404NotFound

Premi: better than Prezi.

Specifica Tecnica

Versione	1.0
Redazione	Vegro Federico Cossu Mattia Camborata Marco Manuto Monica Rettore Andrea Gobbo Ismaele De Lazzari Enrico
Verifica	Manuto Monica Rettore Andrea
Responsabile	Gobbo Ismaele
Uso	Esterno
Stato	Formale
Ultima modifica	XXX
Lista di distribuzione	404NotFound prof. Tullio Vardanega prof. Riccardo Cardin Zucchetti S.p.a.

Registro delle modifiche

Versione	Autore	Data	Descrizione
1.0	Autore vers 1.0	12-01-2015	Scrittura versione finale ecc.
0.1	Autore vers 0.1	19-12-2014	Stesura scheletro ecc.

Tabella 1: Storico versioni del documento.

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento definisce la progettazione ad alto livello di Premi. Viene prima descritta la struttura generale del sistema e successivamente vengono analizzate le varie componenti software in relazione alle loro attività principali. Segue poi la descrizione delle tecnologie e dei Design Pattern$_G$ utilizzati, e un mockup$_G$ dell'interfaccia grafica lato utente.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un software di presentazione di slide non basato sul modello di PowerPoint$_G$, sviluppato in tecnologia HTML5$_G$ e che funzioni sia su desktop che su dispositivo mobile. Il software dovrà permettere la creazione da parte dell'autore e la successiva presentazione del lavoro, fornendo effetti grafici di supporto allo storytelling e alla creazione di mappe mentali.

1.3 Glossario

Al fine di evitare ogni ambiguità relativa al linguaggio e ai termini utilizzati nei documenti formali tutti i termini e gli acronimi presenti nel seguente documento che necessitano di definizione saranno seguiti da una "G" in pedice e saranno riportati in un documento esterno denominato Glossario.pdf. Tale documento accompagna e completa il presente e consiste in un listato ordinato di termini e acronimi con le rispettive definizioni e spiegazioni.

2 Tecnologie utilizzate

2.1 JavaScript

JavaScript$_G$ è un linguaggio di scripting lato client$_G$ orientato agli oggetti, comunemente usato nei siti web, ed interpretato dai browser$_G$. Ciò permette di alleggerire il server$_G$ dal peso della computazione, che viene eseguita dal client$_G$. Questo è un aspetto molto importante per lo sviluppo del capitolato Premi. La caratteristica principale di JavaScript$_G$ 'e, appunto, quella di essere un linguaggio interpretato: il codice non viene compilato, ma interpretato, dal browser$_G$. Essendo molto popolare e ormai consolidato, JavaScript$_G$ può essere eseguito dalla maggior parte dei browser$_G$, sia desktop che mobile, grazie anche alla sua leggerezza. Uno degli svantaggi di questo linguaggio è che ogni operazione che richieda informazioni che devono essere recuperate da un database$_G$ deve passare attraverso un linguaggio che effettui esplicitamente la transazione, per poi restituire i risultati a JavaScript$_G$. Tale operazione richiede l'aggiornamento totale della pagina, ma grazie ad Meteor è possibile superare questo limite.

2.2 HTML5

HTML5$_G$ verrà utilizzato per definire la struttura dell'applicazione web Premi. Tale struttura sarà completamente separata dalla presentazione, che verrà realizzata tramite CSS3$_G$. HTML5$_G$ presenta, rispetto ad HTML—$_G$ 4, diversi vantaggi per lo svolgimento del progetto:

- Introduzione di elementi di controllo per i menu di navigazione (tag "nav");
- Introduzione di elementi specifici per l'inserimento di contenuti multimediali (tag "video" e "audio")

] e molti altri.

2.3 CSS3

CSS$_G$ (Cascading Style Sheet) è un linguaggio pensato con lo scopo di definire l'aspetto di pagine HTML$_G$ e non solo, che devono presentare un collegamento al loro foglio di stile nell'header (la parte del documento HTML$_G$ che introduce un gruppo di ausili introduttivi o di navigazione). Grazie ai CSS$_G$, 'e possibile una completa separazione tra la presentazione (cioè l'aspetto grafico delle pagine web) ed i contenuti delle pagine stesse. Ciò semplifica la comprensione, la manutenzione e la portabilita'. Rispetto a CSS2, CSS3$_G$ introduce funzionalità grafiche più avanzate.

2.4 Angular

Nello sviluppo software AngularJS (più comunemente noto come "Angular") è un framework$_G$ per applicazioni web open-source$_G$ gestito da Google e da una comunità di singoli sviluppatori e aziende per affrontare molte delle sfide incontrate nello sviluppo di applicazioni una sola pagina. Angular mira a semplificare lo sviluppo e la sperimentazione di tali applicazioni , fornendo un quadro di riferimento per l'architettura Model-View-Controller (MVC) lato client$_G$, insieme ai componenti comunemente utilizzati in applicazioni.

Le librerie di AngularJS funzionano leggendo prima la pagina HTML$_G$, che ha incorporati in essa tag attributo personalizzati aggiuntivi. Angular interpreta quegli attributi come direttive per legare parti della pagina (in ingresso o in uscita) a un modello che è rappresentato da variabili JavaScript$_G$ standard. I valori di tali variabili JavaScript$_G$ possono essere impostati manualmente all'interno del codice, o recuperati da risorse JSON$_G$ statiche o dinamiche.

AngularJS è costruito attorno alla convinzione che la programmazione dichiarativa deve essere utilizzata per la costruzione di interfacce utente e il collegamento dei componenti software, mentre la programmazione imperativa è più adatta per definire la logica di business di un'applicazione. Il framework$_G$ adatta ed estende il tradizionale HTML$_G$ per presentare contenuti dinamici attraverso il "Two-Way Data Binding" che consente la sincronizzazione automatica di modelli e viste, con il risultato di migliorare la testabilità e le prestazioni.

I nostri obiettivi nella scelta di Angular:

- Disaccoppiare manipolazione del DOM$_G$ dalla logica dell'applicazione.
  La difficoltà di questo è notevolmente influenzata dal modo in cui il codice è strutturato.

- Disaccoppiare il lato client$_G$ di un'applicazione dal lato server$_G$.
  Questo permette allo sviluppo di progredire in parallelo, e permette il riutilizzo del codice di entrambe le parti.

- Fornire la struttura per il percorso di creazione di un'applicazione:
  dalla progettazione dell'interfaccia utente, attraverso la scrittura della logica, al collaudo.

- Angular implementa il pattern MVC$_G$ per separare la presentazione, i dati e le componenti logiche. Usando la Dependency Injection, che verrà descritta dettagliatamente più avanti, Angular porta servizi tradizionalmente lato server, come i Controllers dipendenti dalle Viste, al lato client$_G$ delle applicazioni Web. Di conseguenza, la maggior parte del carico sul server può essere ridotto.

Perchè Angular:

- **Data Binding**:
  è un modo automatico di aggiornamento della vista ogni volta che il modello cambia, così come l'aggiornamento del modello ogni volta che cambia la vista. Ciò elimina la manipolazione del DOM$_G$ dalla lista delle cose di cui occuparsi.

- **Controller**:
  definiscono il comportamento dietro gli elementi del DOM$_G$. AngularJS permette di esprimere il comportamento in una forma leggibile pulita, registrando callback$_G$ o guardando le modifiche dei modelli.

- **JavaScript**:
  A differenza di altri framework$_G$, non vi è alcuna necessità di ereditare da tipi di proprietà, per wrappare i modelli. I modelli in Angular sono semplici vecchi oggetti JavaScript$_G$. Questo rende il codice facile testare, mantenere,e facilita il riutilizzo..

- **Comunicazione con il Server**:
  AngularJS fornisce servizi integrati basati su XHR$_G$, nonché vari altri backends, utilizzando librerie di terze parti. Le Promises semplificano ulteriormente il codice per la gestione di ritorno asincrona dei dati.

- **Direttive**:
  Le direttive sono una caratteristica unica e potente disponibile solo in Angular. Consentono di inventare nuova sintassi HTML$_G$, specifica per l'applicazione.

- **Componenti Riutilizzabili**:
  Usando le direttive per creare componenti riutilizzabili. Un componente consente di nascondere la complessa struttura del DOM$_G$, CSS$_G$, e il comportamento. Questo permette di concentrarsi sia su ciò che l'applicazione deve fare o su come l'applicazione appare separatamente.

- **Integrabile**:
  AngularJS lavora molto bene con altre tecnologie. E' possibile aggiungere tanto o poco di AngularJS a una pagina esistente a seconda delle esigenze. Molte altri framework$_G$ richiedono di essere totalmente inclusi. Poichè AngularJS non ha uno stato globale più applicazioni possono essere eseguite su una singola pagina.

- **Iniettabile**:
  La dependency injection in AngularJS consente di descrivere in modo dichiarativo come l'applicazione è collegata. Ciò significa che l'applicazione non ha bisogno del metodo main(). Inoltre ogni componente che non si adatta alle nostre esigenze può essere facilmente sostituita.

- **Testabile**: AngularJS è stato progettato da zero per essere verificabile.

2.5 Meteor

Sebbene Meteor sia frequentemente comparato a Backbone.js e AngularJS per il suo design reattivo, esso è invece un framework$_G$ completo, in grado di utilizzare entrambi come moduli.

Le sue principali motivazioni progettuali sono elencate di seguito:

- Al posto di essere il server$_G$ ad inviare interi file HTML al client, Meteor invia solo i dati minimi necessari per rirenderizzare la parte della pagina che è cambiata. Ciò consente la creazione di applicazioni a bassa latenza di una sola pagina che evitano il totale refresh della pagina.

- Unifica il linguaggio (Javascript$_G$) utilizzato sul client$_G$ e sul server$_G$.

- La stessa API può essere utilizzata sia sul server$_G$ e il client$_G$ per interrogare il database. Nel browser$_G$, un'implementazione di MongoDB in memoria chiamata Minimongo permette l'interrogazione una cache di documenti che sono stati inviati al client$_G$.

- La compensazione di latenza: sul client$_G$, Meteor effettua il prefetch dei dati e simula modelli facendo sembrare che le chiamate di metodo sul server ritornino istantaneamente.

- Assoluta reattività: Tutti i livelli, dal database ai template, si aggiornano automaticamente quando necessario.

- Atmosfera, repository di pacchetti di Meteor, detiene più di 5.200.

- Meteor è stato progettato per essere facile da imparare, anche per i principianti.

3 Definizione del Prodotto

3.1 Metodo e formalismo di specifica

Verrà qui esposta l'architettura di Premi ad alto livello seguendo un approccio top-down$_G$: verranno prima descritti i package$_G$ e le loro dipendenze e successivamente le singole classi contenute al loro interno. I diagrammi delle classi e dei package$_G$ seguono il formalismo UML$_G$2.0 e la struttura dei package segue una prassi ("best practice$_G$") di AngularJS$_G$ che propone una suddivisione dei componenti per funzionalità dell'applicazione in alternativa alla classica suddivisione Model-View-Controller$_G$, più difficile da mantenere per applicazioni di medie o grandi dimensioni. Per ulteriori approfondimenti consultare la guida al sito scotch.io oppure il tutorial di urigo:angular-meteor.
Si illustreranno poi i Design Pattern utilizzati nella fase di progettazione ad alto livello e si descriveranno le interazioni dell'utente con l'applicazione attraverso i diagrammi di attività$_G$.

