulo, così da minimizzare errori e accelerare il processo di revisione.

2.2 Stack Tecnologico e Metodologie di Sviluppo

2.2.1 Controllo di Versione e Flusso di Lavoro

Per la gestione del codice sorgente è stato utilizzato **Git**, con **GitHub** come piattaforma. È stato adottato il modello di branching **Git-flow**, un approccio che ha permesso di organizzare lo sviluppo in modo efficiente: La netta separazione tra i branch **feature**, **develop** e **main**.

2.2.2 Gestione del Progetto e delle Dipendenze

Per l'automazione del build, la gestione delle dipendenze e la standardizzazione della struttura del progetto, è stato scelto **Apache Maven**. Il suo utilizzo ha portato i seguenti vantaggi:

- **Gestione Centralizzata delle Dipendenze:** Tutte le librerie esterne sono definite nel file `pom.xml`, specificandone versione e scope, garantendo build riproducibili ed evitando conflitti.
- **Automazione del Ciclo di Vita:** Grazie al lifecycle standard di Maven, l'esecuzione di fasi come `test` o `package` ha permesso di automatizzare la compilazione e l'esecuzione dei test unitari.

2.2.3 Framework e Librerie Principali

L'architettura software si appoggia su un insieme di librerie e framework fondamentali.

2.2.3.1 JavaFX L’intera interfaccia grafica (GUI) dell’applicazione è stata sviluppata utilizzando **JavaFX**, il framework per la creazione di applicazioni client in Java. È stato scelto per la sua architettura flessibile, che si sposa bene con il pattern **Model-View-Controller (MVC)** da noi adottato.

2.2.3.2 JavaFXSmartGraph Per la visualizzazione e l’interazione con le PN è stata utilizzata come base la libreria **JavaFXSmartGraph**. Questa libreria è stata integrata localmente nel progetto per poter apportare modifiche al suo codice sorgente, adattandola alle esigenze specifiche del dominio delle reti di Petri. Le personalizzazioni principali includono:

- **Rendering di Nodi Personalizzati:** È stata estesa la capacità di rendering per visualizzare forme personalizzate per gestire, ad esempio, i token nei place.
- **Gestione aumentata degli eventi:** Sono stati implementati gestori di eventi personalizzati per il click su nodi, archi e sul canvas stesso, abilitando così le funzionalità di creazione, connessione e cancellazione interattiva degli elementi della rete.

2.2.3.3 JUnit 5 Per la fase di testing è stato impiegato **JUnit 5**. Questa libreria ha permesso di implementare una suite di test unitari (descritta nel capitolo dedicato al Testing) per validare la correttezza delle classi del modello e della logica di business. L’integrazione con Maven ha automatizzato l’esecuzione di questi test a ogni build, garantendo un feedback continuo sulla qualità del codice.

2.2.3.4 SQLite e JDBC Per la persistenza dei dati relativi a utenti, reti e computazioni, è stato scelto **SQLite** come motore di database. Questa scelta è stata motivata dalla sua natura *serverless* e basata su file, che la rende ideale per un’applicazione desktop senza richiedere l’installazione di un server di database separato. L’interazione con il database dall’applicazione Java è gestita tramite il driver **JDBC** specifico per SQLite, la cui dipendenza è inclusa nel `pom.xml`

2.3 Serializzazione e deserializzazione delle PN

2.3.1 Integrazione dello Standard PNML (ISO/IEC 15909-2)

Per poter salvare in locale la struttura delle PN create con il nostro sistema, abbiamo adottato lo standard PNML (Petri Net Markup Language). Questo standard è formalmente definito dalla norma ISO/IEC 15909-2, che fornisce una grammatica XML universale per la rappresentazione di modelli di PN e delle loro estensioni.

Questo ci consente di prendere i file generati dalla nostra applicazione ed usarli su simulatori online compatibili con questo standard, ad esempio <https://pes.vsb.cz/petrineteditor//#/model>.

2.3.2 Utilizzo e Adattamento di PNML nel Progetto

Nel nostro progetto, le funzionalità di salvataggio e caricamento delle PN sono gestite da due classi principali che implementano la logica di (de)serializzazione da e verso il formato PNML:

- **PNMLSerializer:** Questa classe è responsabile della scrittura. Prende un’istanza del nostro `PetriNetModel` e la traduce in un file `.pnml`. Durante questo processo, ogni `Place`, `Transition` e `Arc` del modello logico viene convertito nel suo corrispondente elemento XML (`<place>`, `<transition>`, `<arc>`), completo di ID e coordinate grafiche.
- **PNMLParser:** Questa classe gestisce il processo inverso, ovvero la lettura. Analizza un file `.pnml`, ne estrae gli elementi e utilizza un `PetriNetBuilder` per ricostruire l’oggetto `PetriNetModel` corrispondente in memoria, consentendo così il caricamento di reti preesistenti.

2.3.3 Estensione dello Standard per Attributi Personalizzati

Una delle necessità specifiche del nostro progetto era quella di distinguere le transizioni in base al ruolo dell’utente che può attivarle: `USER` o `ADMIN`. Poiché lo schema PNML di base non prevede un attributo nativo per questa distinzione, abbiamo esteso la struttura standard sfruttando la sua flessibilità.

In fase di serializzazione, abbiamo aggiunto un attributo personalizzato chiamato `type` all’elemento `<transition>`. Durante il salvataggio, il valore di questo attributo viene impostato in base al `TRANSITION_TYPE` della transizione nel nostro modello.

Ecco un esempio di come appare un frammento del file PNML generato dal nostro sistema:

```
1 <transition id="t1" type="ADMIN">
2   <name>
3     <text>t1</text>
4   </name>
5   <graphics>
6     <offset x="350.0" y="200.0"/>
7   </graphics>
8 </transition>
```

2.4 Elenco dei Design Pattern utilizzati

In questo progetto sono stati applicati i seguenti design pattern:

- **Singleton**: Utilizzato per i service di gestione delle notifiche e delle informazioni utente.
- **Facade**: Implementato nella classe `ViewNavigator` per semplificare la logica di navigazione.
- **Observer**: Presente nelle classi `NetVisualController` e `NetCreationController` per gestire le notifiche di cambiamento di stato.
- **Builder**: Impiegato per la creazione sequenziale di oggetti quali notifiche e reti.
- **Data Access Object (DAO)**: Adottato in tutte le classi DAO per l'accesso e la manipolazione dei dati persistenti.
- **State**: questo pattern è utilizzato in `ShowAllController` e in `NetVisualController`; modifica il comportamento del controller in base al valore dell'`enum` assegnato, isolando i comportamenti specifici di ciascuno stato in classi dedicate e riducendo la complessità delle strutture condizionali.

Per quanto riguarda l'architettura dell'applicazione, è stato utilizzato il pattern **Model–View–Controller (MVC)**.

2.5 Diagrammi delle classi

Di seguito vengono mostrati i diagrammi UML più rilevanti relativi alla struttura dell'applicazione. Ogni diagramma evidenzia una specifica area funzionale o architetturale del sistema.

2.5.1 Database

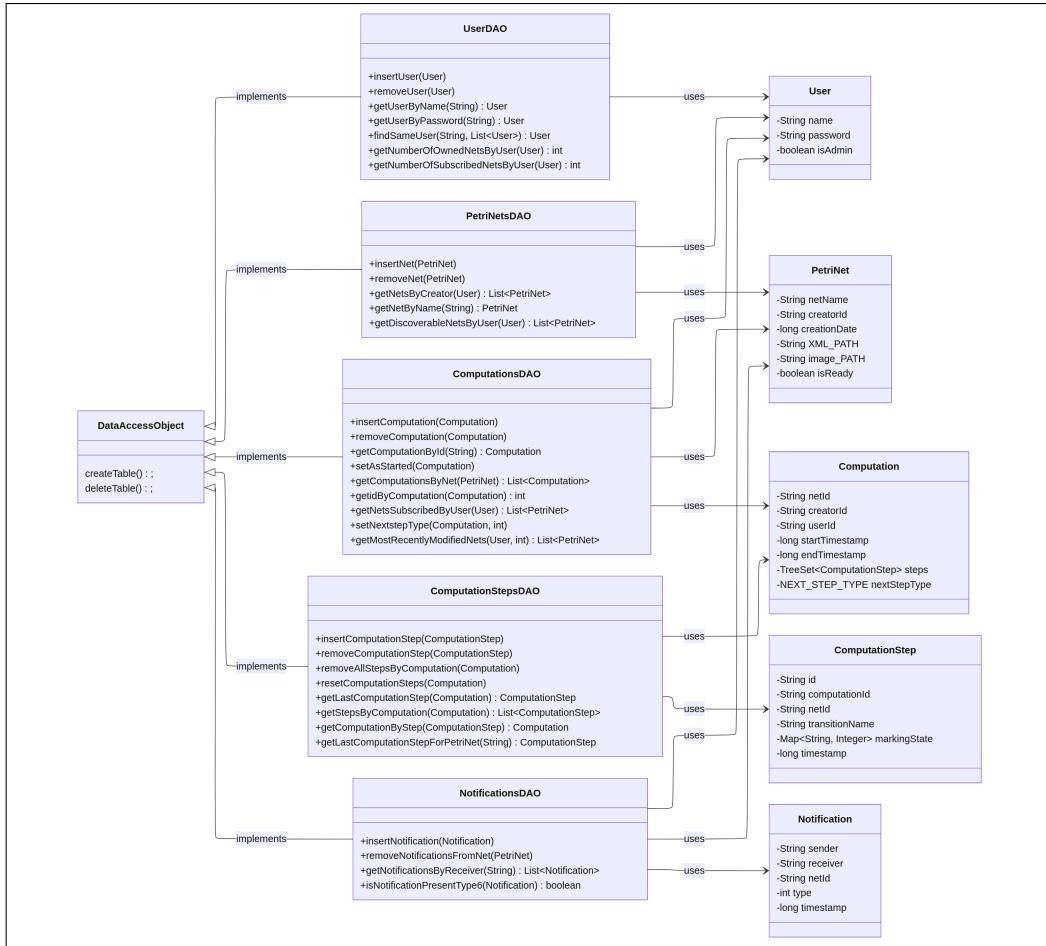


Figure 5: Diagramma delle classi del livello DAO (Data Access Object)