3.2 Presentazione dell'architettura generale del sistema

I componenti sono stati suddivisi prima in base al loro contributo a specifiche funzionalità del software e solo successivamente per appartenenza ai ruoli del pattern MVC$_G$. Questo aumenta la chiarezza espositiva dei diagrammi, evita la creazione di package$_G$ contenenti un numero eccessivo di classi e aiuta a compiere verifiche mirate a singoli componenti.
È importante specificare che il framework AngularJS unisce view e controller attraverso una dichiarazione esterna a entrambi, che fa parte del meccanismo detto di *routing* o di reindirizzamento dell'utente; view e controller inoltre non sanno di essere collegati tra loro e comunicano attraverso un oggetto chiamato *$scope*. Questo rende l'architettura sia di tipo Model-View-Controller$_G$ che di tipo Model-View-ViewModel$_G$.
Per motivi di leggibilità $scope e routing non verranno rappresentati in modo esplicito nei diagrammi dei package e delle classi di questo documento, ma sono comunque da considerarsi impliciti nelle dipendenze tra i view e controller dei componenti.

4 Diagrammi dei Package

Di seguito vengono descritti componenti principali del sistema e le loro dipendenze.

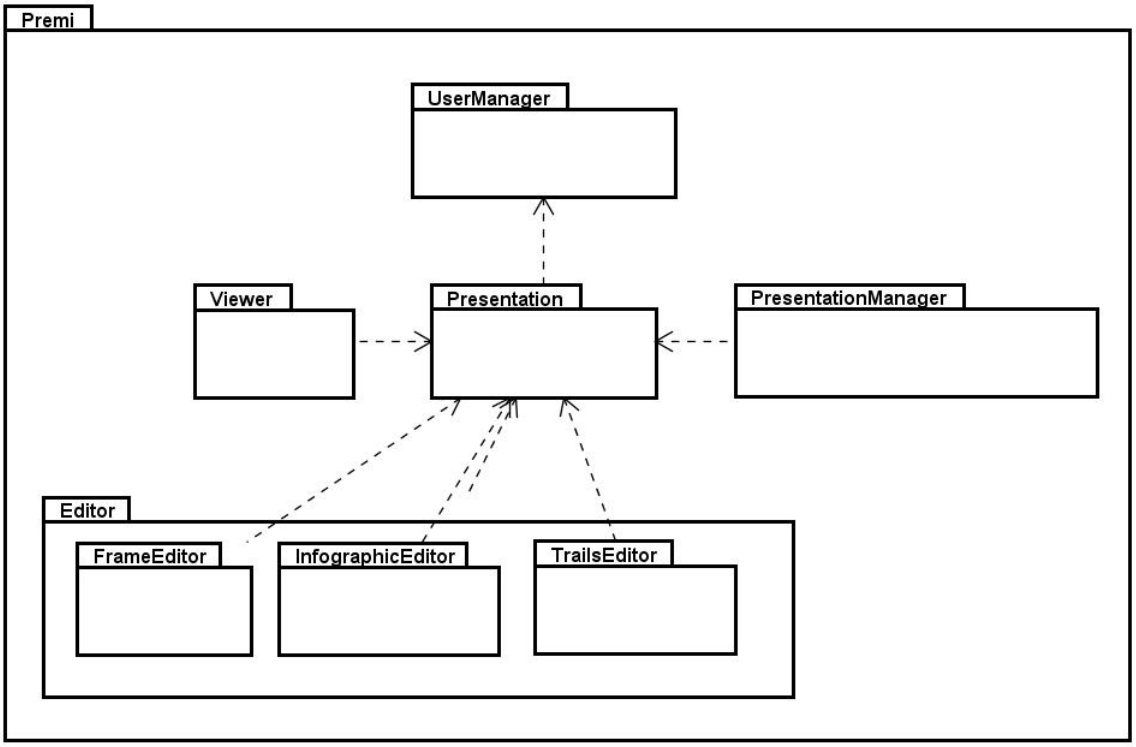

Figura 1: Diagramma dei package di Premi.

L'applicazione è costituita da un solo package$_G$ principale chiamato `Premi`; oltre alla view ed al controller principali, al suo interno sono presenti:

- `Premi.UserManager` è il package$_G$ di gestione dei dati dell'utente, contiene view e controller per la sua creazione e modifica, e la classe che lo definisce;
- `Premi.Viewer` racchiude gli elementi necessari alla visualizzazione della presentazione nei vari contesti previsti(presentazione live, pubblica e privata);
- `Premi.Presentation` racchiude la struttura generale della presentazione;
- `Premi.PresentationManager` contiene gli elementi necessari alla gestione delle presentazioni da parte dell'utente;
- `Premi.Editor` è il package$_G$ dedicato alla modifica interna delle presentazioni; possiede al suo interno tre ulteriori package$_G$:
  - `Premi.Editor.FrameEditor` si occupa di creare, modificare o cancellare i Frame$_G$ contenuti nella presentazione;
  - `Premi.Editor.InfographicEditor` posiziona Frame$_G$ o altri elementi all'interno di un poster;

– `Premi.Editor.TrailsEditor` ordina i Frame per la creazione di uno o più percorsi di presentazione.

5 Descrizione dei singoli componenti

5.1 Premi

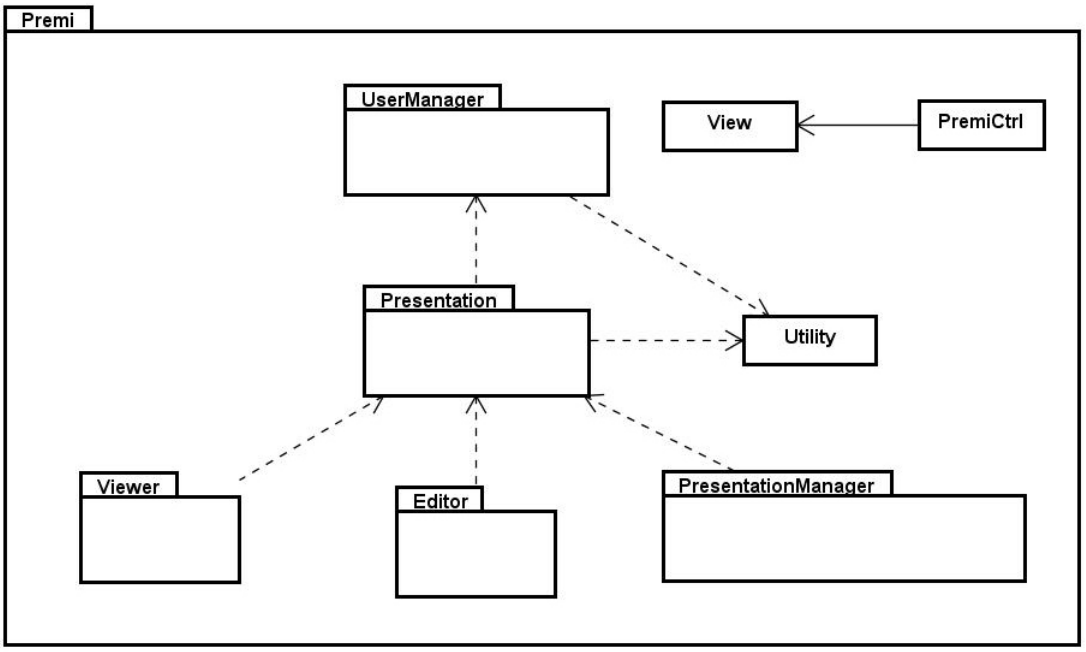

Figura 2: Diagramma del package Premi

5.1.1 Premi.Utility

Nome: Utility

Tipo: class

Package: Premi

Descrizione: classe statica di utilità che fornisce strumenti per interagire con la base di dati

5.1.2 Premi.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi
- **Descrizione:** view generale che funge da sfondo per le altre view

5.1.3 Premi.PremiCtrl

- **Nome:** PremiCtrl
- **Tipo:** class
- **Package:** Premi
- **Descrizione:** controller generale che modifica la view principale in base alle scelte dell'utente
- **Relazioni con altri componenti:** la classe altera l'aspetto di `Premi.View` caricando le view di cui l'utente ha bisogno

5.2 Premi.UserManager

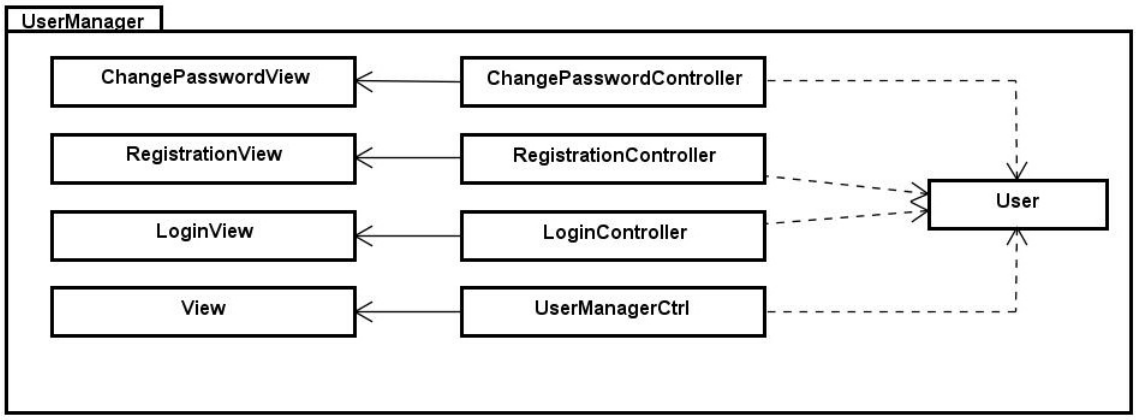

Figura 3: Diagramma del package Premi.UserManager

5.2.1 Premi.UserManager.User

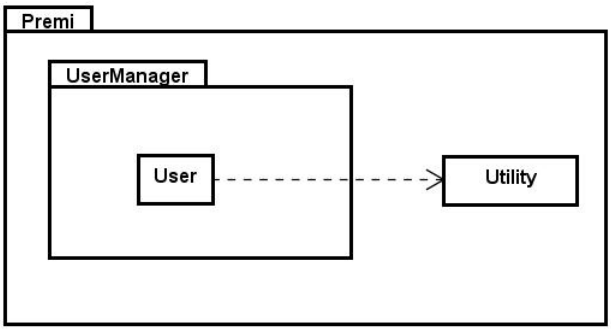

Figura 4: Diagramma delle dipendenze della classe User.

- **Nome:** User
- **Tipo:** class
- **Package:** Premi.UserManger
- **Descrizione:** classe che definisce un utente e fornisce le funzionalità necessarie alla sua creazione, al suo login e ad un eventuale cambio di password
- **Relazioni con altri componenti:** si collega a `Premi.Utility` per le interazioni con la base di dati

5.2.2 Premi.UserManager.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.UserManger
- **Descrizione:** vista generale del package$_G$ `UserManager`. Puo' contenere le view abbinate alle funzionalità di `User`

5.2.3 Premi.UserManager.UserManagerCtrl

- **Nome:** UserManagerCtrl
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** controller generale del package$_G$ `UserManager` dedicato a fornire alla view strumenti per l'interazione con l'utente
- **Relazioni con altri componenti:** si collega a `Premi.UserManager.View` per mostrare i dati dell'utente che va a prelevare tramite Premi.UserManager.User

5.2.4 Premi.UserManager.LoginView

- **Nome:** LoginView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di effettuare la login

5.2.5 Premi.UserManager.LoginController

- **Nome:** LoginController
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per effettuare la login
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.UserManager.LoginView` e utilizza `Premi.UserManager.User` per trasmettere e ricevere le informazioni relative al login

5.2.6 Premi.UserManager.RegistrationView

- **Nome:** RegistrationView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di registrarsi nel sistema

5.2.7 Premi.UserManager.RegistrationController

- **Nome:** RegistrationController
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per registrare l'utente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.UserManager.RegistrationView` e utilizza `Premi.UserManager.User` per trasmettere le informazioni relative alla registrazione

5.2.8 Premi.UserManager.ChangePasswordView

- **Nome:** ChangePasswordView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di cambiare la propria password

5.2.9 Premi.UserManager.ChangePasswordController

- **Nome:** ChangePassword
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per cambiare la password dell'utente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.UserManager.ChangePasswordView` e utilizza `Premi.UserManager.User` per trasmettere le informazioni relative al cambio password

5.3 Premi.Viewer

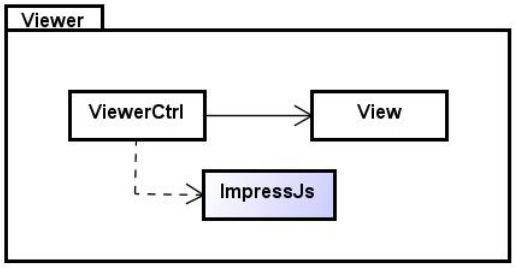

Figura 5: Diagramma del package Premi.Viewer

5.3.1 Premi.Viewer.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Viewer
- **Descrizione:** view che mostra la presentazione all'utente, offrendo funzionalità dipendenti dalla visibilità(pubblica o privata) o dal contesto in cui si trova(presentazione live)

5.3.2 Premi.ViewerCtrl

- **Nome:** ViewerCtrl
- **Tipo:** class
- **Package:** Premi.Viewer
- **Descrizione:** abilita funzionalità nella view in base all'utente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Viewer.View`, utilizza `Premi.UserManager.User` per verificare se l'utente è il proprietario della presentazione, e la libreria$_G$ esterna `ImpressJs` per aggiungere animazioni alla presentazione

5.4 Premi.Presentation

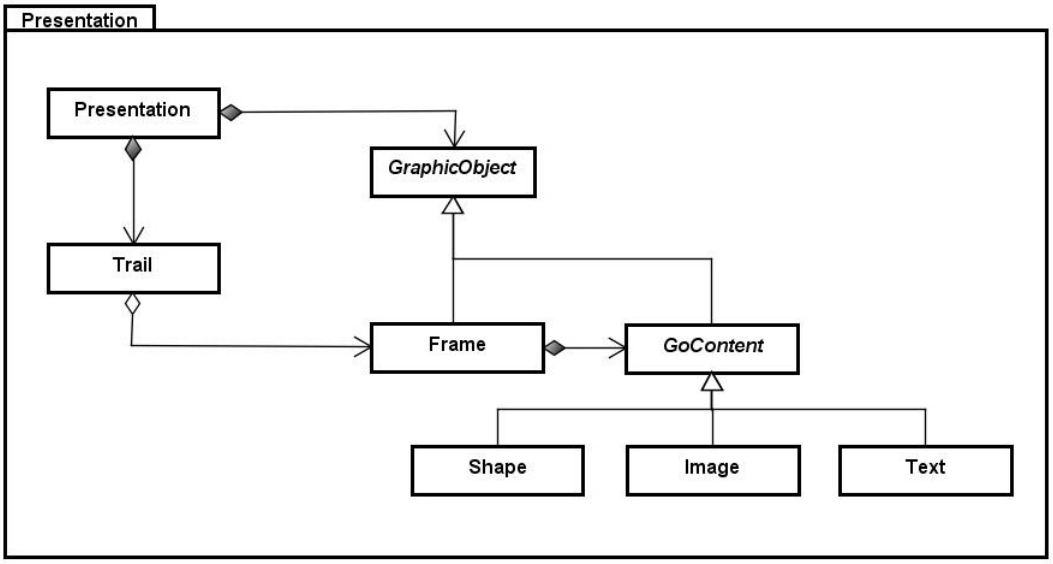

Figura 6: Diagramma del package Premi.Presentation

5.4.1 Premi.Presentation.GraphicObject

- **Nome:** GraphicObject
- **Tipo:** *abstract class*
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta gli oggetti grafici nella presentazione

5.4.2 Premi.Presentation.GoContent

- **Nome:** GoContent
- **Tipo:** *abstract class*
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta gli oggetti grafici con contenuto di una presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GraphicObject`

5.4.3 Premi.Presentation.Text

- **Nome:** Text
- **Tipo:** class
- **Package:** Premi.Presentation

- **Descrizione:** rappresenta un'area di testo nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

5.4.4 Premi.Presentation.Image

- **Nome:** Image
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un'immagine nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

5.4.5 Premi.Presentation.Shape

- **Nome:** Shape
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta una figura nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

5.4.6 Premi.Presentation.Frame

- **Nome:** Frame
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un frame_G nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GraphicObject` e possiede un insieme di oggetti `Premi.Presentation.GoContent`

5.4.7 Premi.Presentation.Trail

- **Nome:** Trail
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un percorso nella presentazione
- **Relazioni con altri componenti:** possiede una lista di riferimenti ad oggetti di tipo `Premi.Presentation.Frame`, ma non è in possesso degli oggetti stessi

5.4.8 Premi.Presentation.Presentation

- **Nome:** Presentation
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta una presentazione
- **Relazioni con altri componenti:** possiede una lista di oggetti `premi.Presentation.GraphicObject` che possono essere sia Frame$_G$ che oggetti grafici generici e assieme compongono l'infografica$_G$ della presentazione, possiede inoltre una lista di percorsi e quindi di oggetti `Premi.Presentation.Trail`, identifica l'appartenenza di una presentazione ad un determinato `Premi.UserManager.User` e si collega alla base di dati tramite la classe statica `Premi.Utility`

5.5 Premi.PresentationManager

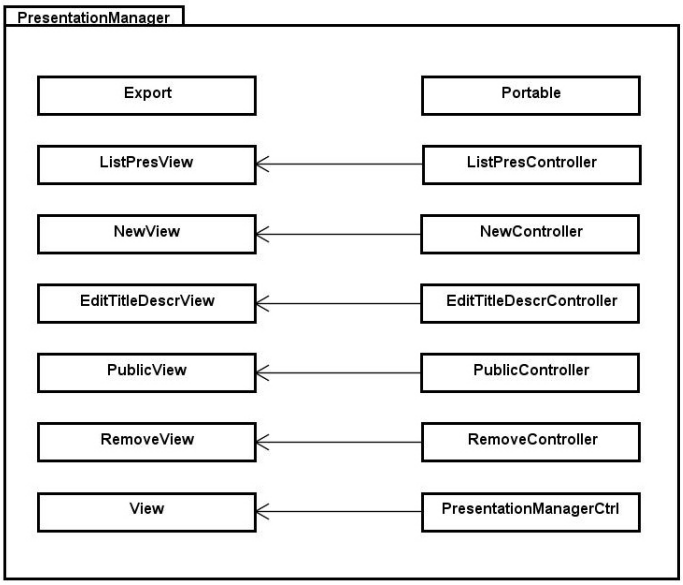

Figura 7: Diagramma del package Premi.PresentationManager

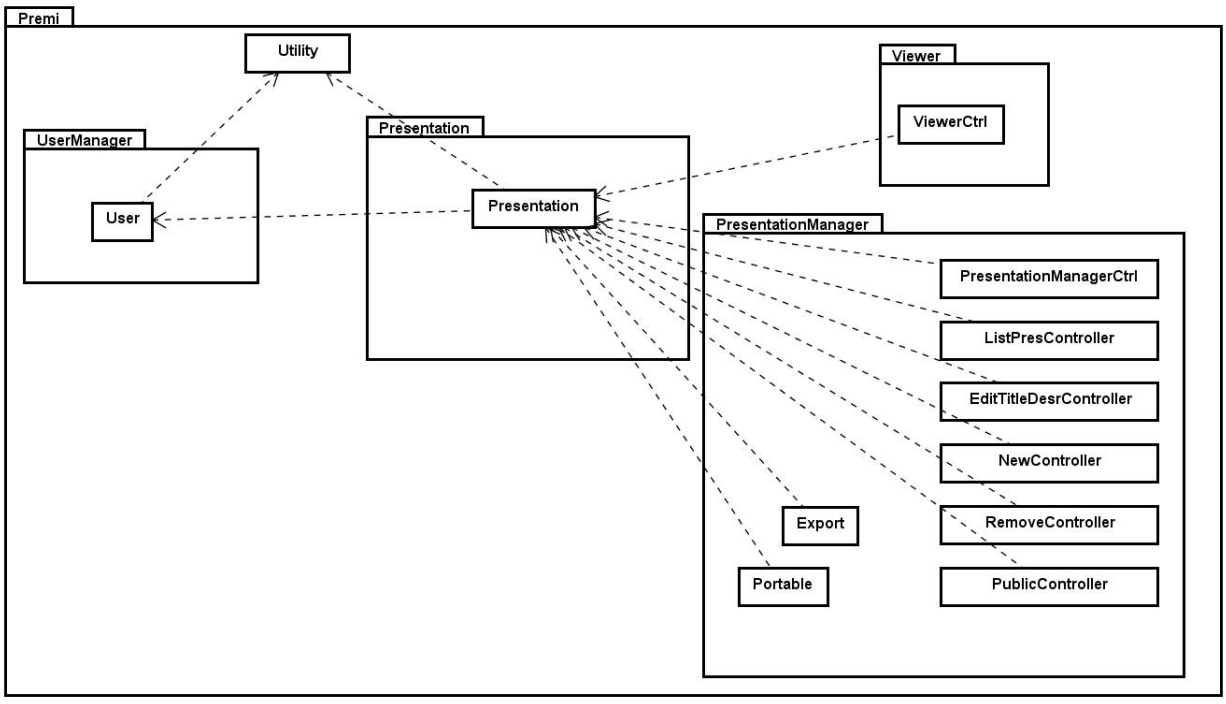

Figura 8: Diagramma delle dipendenze delle classi di PresentationManager

5.5.1 Premi.PresentationManager.Export

- **Nome:** Export
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** permette di esportare la presentazione in formato poster
- **Relazioni con altri componenti:** preleva le informazioni da `Premi.Presentation.Presentation` per costruire il poster