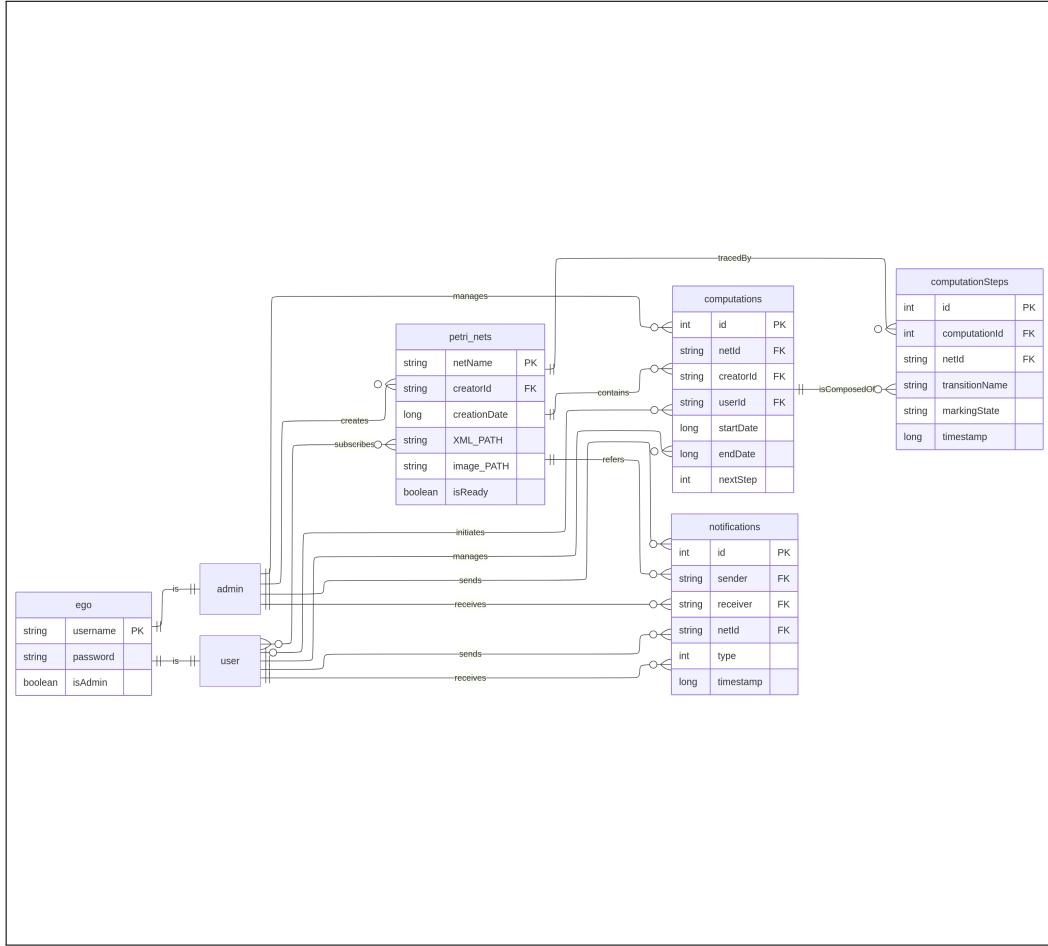


Figure 6: Diagramma delle classi relative alla gestione del database (Entity-Relationship)

2.5.2 Petri net

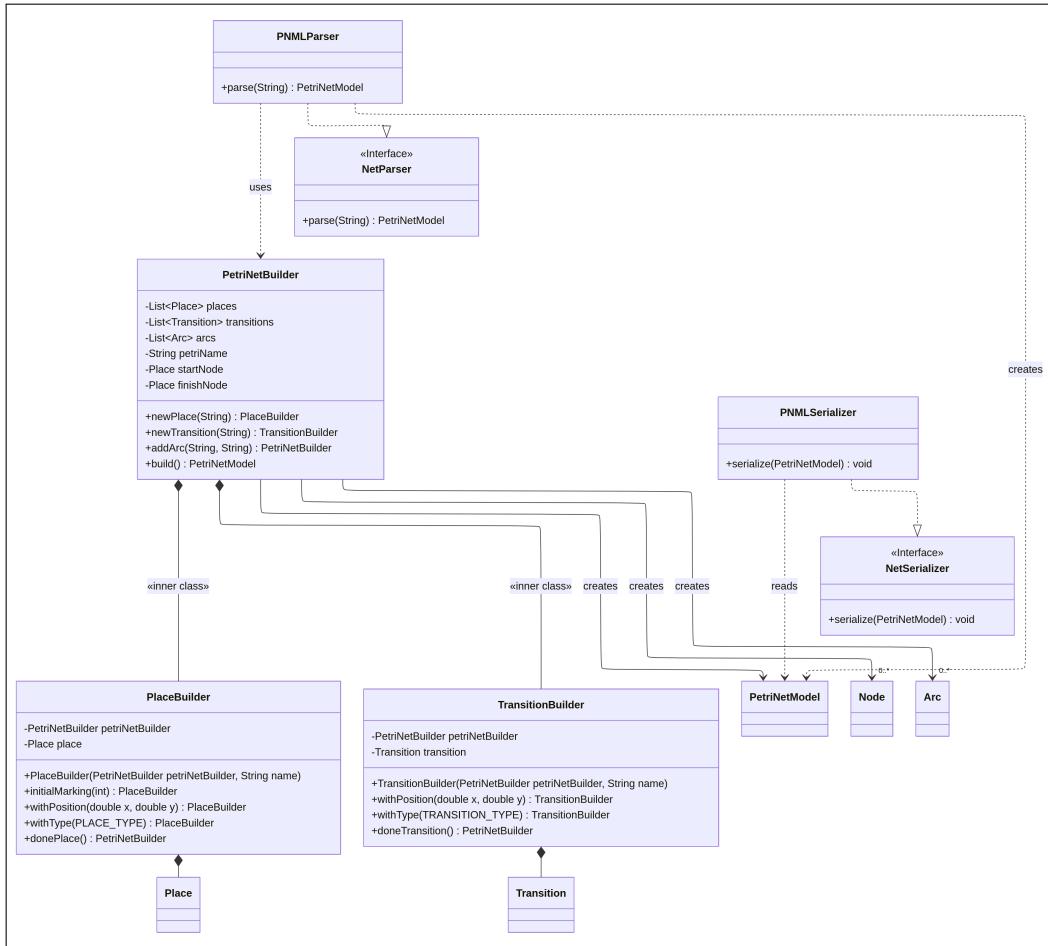


Figure 7: Diagramma delle classi relative al builder, alla serializzazione e al parser delle PN