5.5.2 Premi.PresentationManager.Portable

- **Nome:** Portable
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** permette di rendere portabile una presentazione, e quindi di essere indipendente dal software Premi e di essere caricata come pagina HTML$_G$ da un browser$_G$
- **Relazioni con altri componenti:** preleva le informazioni da `Premi.Presentation.Presentation` per costruire la pagina HTML$_G$

5.5.3 Premi.PresentationManager.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.PresentationController
- **Descrizione:** è la view generale delle operazioni che l'utente puo' effettuare sulla sua lista di presentazioni

5.5.4 Premi.PresentationManager.PresentationManagerCtrl

- **Nome:** PresentationManagerCtrl
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti necessari alla gestione delle presentazioni
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.View`

5.5.5 Premi.PresentationManager.RemoveView

- **Nome:** RemoveView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra di conferma eliminazione della presentazione selezionata

5.5.6 Premi.PresentationManager.RemoveController

- **Nome:** RemoveController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per rimuovere la presentazione o annullare l'azione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.RemoveView` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per l'eliminazione della presentazione

5.5.7 Premi.PresentationManager.PublicView

- **Nome:** PublicView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra di conferma per rendere pubblica o privata una presentazione

5.5.8 Premi.PresentationManager.PublicController

- **Nome:** PublicController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per rendere pubblica o privata una presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.PublicView` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

5.5.9 Premi.PresentationManager.EditTitleDescrView

- **Nome:** EditTitleDescrView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra per la modifica del titolo e della descrizone della presentazione

5.5.10 Premi.PresentationManager.EditTitleDescrController

- **Nome:** EditTitleDescrController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view la possibilità di modificare titolo e descrizione della presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.EditTitleDescrView` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

5.5.11 Premi.PresentationManager.NewView

- **Nome:** NewView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra per la creazione di una nuova presentazione

5.5.12 Premi.PresentationManager.NewController

- **Nome:** NewController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per creare una nuova presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.NewView` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

5.5.13 Premi.PresentationManager.ListPresView

- **Nome:** ListPresView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra all'utente la lista delle sue presentazioni, e bottoni aggiuntivi per accedere alle altre funzionalità del package$_G$

5.5.14 Premi.PresentationManager.ListPresController

- **Nome:** ListPresController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view una lista preformata delle presentazioni dell'utente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.ListPresView` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per recuperare la lista delle presentazioni

5.6 Premi.Editor

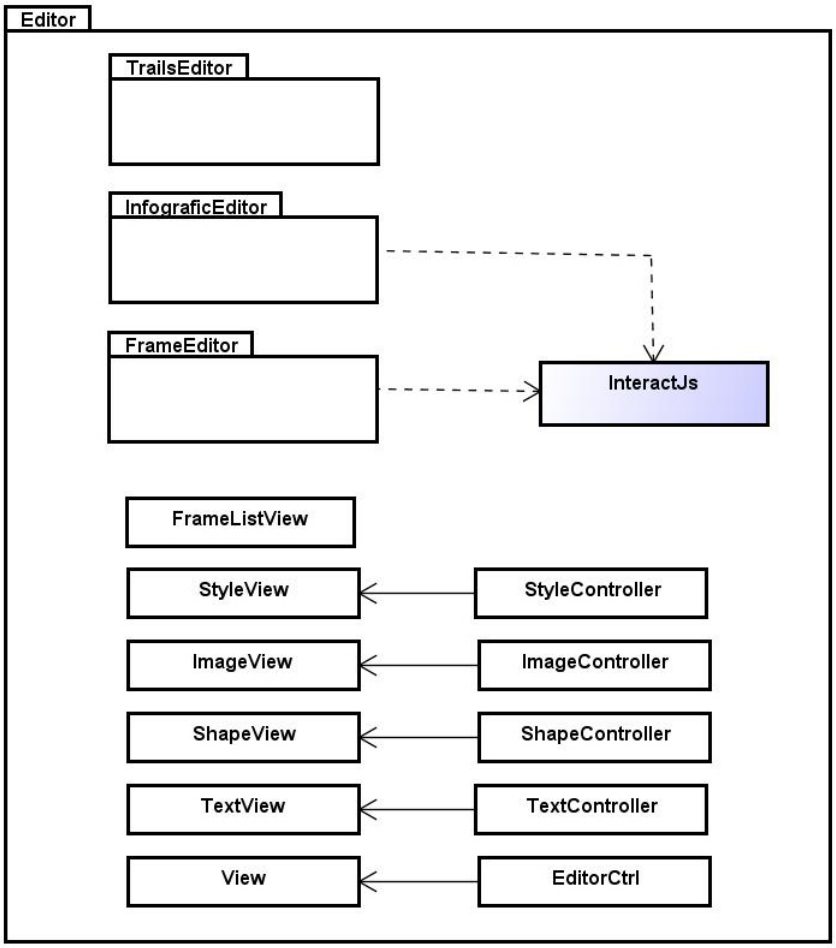

Figura 9: Diagramma del package Premi.Editor

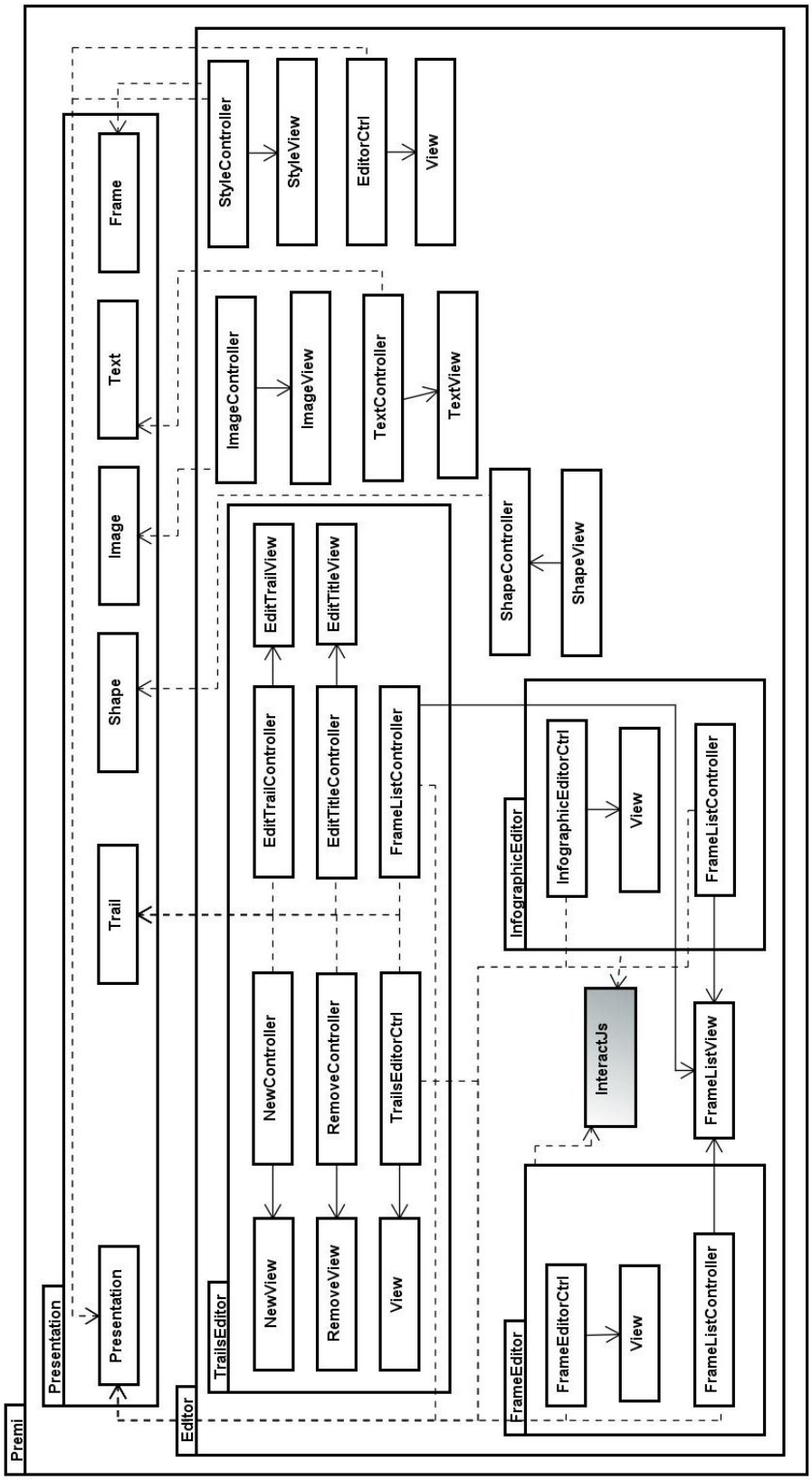

Figura 10: Diagramma delle classi di Premi.Editor e delle loro relazioni

5.6.1 Premi.Editor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** view generica dell'editor che funge da contenitore

5.6.2 Premi.Editor.EditorCtrl

- **Nome:** EditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionzalità per la modifica di una presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.View` e sfrutta `Premi.Presentation.Presentation` per recuperare la presentazione dalla base di dati

5.6.3 Premi.Editor.TextView

- **Nome:** TextView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di un'area di testo

5.6.4 Premi.Editor.TextController

- **Nome:** TextController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per creare e modificare un'area di testo
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TextView` e utilizza la classe `Premi.Presentation.Text` per rappresentare le aree di testo

5.6.5 Premi.Editor.ShapeView

- **Nome:** ShapeView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di una figura

5.6.6 Premi.Editor.ShapeController

- **Nome:** ShapeController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per creare e modificare una figura
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.ShapeView` e utilizza la classe `Premi.Presentation.Shape` per rappresentare le figure

5.6.7 Premi.Editor.ImageView

- **Nome:** ImageView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di un'immagine

5.6.8 Premi.Editor.ImageController

- **Nome:** ImageController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per aggiungere un'immagine
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.ImageView` e utilizza la classe `Premi.Presentation.Image` per rappresentare le immagini

5.6.9 Premi.Editor.StyleView

- **Nome:** StyleView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** permette la modifica dello stile di un Frame$_G$ oppure dello sfondo dell'infografica$_G$

5.6.10 Premi.Editor.StyleController

- **Nome:** StyleController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per la modifica dello stile
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.StyleView`, e utilizza la classe `Premi.Presentation.Frame` per rappresentare lo stile dei Frame$_G$ e dell'infografica$_G$