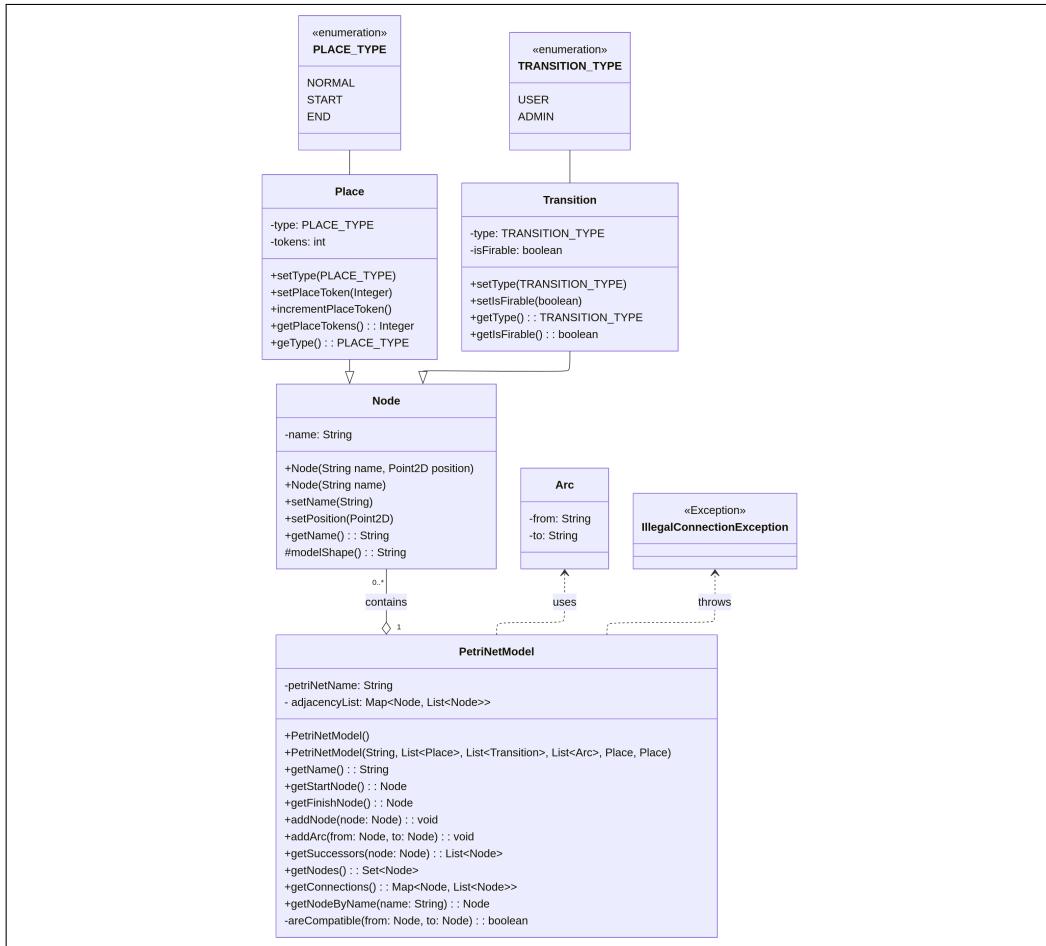


Figure 8: Diagramma delle classi del modello delle PN

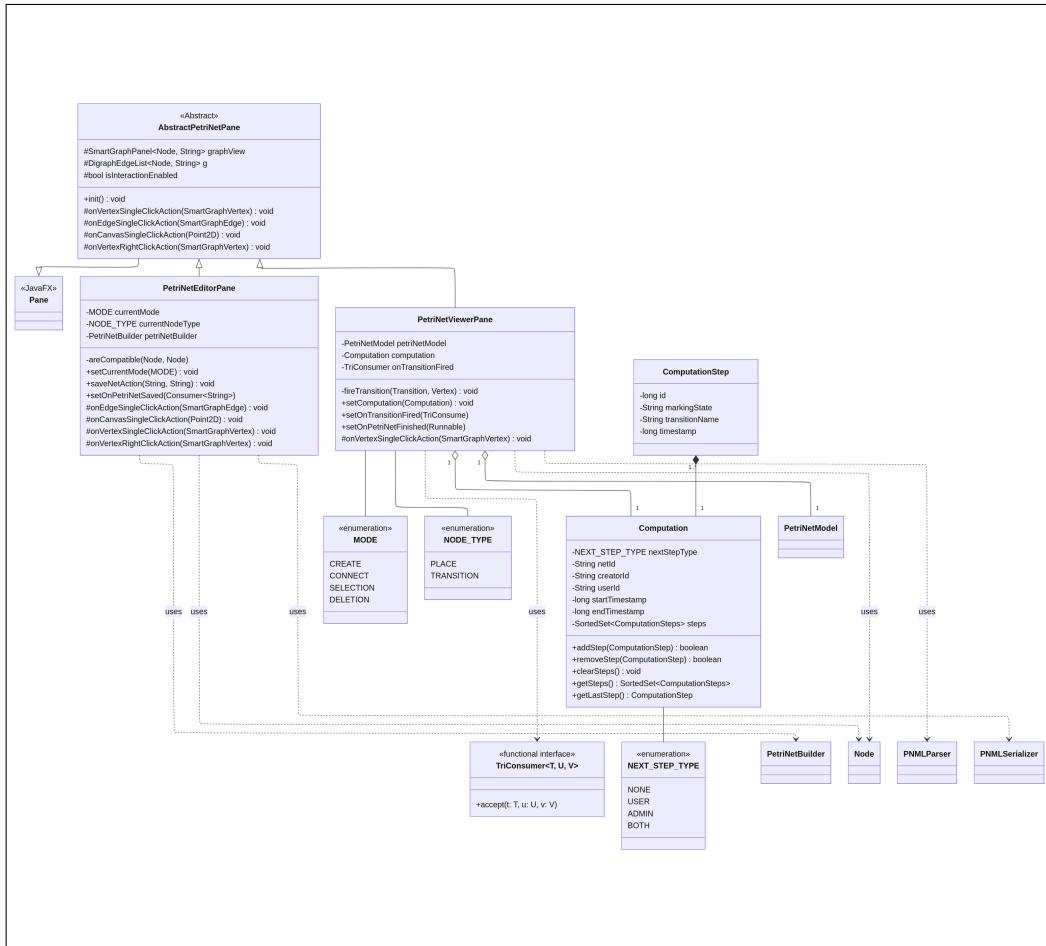


Figure 9: Diagramma delle classi relative alla vista delle PN

2.5.3 Tabelle

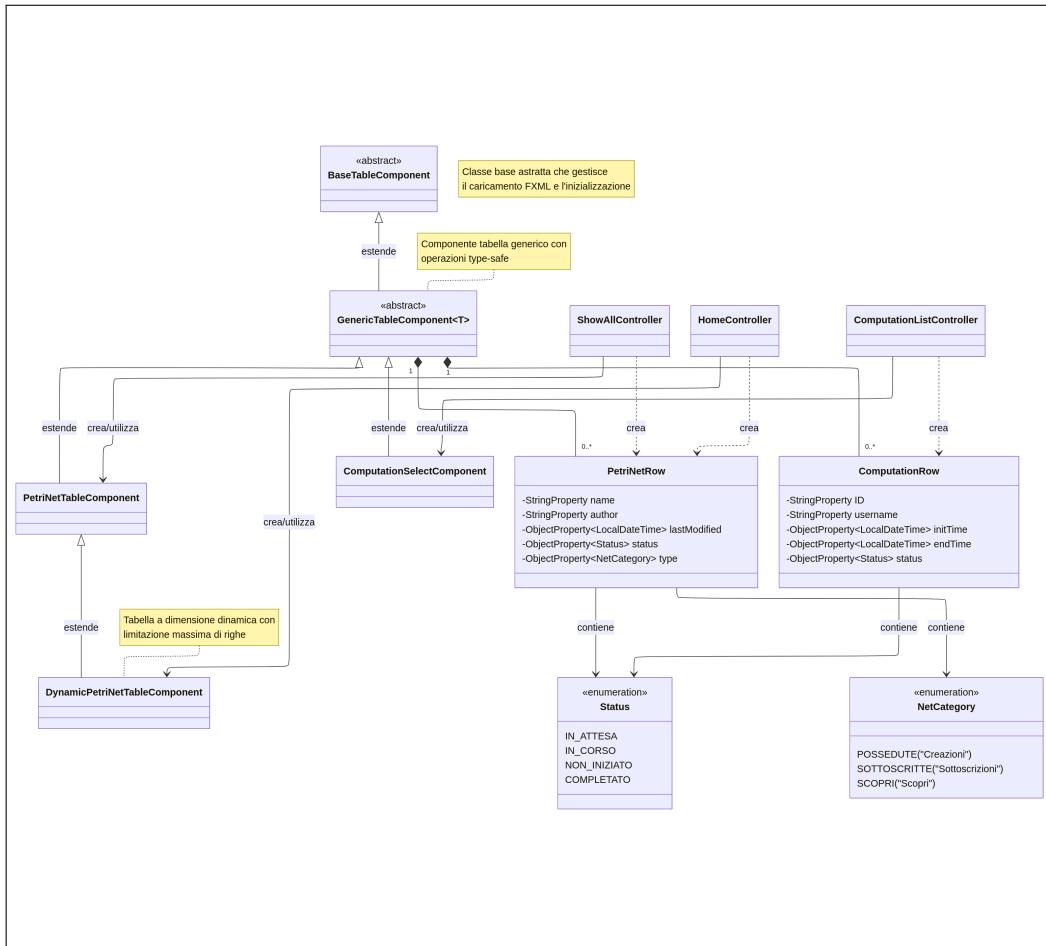


Figure 10: Diagramma delle classi relative alle tabelle per la visualizzazione delle reti

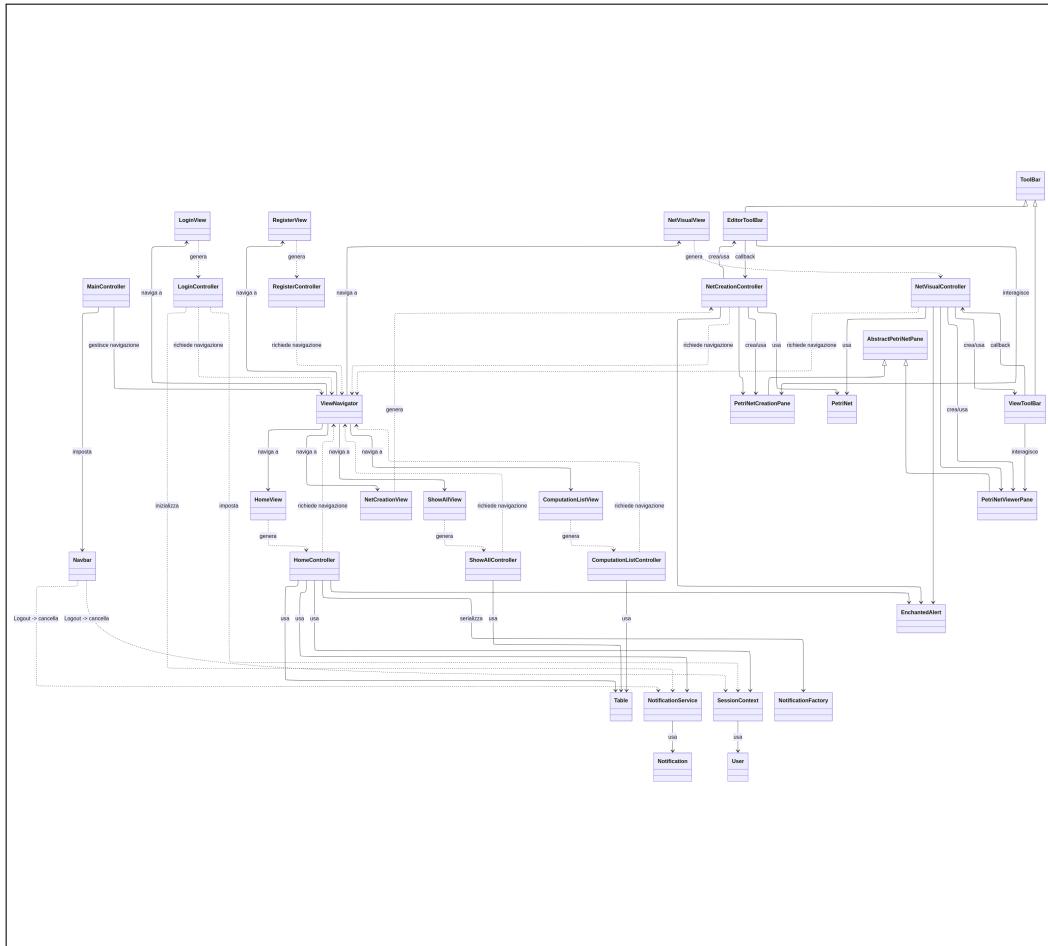


Figure 11: Diagramma complessivo delle iterazioni tra le classi principali dell'applicazione

2.6 Diagrammi delle sequenze

Di seguito sono riportati i diagrammi delle sequenze che illustrano i flussi di interazione tra gli attori e il sistema per le operazioni principali.

2.6.1 Petri net

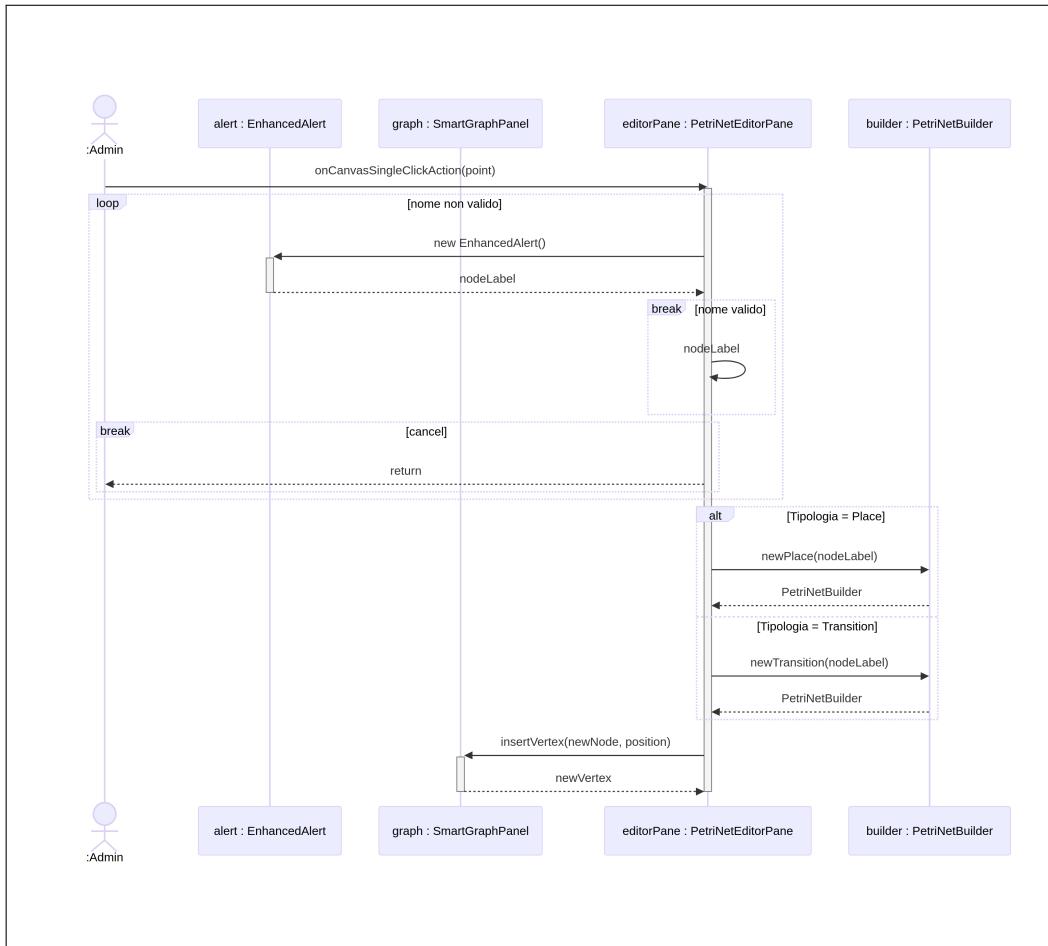


Figure 12: Diagramma di sequenza per la gestione del click sul canvas del editor delle Petri Net