5.6.11 Premi.Editor.FrameListView

- **Nome:** FrameListView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** view di base per la rappresentazione di una lista dei Frame$_G$ contenuti all'interno della presentazione

5.7 Premi.Editor.FrameEditor

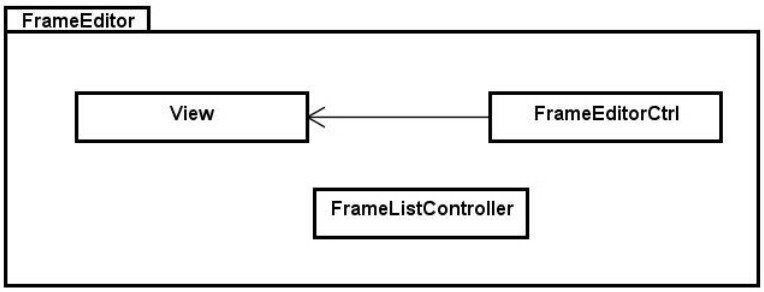

Figura 11: Diagramma del package Premi.Editor.FrameEditor

5.7.1 Premi.Editor.FrameEditor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor.FrameEditor.View
- **Descrizione:** è la view generale della fase di modifica dei Frame$_G$

5.7.2 Premi.Editor.FrameEditor.FrameEditorCtrl

- **Nome:** FrameEditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor.FrameEditor.FrameEditorCtrl
- **Descrizione:** fornisce alla view le funzionalità per la gestione delle altre view contenute al suo interno
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.FrameEditor.View` ed elabora la presentazione tramite `Premi.Presentation.Presentation`

5.7.3 Premi.Editor.FrameEditor.FrameListController

- **Nome:** FrameListController
- **Tipo:** class
- **Package:** Premi.Editor.FrameEditor
- **Descrizione:** fornisce alla view `FrameListView` le funzionalità per rappresentazione della lista dei Frame$_G$
- **Relazioni con altri componenti:** è uno dei tre controller dedicati di `Premi.Editor.FrameListView` e utilizza la classe `Premi.Presentation.Presentation` per prelevare i Frame$_G$ e crearne un'anteprima

5.8 Premi.Editor.InfographicEditor

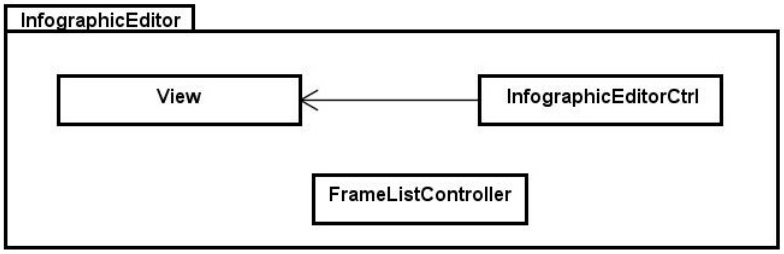

Figura 12: Diagramma del package Premi.Editor.InfographicEditor

5.8.1 Premi.Editor.InfographicEditor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor.InfographicEditor
- **Descrizione:** è la view generale della fase di creazione o modifica dell'infografica$_G$

5.8.2 Premi.Editor.InfographicEditor.InfographicEditorCtrl

- **Nome:** InfographicEditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor.InfographicEditor
- **Descrizione:** fornisce alla view le funzionalità per la gestione delle altre view contenute al suo interno, che permetteranno all'utente di produrre un'infografica$_G$ attraverso l'utilizzo di oggetti grafici, tra cui i Frame$_G$ creati nella fase precedente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.InfoGraphicEditor.View` ed elabora la presentazione tramite `Premi.Presentation.Presentation`

5.8.3 Premi.Editor.InfographicEditor.FrameListController

- **Nome:** FrameListController
- **Tipo:** class
- **Package:** Premi.Editor.InfographicEditor
- **Descrizione:** fornisce alla view `FrameListView` le funzionalità per rappresentazione della lista dei Frame$_G$
- **Relazioni con altri componenti:** è uno dei tre controller dedicati di `Premi.Editor.FrameListView` e utilizza la classe `Premi.Presentation.Presentation` per prelevare i Frame$_G$ e crearne un'anteprima

5.9 Premi.Editor.TrailsEditor

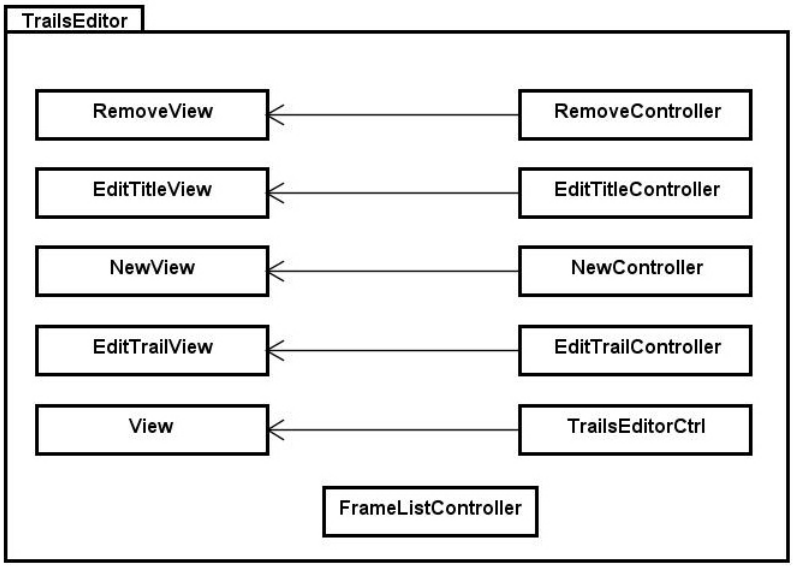

Figura 13: Diagramma del package Premi.Editor.TrailsEditor

5.9.1 Premi.Editor.TrailsEditor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** view generale della della fase di creazione o modifica dei percorsi di presentazione

5.9.2 Premi.Editor.TrailsEditor.TrailsEditorCtrl

- **Nome:** TrailsEditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view le funzionalità per la gestione delle altre view contenute al suo interno, che permetteranno di gestire i percorsi di presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TrailsEditor.View` , si collega a `Premi.Presentation.Presentation` per recuperare oggetti di tipo `Premi.Presentation.Trail`

5.9.3 Premi.Editor.TrailsEditor.FrameListController

- **Nome:** FrameListController
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view `FrameListView` le funzionalità per rappresentazione della lista dei Frame$_G$
- **Relazioni con altri componenti:** è uno dei tre controller dedicati di `Premi.Editor.FrameListView` e utilizza la classe `Premi.Presentation.Presentation` per prelevare i Frame$_G$ e crearne un'anteprima

5.9.4 Premi.Editor.TrailsEditor.EditTrailView

- **Nome:** EditTrailView
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** view dedicata alla modifica di un percorso di presentazione

5.9.5 Premi.Editor.TrailsEditor.EditTrailController

- **Nome:** EditTrailController
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view le funzionalità per la gestione delle altre view contenute al suo interno, che permetteranno all'utente di produrre un percorso di presentazione con i Frame da lui creati
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TrailsEditor.EditTrailView` e modifica il percorso sfruttando la classe `Premi.Presentation.Trail`

5.9.6 Premi.Editor.TrailsEditor.NewView

- **Nome:** NewView
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** mostra la finestra di creazione di un percorso di presentazione

5.9.7 Premi.Editor.TrailsEditor.NewController

- **Nome:** NewController
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view le funzionalità per la creazione di un nuovo percorso di presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TrailsEditor.NewView` e crea il nuovo percorso sfruttando la classe `Premi.Presentation.Trail`

5.9.8 Premi.Editor.TrailsEditor.EditTitleView

- **Nome:** EditTitleView
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** mostra la finestra di modifica del titolo di un percorso di presentazione

5.9.9 Premi.Editor.TrailsEditor.EditTitleController

- **Nome:** EditTileController
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view le funzionalità per la modifica del titolo di un percorso di presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TrailsEditor.EditTitleView` e modifica il titolo del percorso sfruttando la classe `Premi.Presentation.Trail`

5.9.10 Premi.Editor.TrailsEditor.RemoveView

- **Nome:** RemoveView
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** mostra la finestra di conferma rimozione di un percorso di presentazione

5.9.11 Premi.Editor.TrailsEditor.RemoveController

- **Nome:** RemoveController
- **Tipo:** class
- **Package:** Premi.Editor.TrailsEditor
- **Descrizione:** fornisce alla view le funzionalità per la rimozione di un percorso di presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TrailsEditor.RemoveView`

e rimuove il percorso sfruttando la classe `Premi.Presentation.Trail`

6 Diagrammi delle attività

Vengono illustrati ora i diagrammi di attività. Viene illustrato il diagramma principale ad alto livello e i diversi sotto-diagrammi specifici.