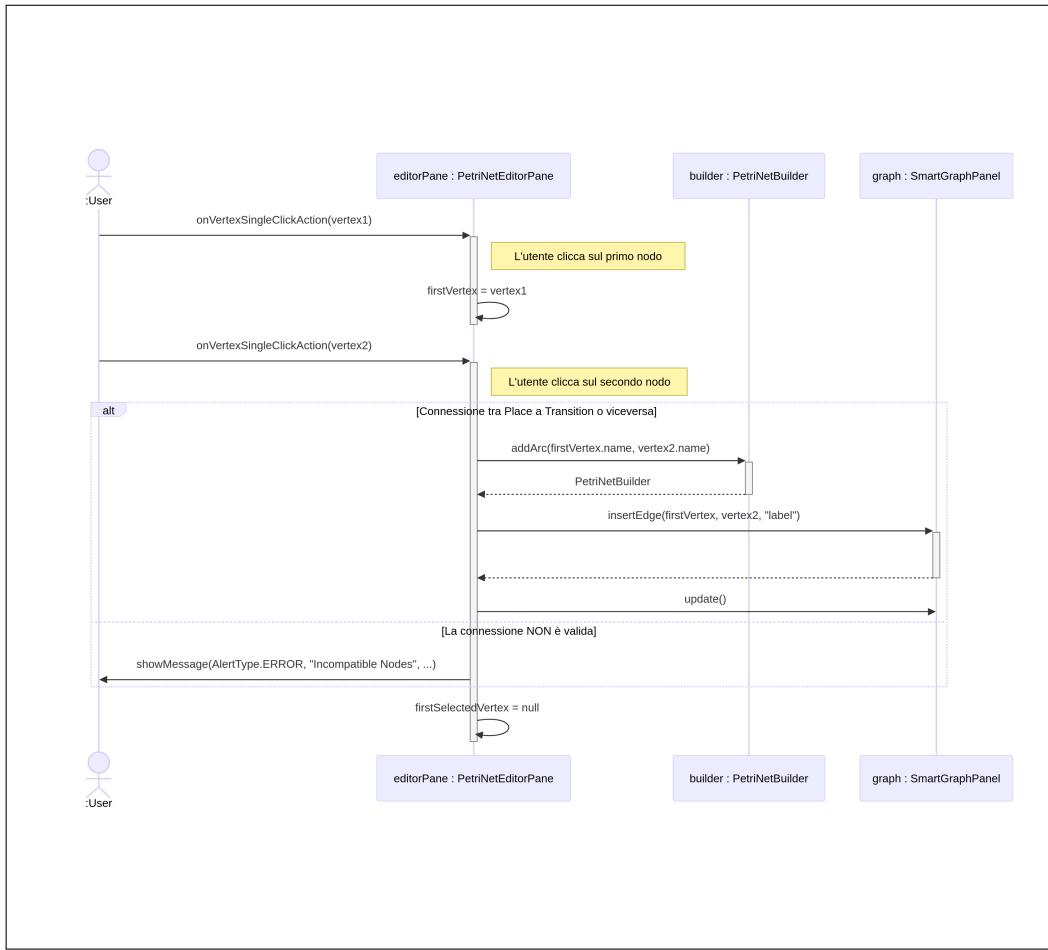


Figure 13: Diagramma di sequenza per la gestione della creazione di una connessione

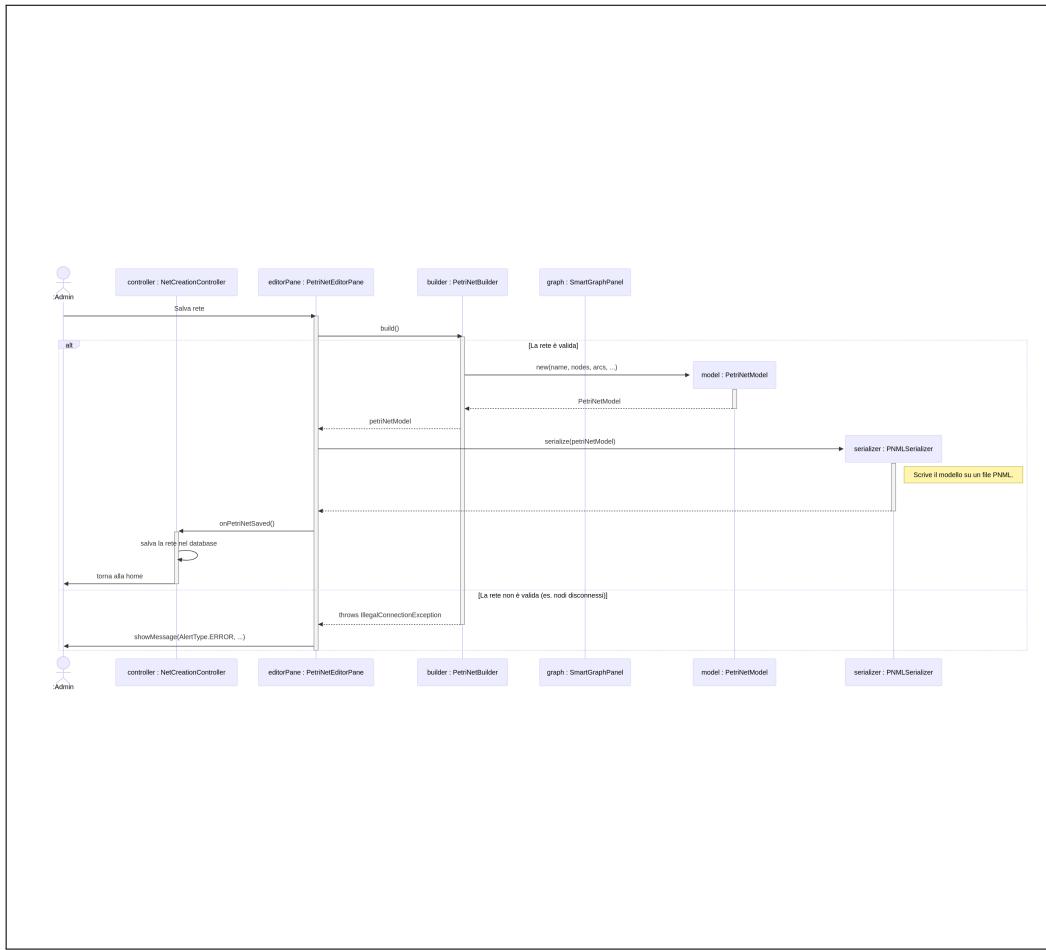


Figure 14: Diagramma di sequenza per la gestione del salvataggio di una Petri Net

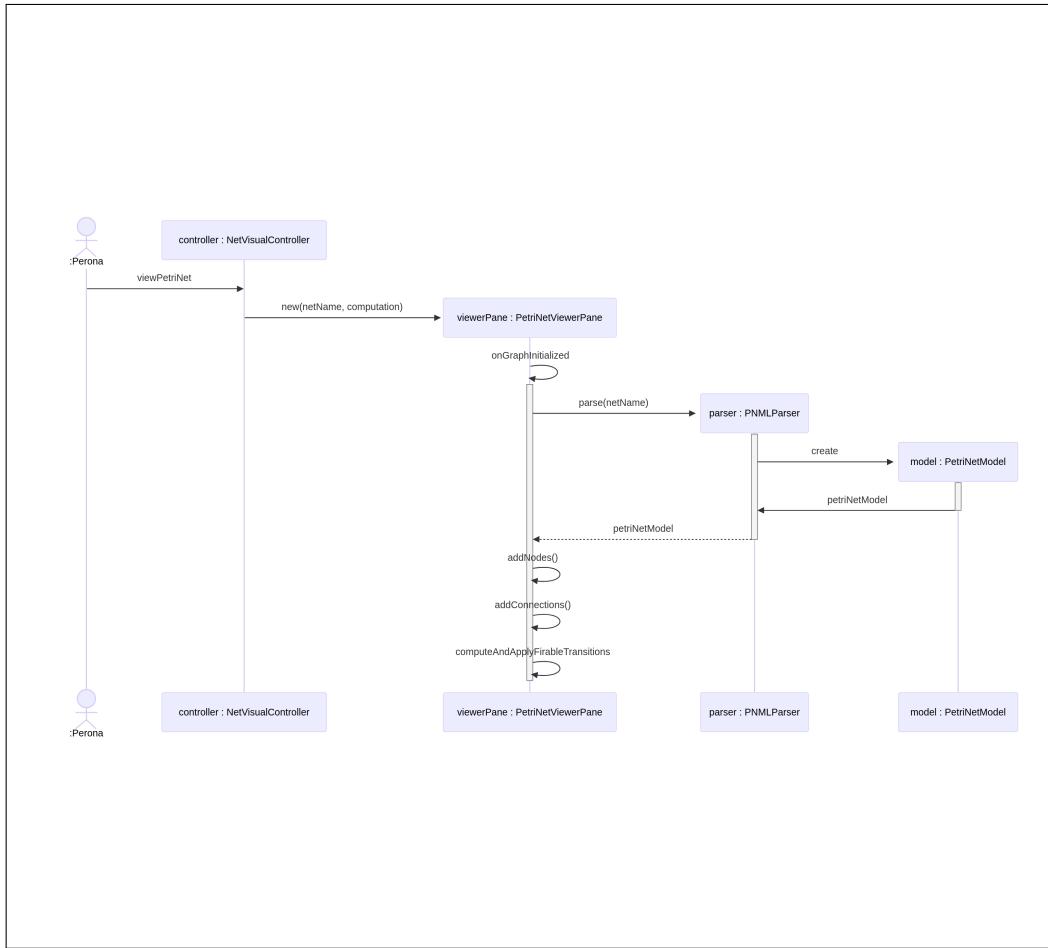


Figure 15: Diagramma di sequenza per la visualizzazione di una Petri Net

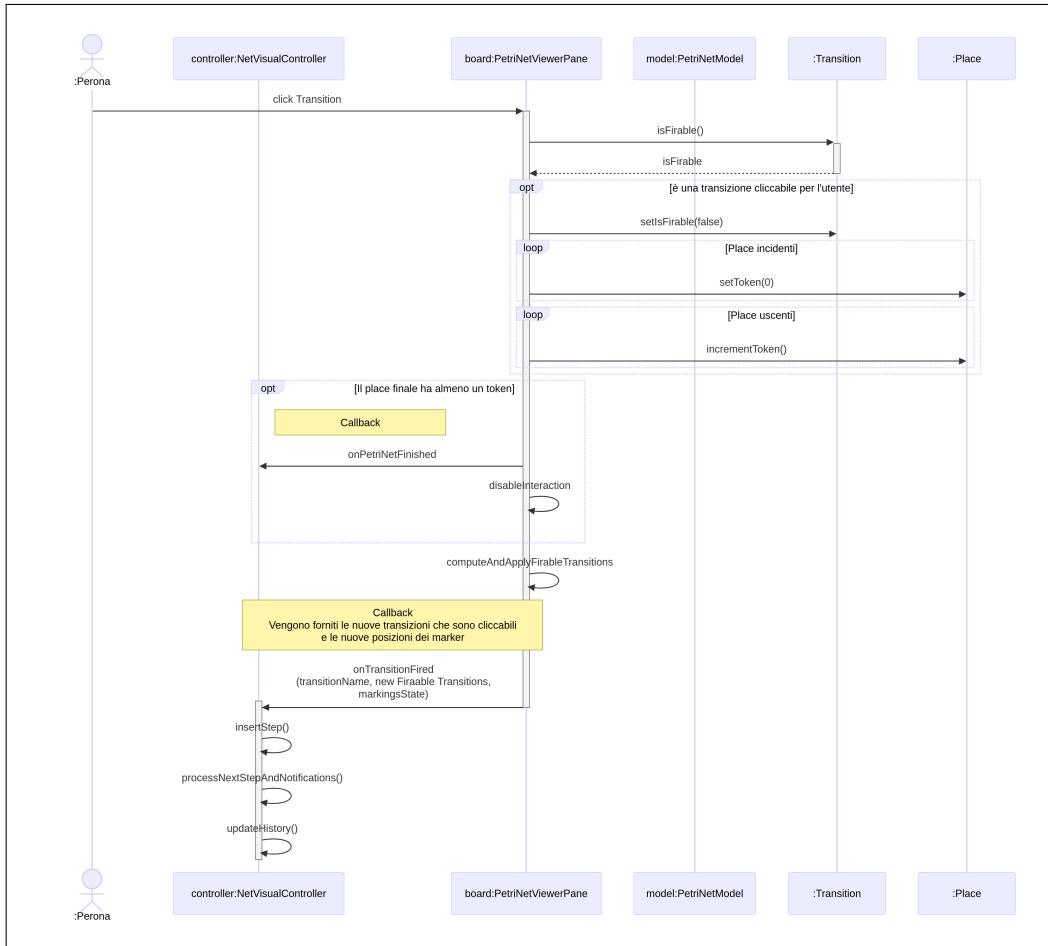


Figure 16: Diagramma di sequenza per la gestione del click su una transizione

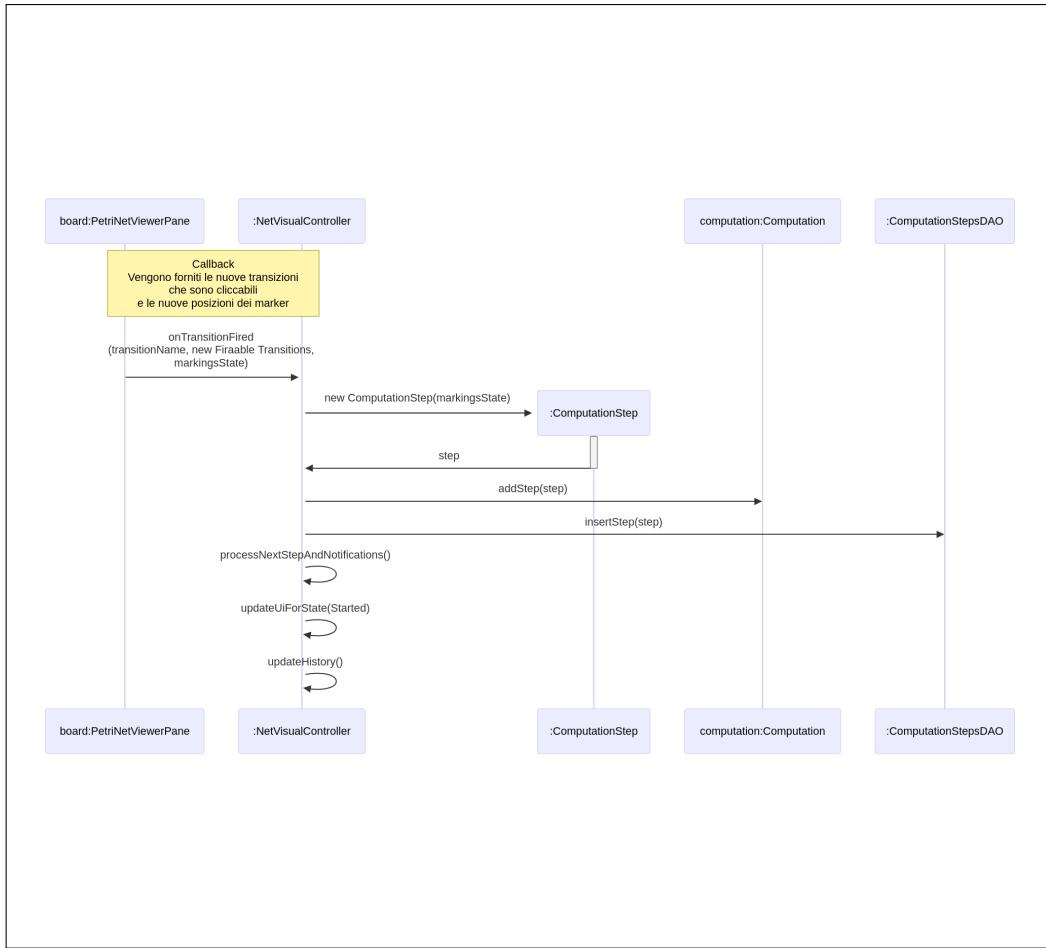


Figure 17: Diagramma di sequenza per il callback di una transizione cliccata

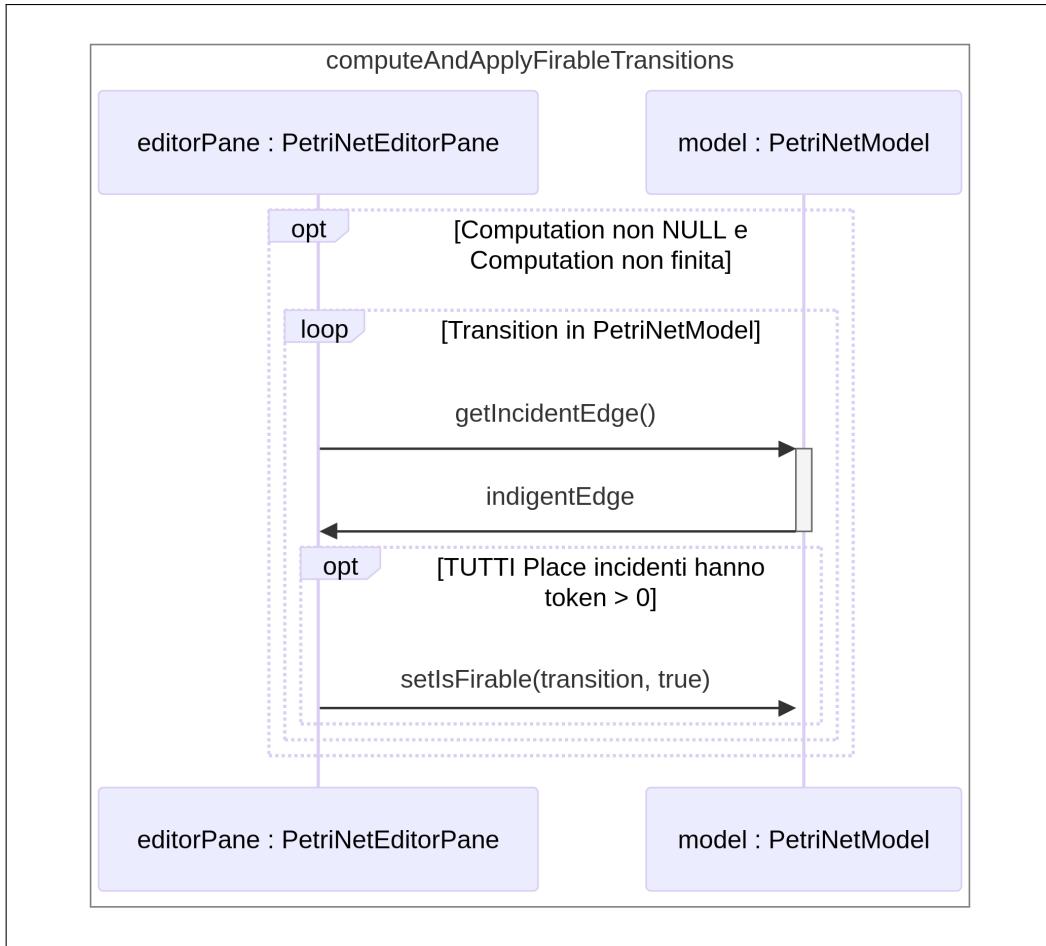


Figure 18: Diagramma di sequenza per determinare le transizioni "Firable"

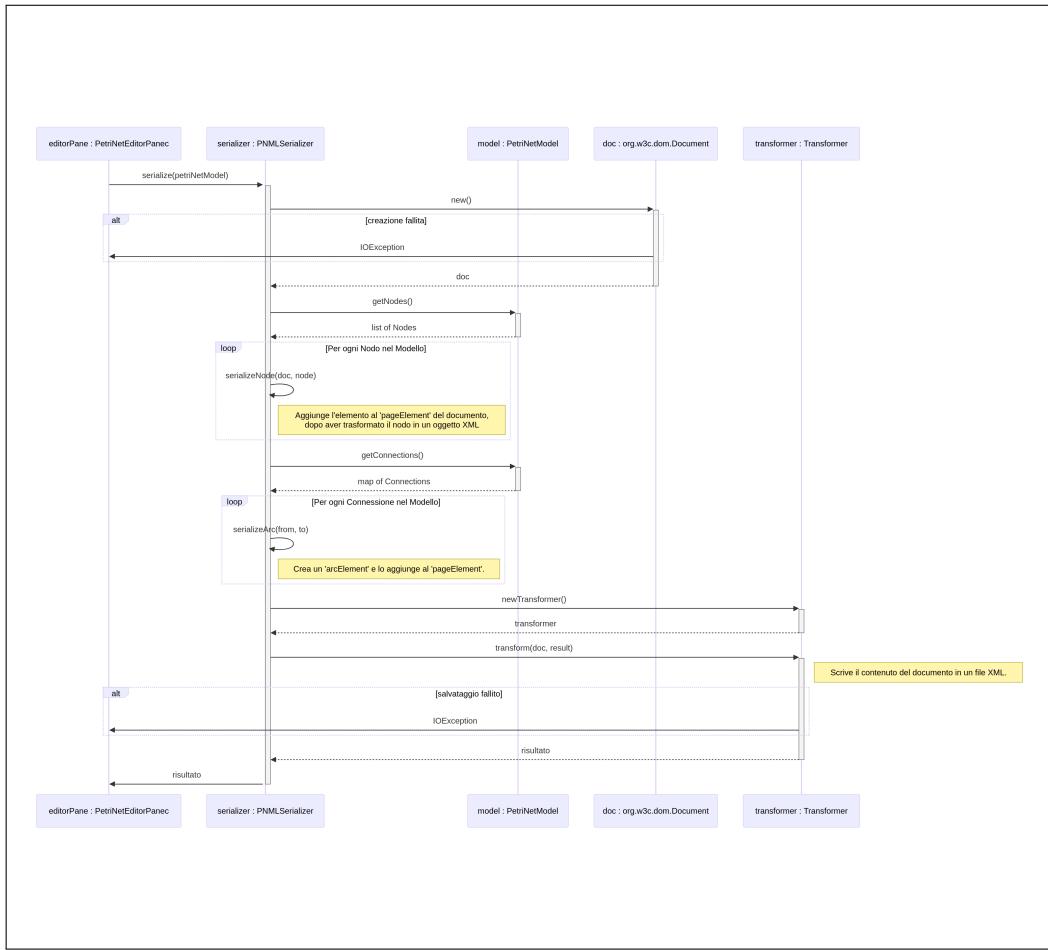


Figure 19: Diagramma di sequenza per la serializzazione di una Petri Net in formato PNML