6.1 Attività principali

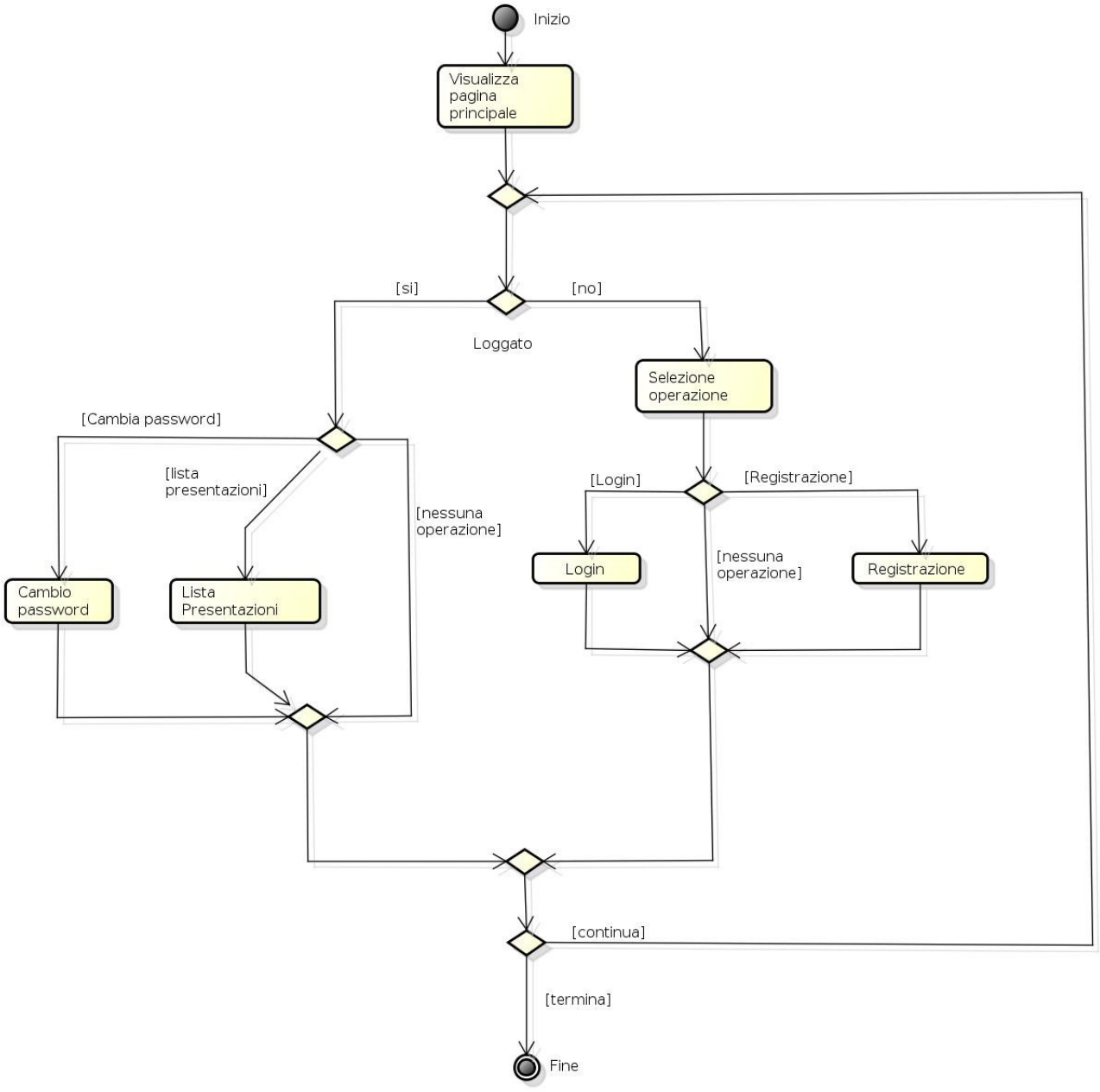

Figura 14: Attività principali

L'utente nel momento in cui accede nel programma ha la possibilità di effettuare la login o di registrarsi nel sistema. L'utente loggato può invece effettuare il logout, andare nella lista presentazioni ed effettuare il cambio password.

6.2 Lista presentazioni

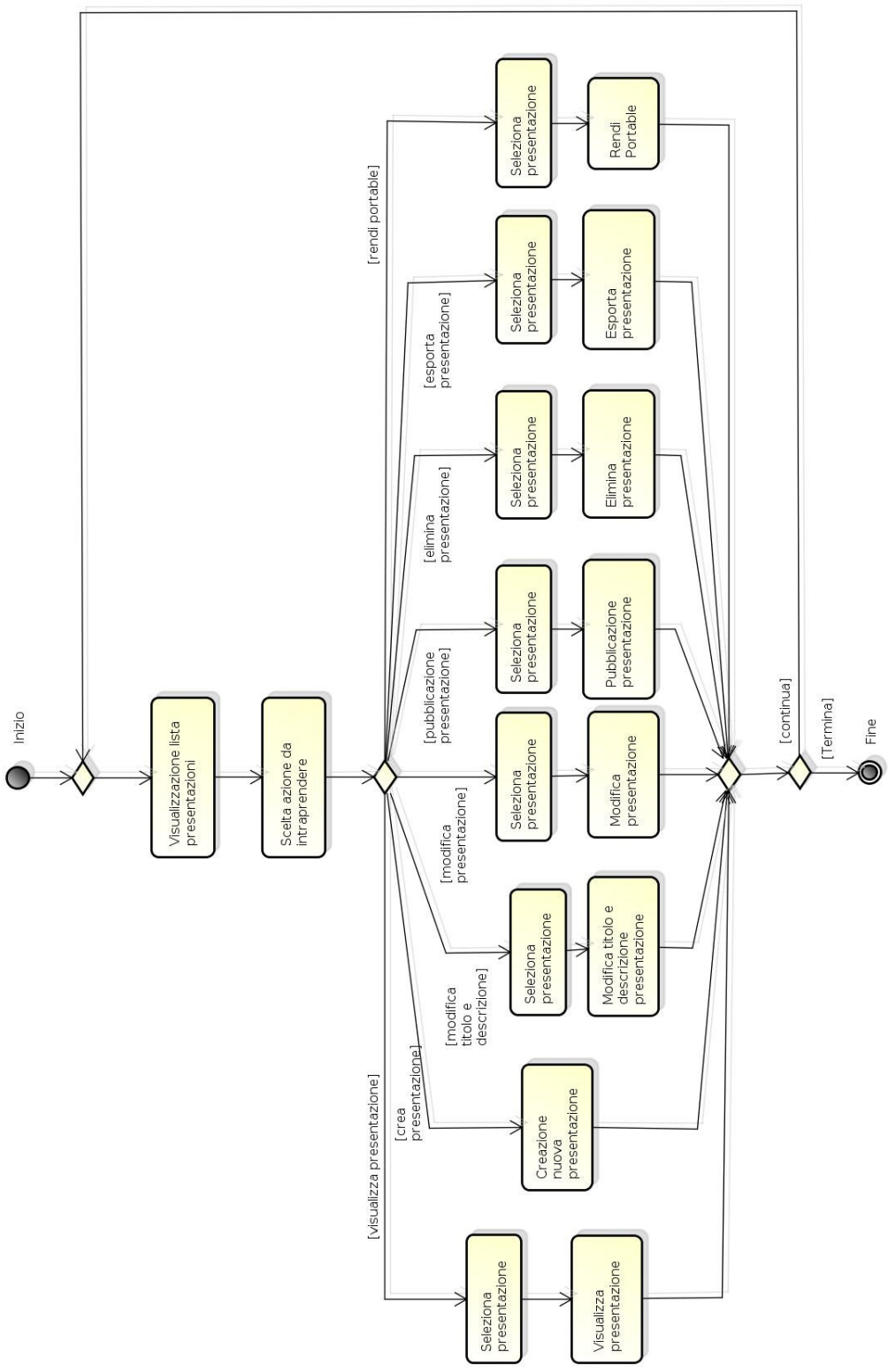

Figura 15: Lista presentazioni

Le scelte che ha l'utente una volta entrato nella lista presentazioni sono: Visualizza presentazione, creazione nuova presentazione, modifica titolo e descrizione presentazione, modifica presentazione, pubblicazione presentazione, elimina presentazione, esporta presentazione, rendi portable.

6.3 Login

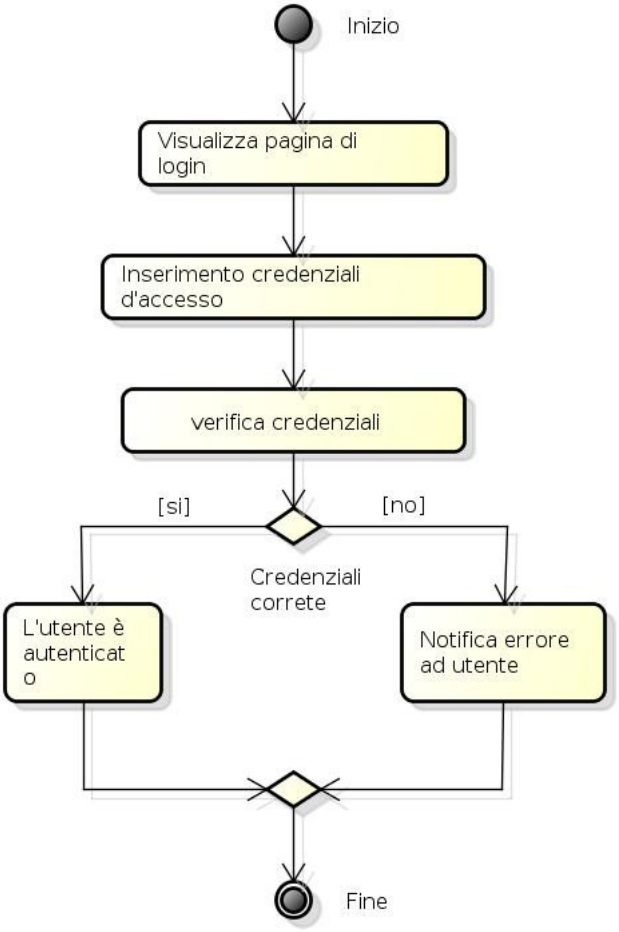

Figura 16: Login utente

L'utente quando accede nella pagina di login inserisce le credenziali che corrispondono a email e password. Se sono corrette viene autenticato altrimenti viene restituito un messaggio d'errore.

6.4 Registrazione

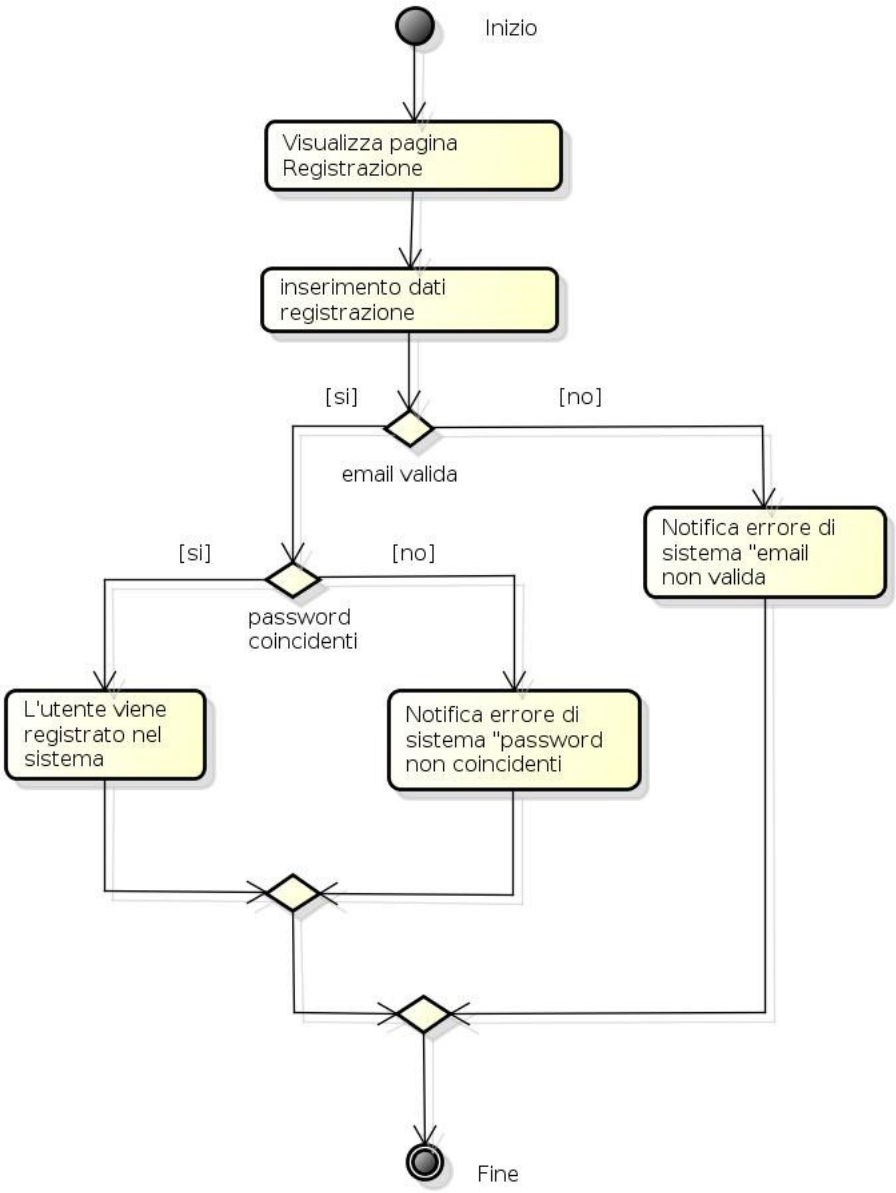

Figura 17: Registrazione utente

L'utente quando accede nella parte di registrazione inserisce l'email e la password. Se l'email non ha un formato valido o se è già presente nel sistema viene restituito un errore altrimenti viene verificato che le password inserite coincidano. In caso affermativo l'utente viene registrato nel sistema, altrimenti viene restituito un messaggio d'errore.

6.5 Cambio password

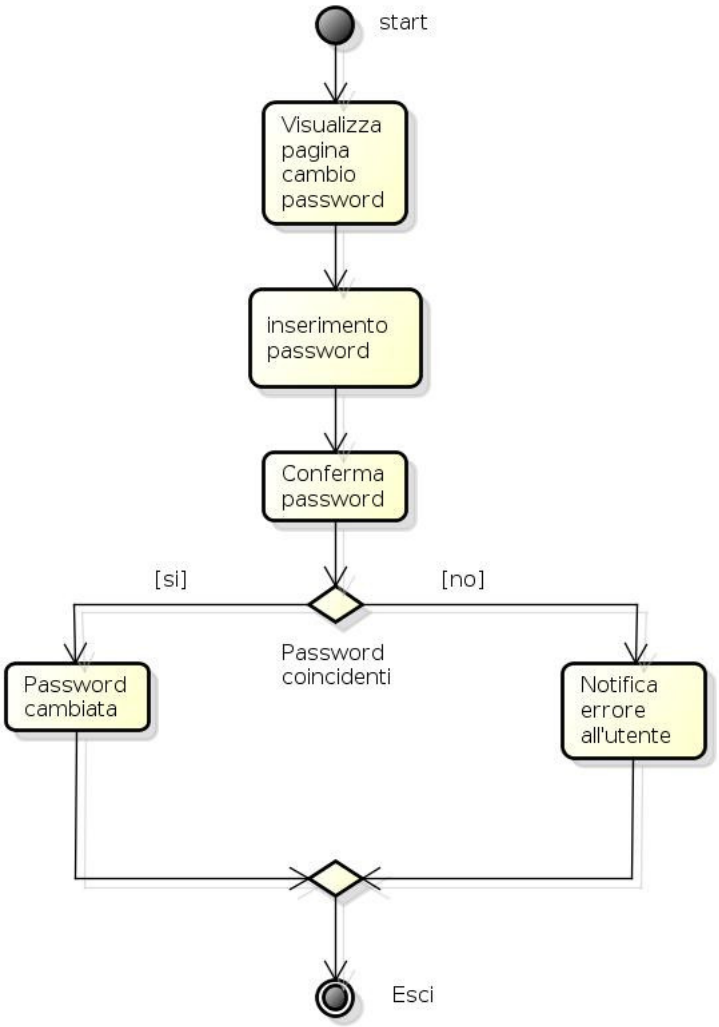

Figura 18: Cambio password

Per effettuare il cambio password l'utente inserisce la nuova password e la conferma della nuova. Se sono coincidenti viene effettuato il cambio password altrimenti viene notificato un errore all'utente.

6.6 Visualizzatore

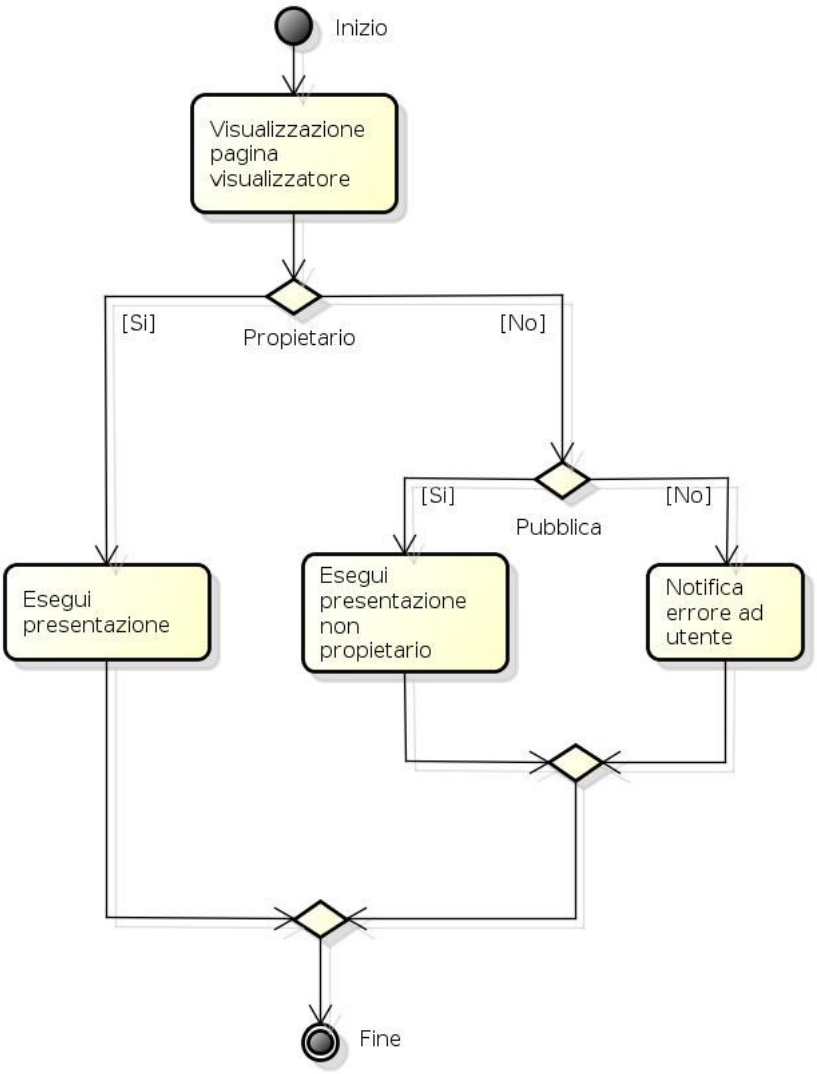

Figura 19: Visualizzatore

Se l'utente che visualizza la presentazione è l'utente proprietario viene eseguita la presentazione in modalità proprietario altrimenti viene controllato se la presentazione è pubblica. In caso affermativo viene eseguita la presentazione in modalità non proprietario altrimenti viene notificato un errore, in quanto se la presentazione non è pubblica non può essere visualizzata. L'utente non proprietario per accedere ad una presentazione deve ottenere il link generato dall'utente proprietario nel momento in cui la rende pubblica. L'utente proprietario può rendere privata una presentazione in ogni momento perciò il controllo se è pubblica, deve essere effettuato per controllare se il link è ancora valido.

6.7 Esegui presentazione proprietario

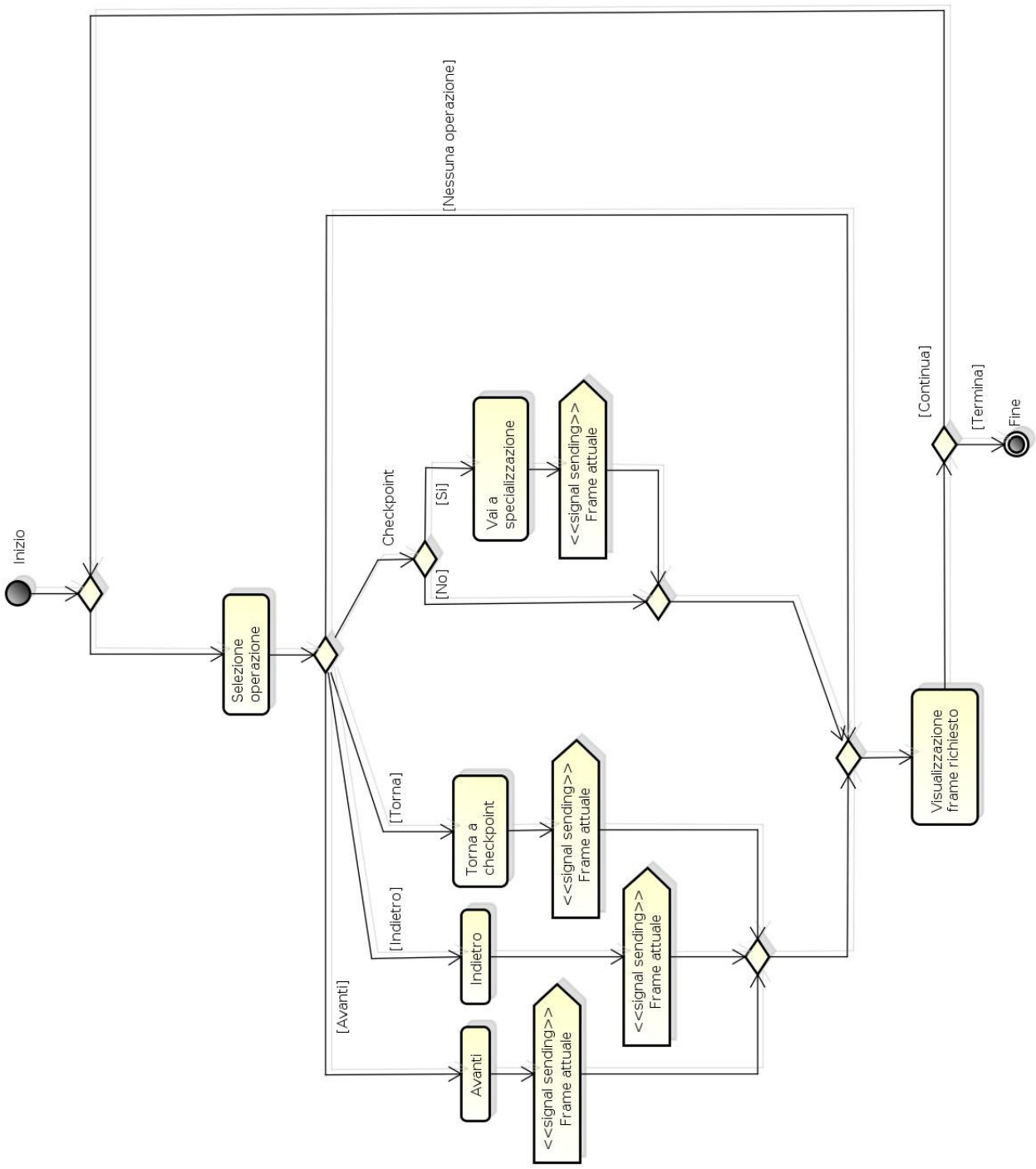

Figura 20: Esegui presentazione proprietario

Se la modalità di visualizzazione presentazione è in modalità proprietario si hanno le seguenti scelte:

- avanti: per andare avanti di una frame;
- indietro: per tornare indietro di un frame;

- torna a checkpoint: permette di tornare al frame iniziale o di tornare al checkpoint se si è entrati in un percorso di specializzazione;
- checkpoint: se il frame corrente è un checkpoint l'utente con questa scelta entra nel percorso di specializzazione.