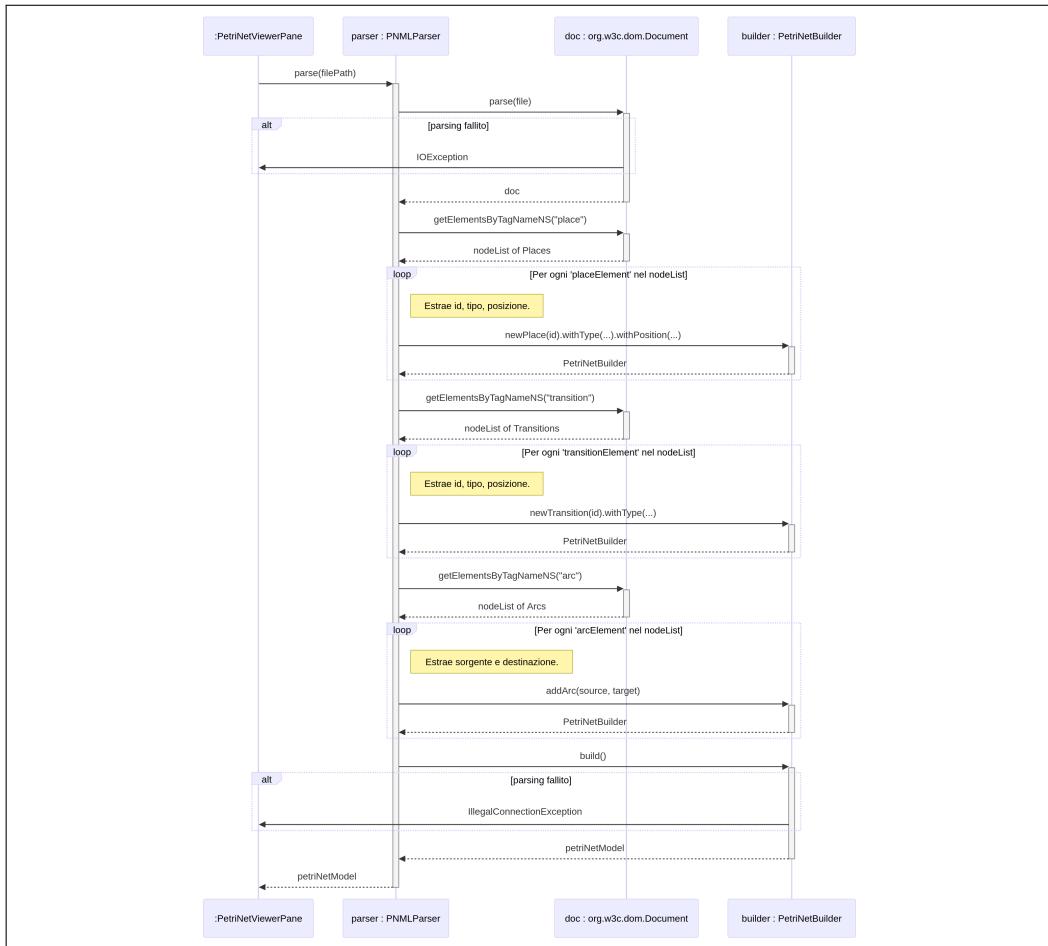


Figure 20: Diagramma di sequenza per la deserializzazione di una Petri Net da un file PNML

2.6.2 Gestione delle reti

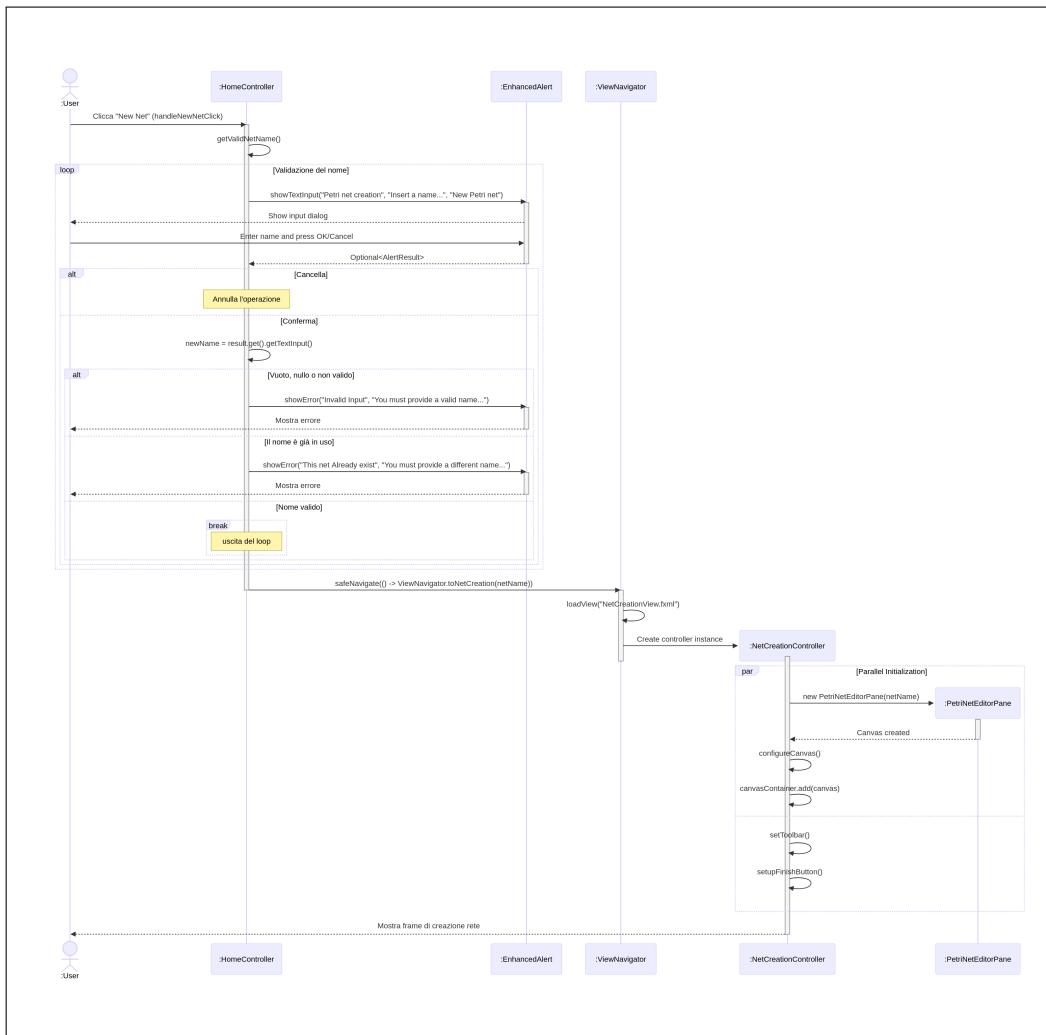


Figure 21: Diagramma di sequenza per la creazione di una nuova Petri Net

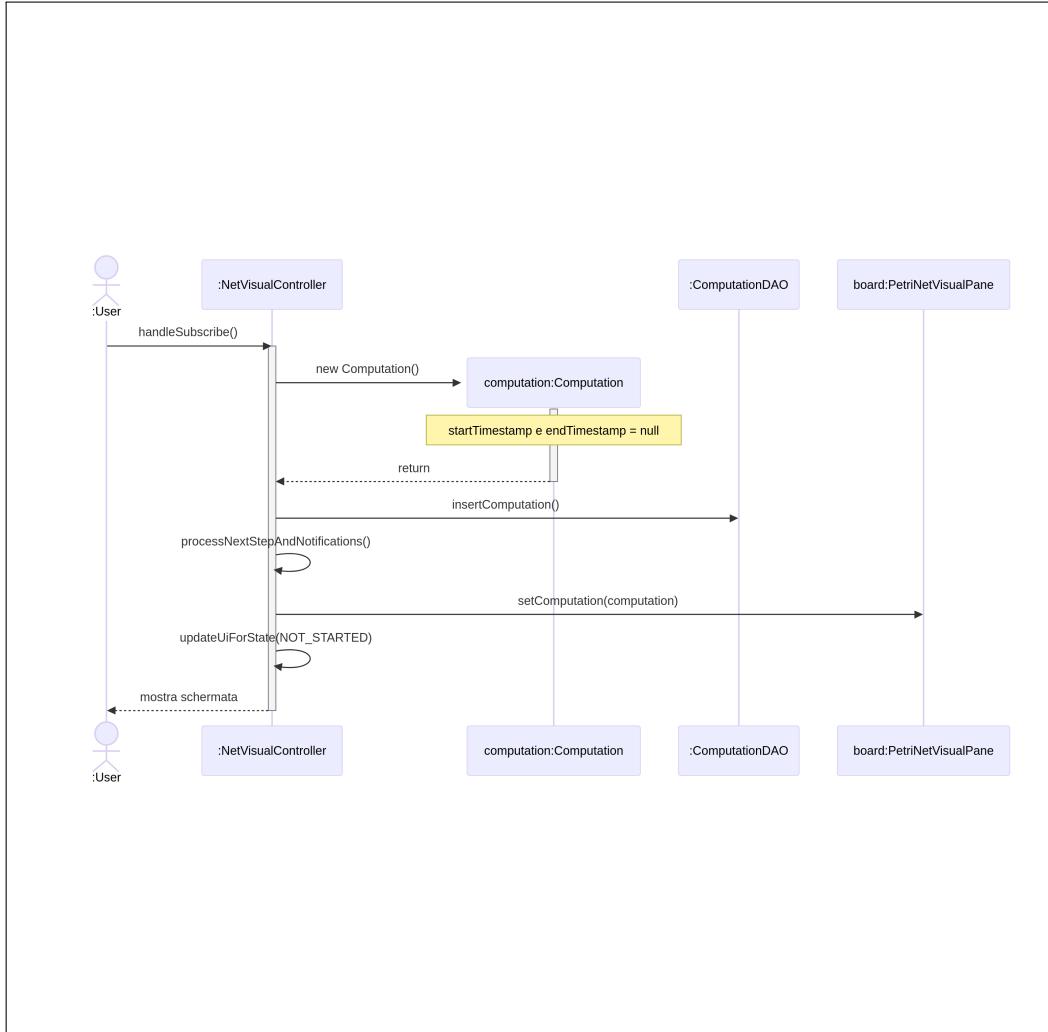


Figure 22: Diagramma di sequenza per la sottoscrizione a una Petri Net

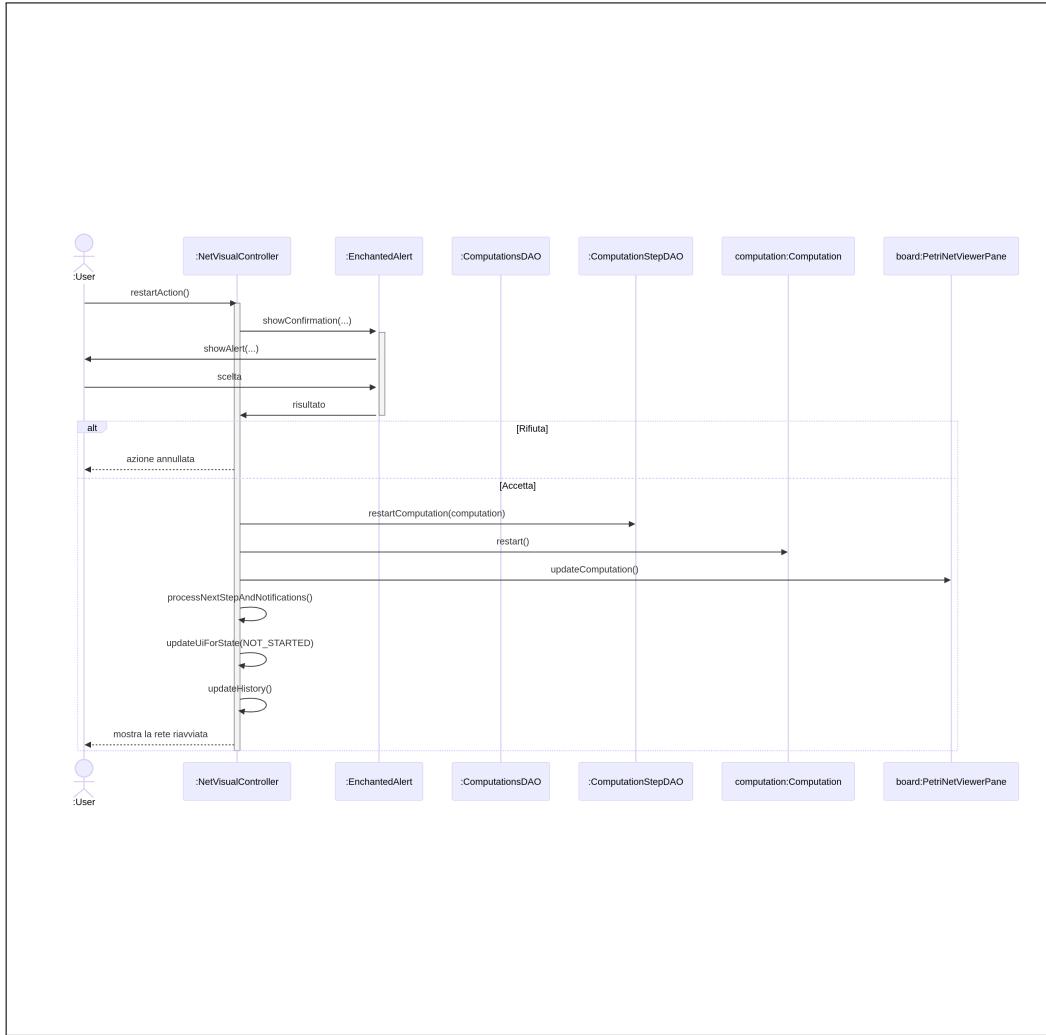


Figure 23: Diagramma di sequenza per il reset di una computazione per un Petri net

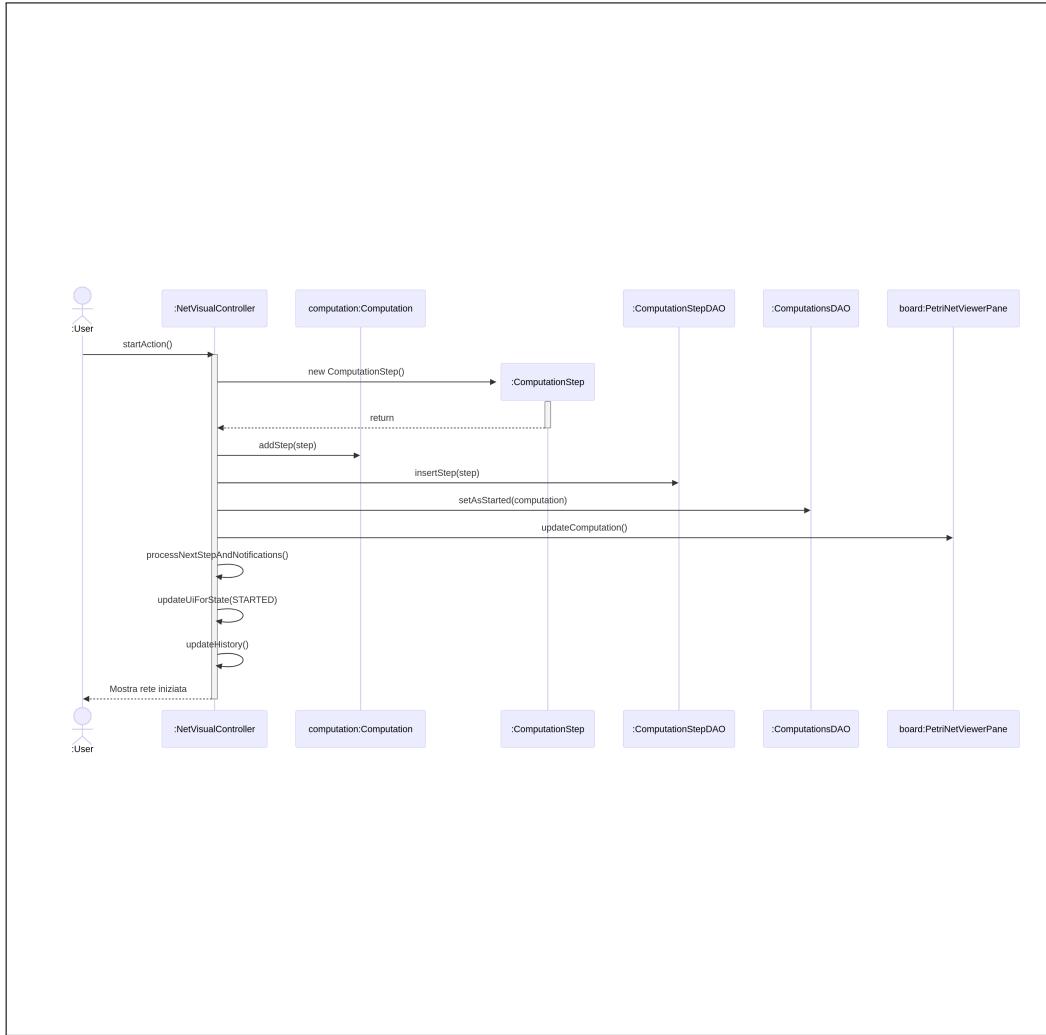


Figure 24: Diagramma di sequenza per l'avvio di una computazione per un Petri Net

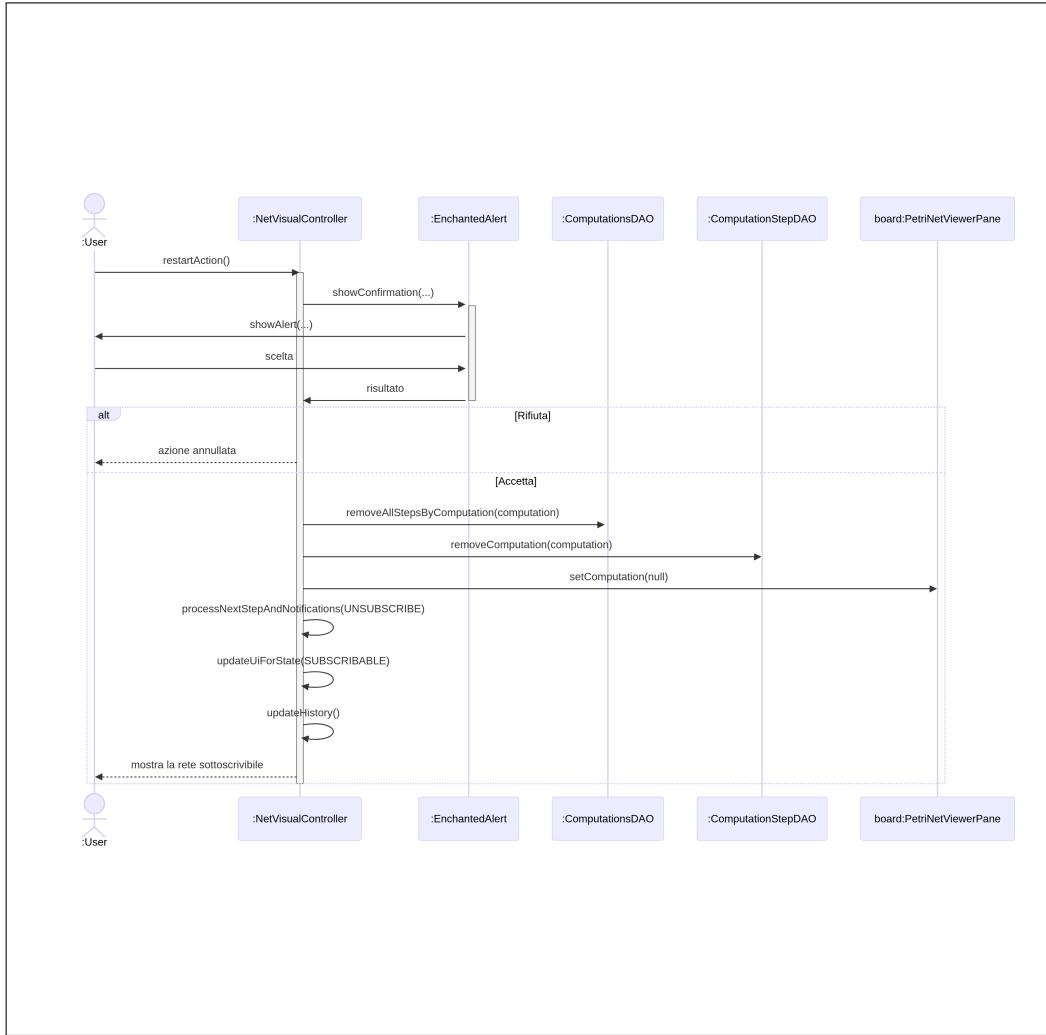


Figure 25: Diagramma di sequenza per la cancellazione della sottoscrizione a una Petri Net

2.6.3 Creazione delle Tabelle

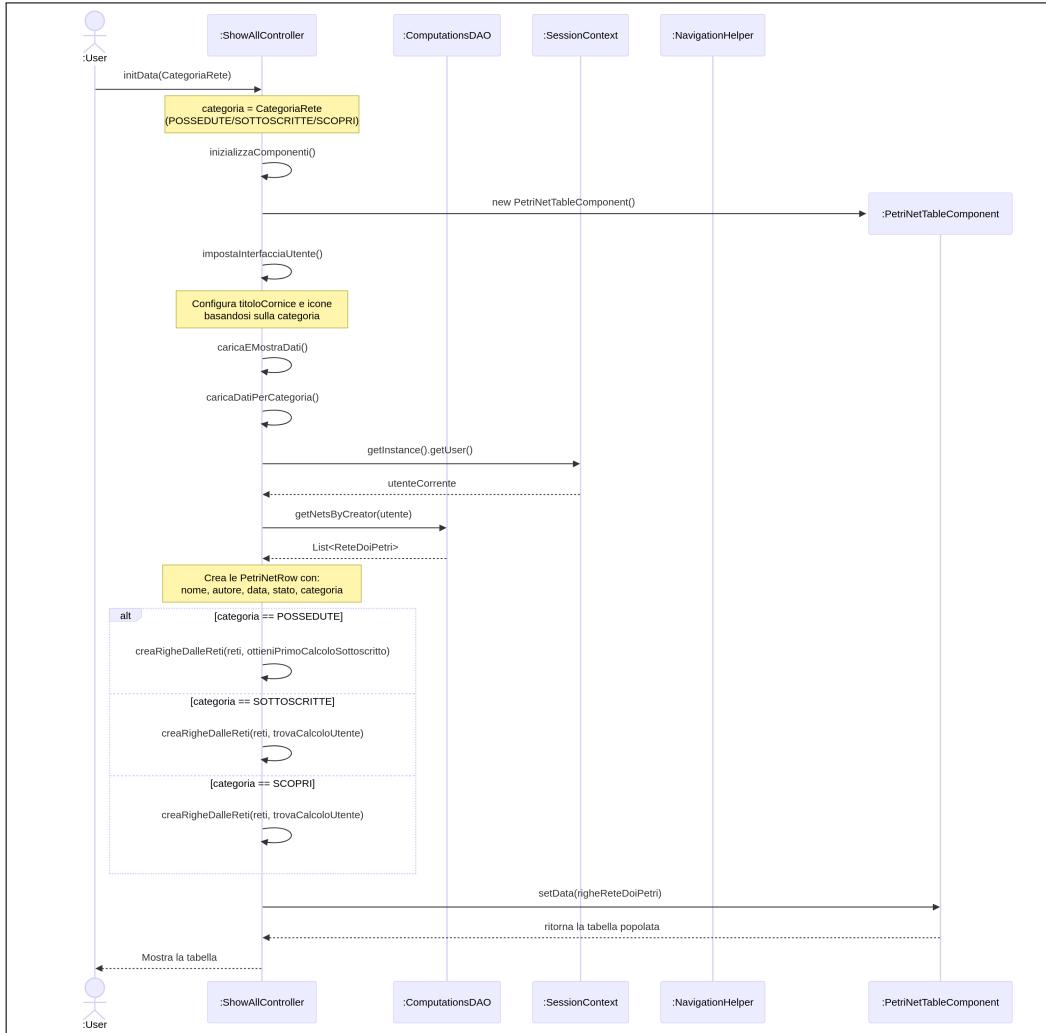


Figure 26: Diagramma di sequenza per la popolazione delle tabelle con le Petri Net

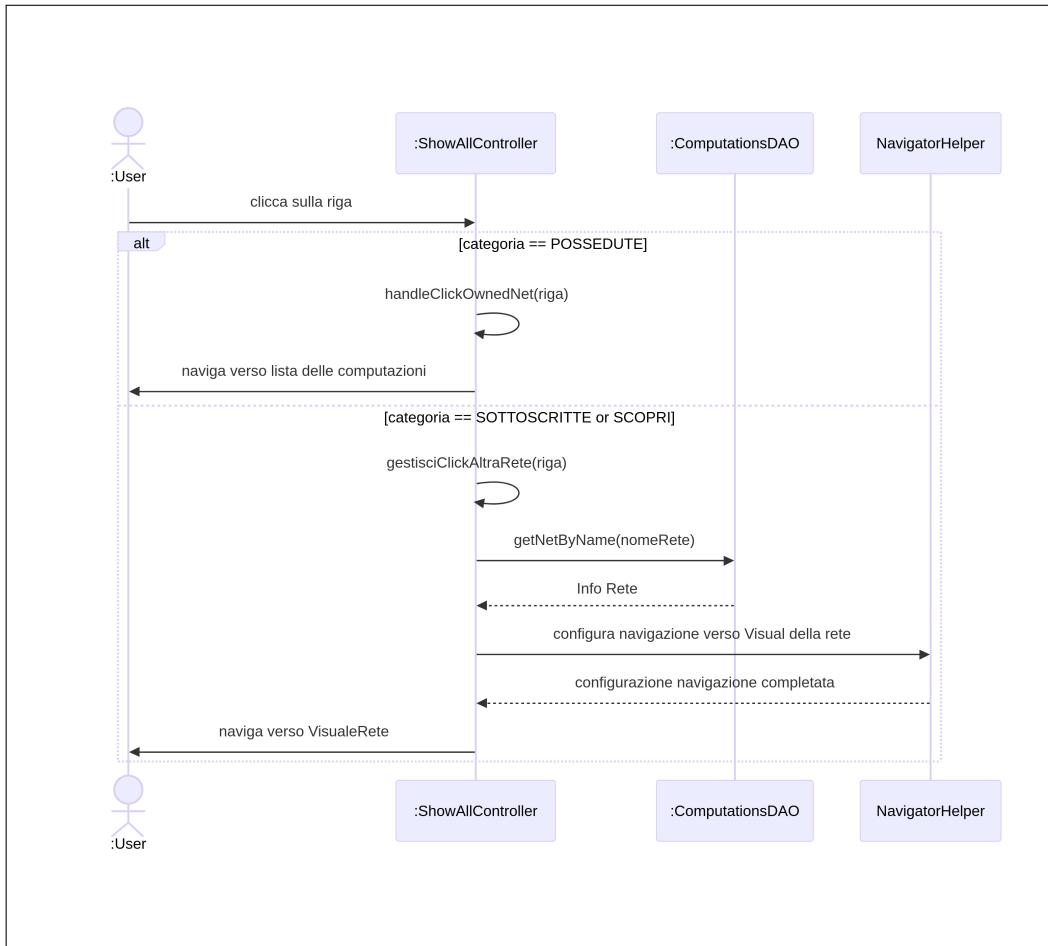


Figure 27: Diagramma di sequenza per la gestione del click su una riga della tabella delle Petri Net

3 Validazione e test

3.1 Test interni

Oltre ai test unitari automatizzati, è stata adottata una metodologia di peer testing per la validazione delle nuove funzionalità prima della loro integrazione nel branch di sviluppo principale. Questo processo ha fornito un livello aggiuntivo di controllo sul codice prodotto.

La procedura seguita era la seguente:

1. Una volta che un componente completava una *feature*, la rendeva disponibile agli altri membri del team per una revisione.
2. I revisori testavano la funzionalità seguendo un approccio di tipo **black-box**, ovvero senza ricevere indicazioni specifiche o "dritte" dal suo creatore. Questo metodo è stato scelto per simulare l'esperienza di un utente finale che si approccia alla feature per la prima volta, permettendo di valutarne l'intuitività e la facilità d'uso.
3. Le eventuali criticità, bug o problemi di usabilità riscontrati venivano riportati allo sviluppatore per la risoluzione.

Questo approccio qualitativo ha agito in modo complementare ai test unitari, consentendo di identificare non solo errori funzionali, ma anche incongruenze nell'esperienza utente che test automatici non avrebbero potuto rilevare.

3.2 Test esterni

Per valutare interfaccia utente, l'esperienza d'uso e l'intuitività generale dell'applicazione, è stata condotta una sessione di test.

3.2.1 Metodologia di Test

Per condurre i diversi test abbiamo deciso di chiedere a due categorie di utenze distinte:

- **Utenti Esperti:** composto da studenti con un background informatico, come ad esempio nostri colleghi di corso.

- **Utenti Novizi:** composto da persone esterne all’ambito informatico, rappresentative di un utente generico.

Data la natura teorica delle reti di Petri, a entrambi i gruppi è stata fornita una descrizione preliminare che ne descriveva i concetti base, i vincoli formali e alcuni possibili campi di applicazione. L’obiettivo era misurare quanto l’applicazione fosse in grado di guidare l’utente nella comprensione e nell’utilizzo del modello.

3.2.2 Risultati Emersi

Il feedback raccolto è stato eterogeneo e ha permesso di identificare con chiarezza sia i punti di forza del design implementato sia le aree che necessiterebbero di ulteriori miglioramenti.

3.2.2.1 Aree di Miglioramento

- **Intuitività del obiettivo:** La critica principale emersa, comune a entrambi i gruppi, riguarda la difficoltà nel contestualizzare l’uso del software in uno scenario applicativo reale. L’astrattezza del modello delle reti di Petri, se non legata a un esempio concreto, rappresenta una barriera all’utilizzo intuitivo e creativo dello strumento.
- **Curva di Apprendimento per Utenti non Esperti:** Gli utenti del gruppo novizio hanno riscontrato difficoltà nel comprendere i passaggi necessari per comporre una rete di Petri sintatticamente valida e completa. Questo suggerisce che l’interfaccia, pur essendo chiara nella navigazione, non fornisce sufficiente supporto concettuale. È stato quindi suggerito di integrare una sezione di tutorial interattivo o delle schede informative accessibili dall’applicazione per guidare i nuovi utenti.

3.2.2.2 Punti di Forza

- **Design dell’Interfaccia ed usabilità:** È stato espresso un forte apprezzamento per il design dell’applicazione, definito moderno, pulito e intuitivo. La navigazione tra le diverse schermate è risultata fluida e prevedibile. Le azioni, come la selezione delle diverse viste delle reti (personalni, sottoscritte, da scoprire), sono state giudicate rapide e ben organizzate.

- **Chiarezza delle Informazioni di Stato:** Un elemento particolarmente apprezzato è stato l'uso di "tag" o "badge" colorati per visualizzare lo stato delle computazioni (es. *Completed*, *In Progress*, *Waiting*). Questa scelta di design è stata ritenuta molto efficace, in quanto fornisce un feedback visivo immediato e comprensibile a colpo d'occhio.

3.3 Unit Testing

La validazione del sistema è stata condotta attraverso una strategia di Unit Testing per assicurare la correttezza funzionale e la robustezza delle componenti logiche del backend. Per l'implementazione dei test è stato utilizzato il framework JUnit 5, che ha permesso di verificare in modo sistematico e isolato il comportamento delle singole classi.

Nello specifico abbiamo testato le sezioni critiche che riguardano i modelli delle reti di petri e la loro creazione. Componenti non strettamente legati alla parte grafica che quindi possono essere testati in maniera singolo ad indipendente dal resto. Nello specifico abbiamo verificato la funzionalità di:

- Modello della rete di petri che tramite il costruttore, che necessita di place, transition e connessioni, ne verifica la validità. Quindi provando a passargli:
 - una composizione valida e non aspettarci errori
 - una composizione con connessione Place →Place ed aspettarci un IllegalConnectionException
 - una composizione con una connessione dove il nodo di provenienza non era presente nella lista dei nodi ed aspettarci un IllegalConnectionException
 - una composizione senza nodi di start/finish
- Builder della rete, usato ad esempio anche nella creazione interattiva, e vedere se il metodo build() viene eseguito con successo.
 - Test con composizione valida
 - Test nell'eliminare un nodo e verificare che non sia presente dopo il build()
 - Test costruzione rete senza start/finish
 - Test con connessione non valida