Il segnale Frame attuale inviato permette agli utenti non proprietari di visualizzare il frame corrente scelto dal proprietario. L'utente ha la possibilità di uscire quando lo desidera.

6.8 Esegui presentazione non proprietario

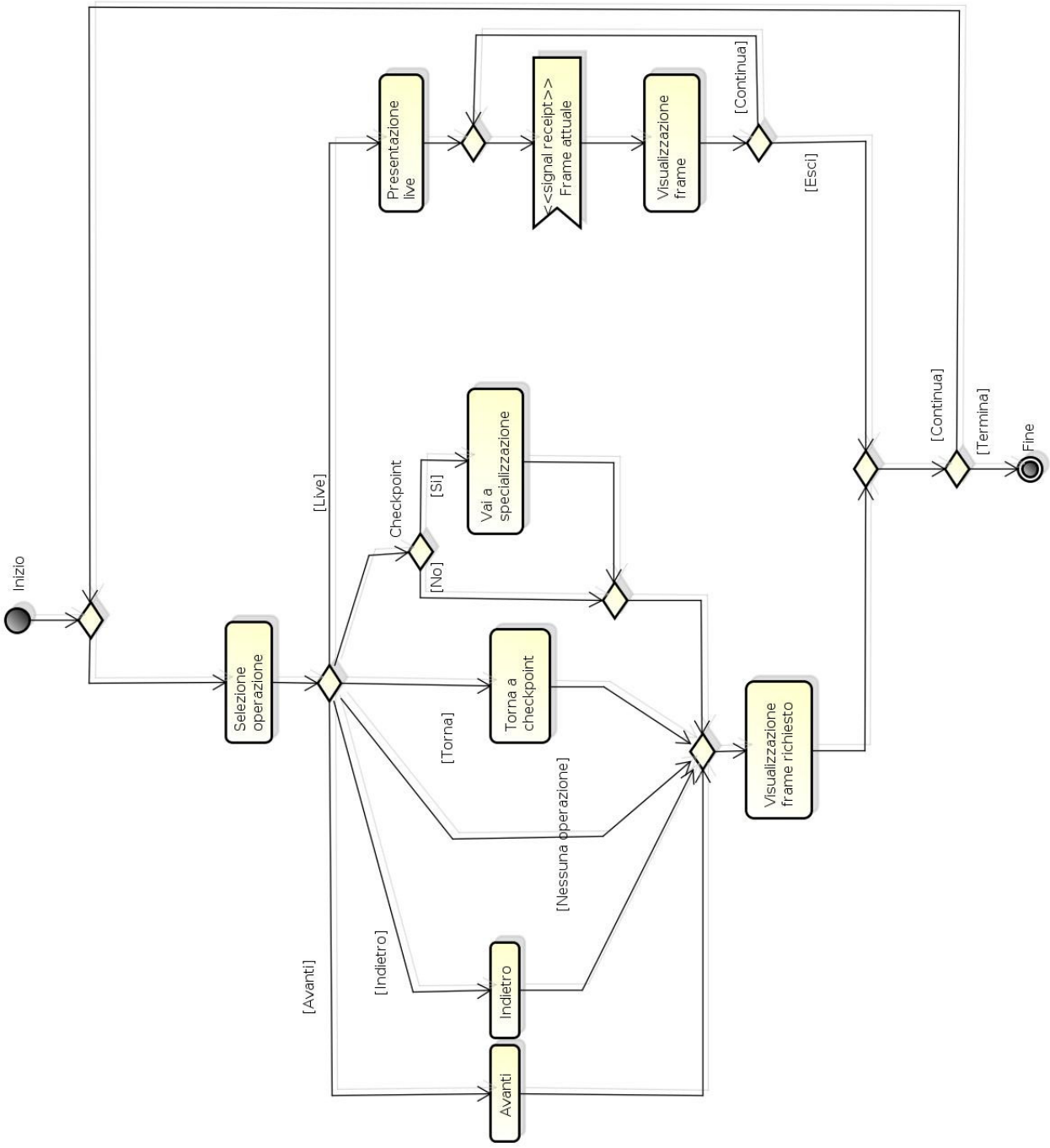

Figura 21: Esegui presentazione non proprietario

Se la modalità di visualizzazione presentazione è in modalità proprietario si hanno le seguenti scelte:

- avanti: per andare avanti di una frame;
- indietro: per tornare indietro di un frame;

- torna a checkpoint: permette di tornare al frame iniziale o di tornare al checkpoint se si è entrati in un percorso di specializzazione;
- checkpoint: se il frame corrente è un checkpoint l'utente con questa scelta entra nel percorso di specializzazione;
- presentazione live: con questa scelta l'utente visualizza il frame corrente che l'utente proprietario ha scelto di visualizzare.

L'utente ha la possibilità di uscire quando lo desidera.

6.9 Creazione presentazione

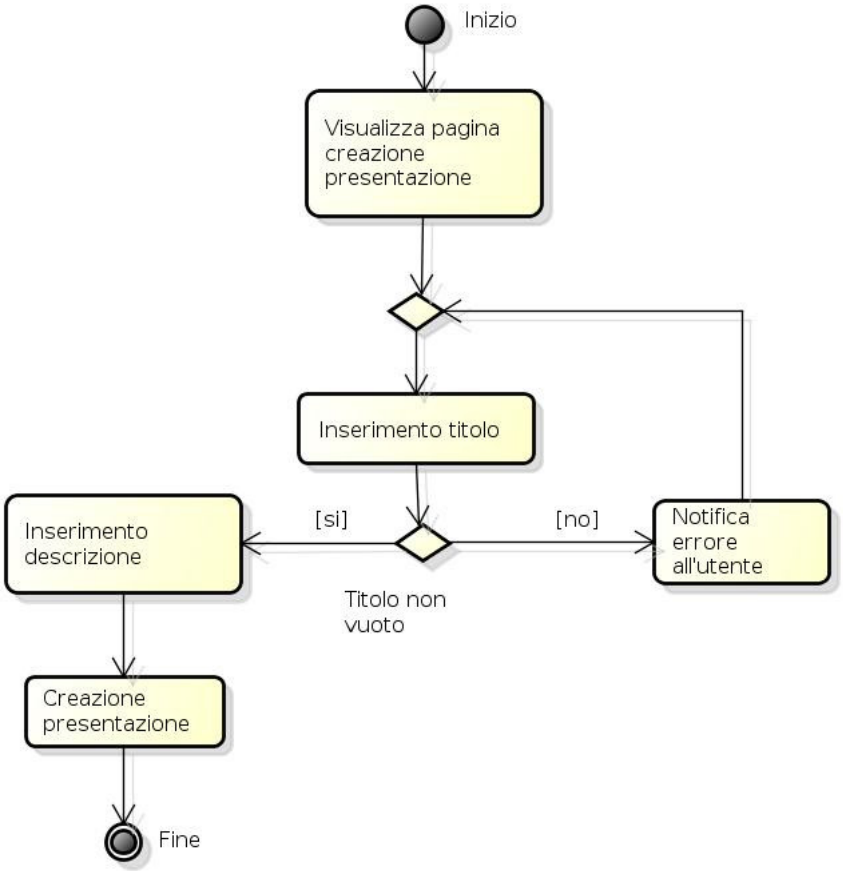

Figura 22: Creazione presentazione

L'utente può creare una nuova presentazione inserendo titolo e descrizione. Se non inserisce il titolo viene visualizzata una notifica di errore altrimenti la presentazione viene inserita nel sistema.

6.10 Modifica titolo e descrizione presentazione

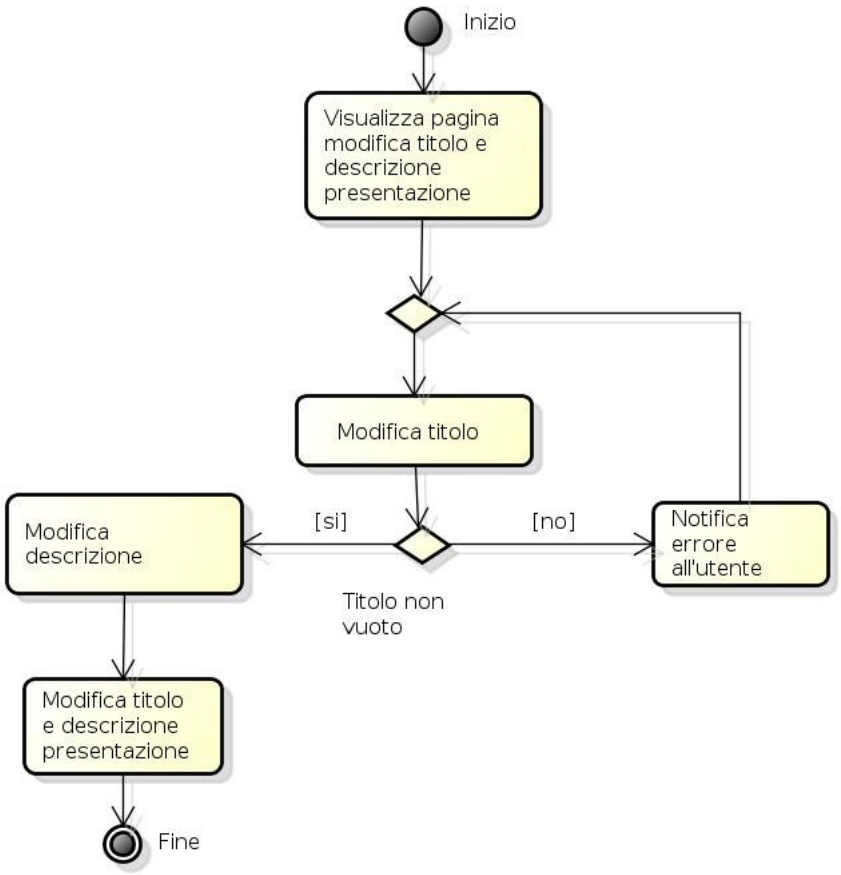

Figura 23: Modifica titolo e descrizione della presentazione

L'utente può modificare sia il titolo che la descrizione. Se il titolo non è vuoto le modifiche vengono salvate altrimenti viene restituita una notifica di errore.

6.11 Pubblicazione presentazione

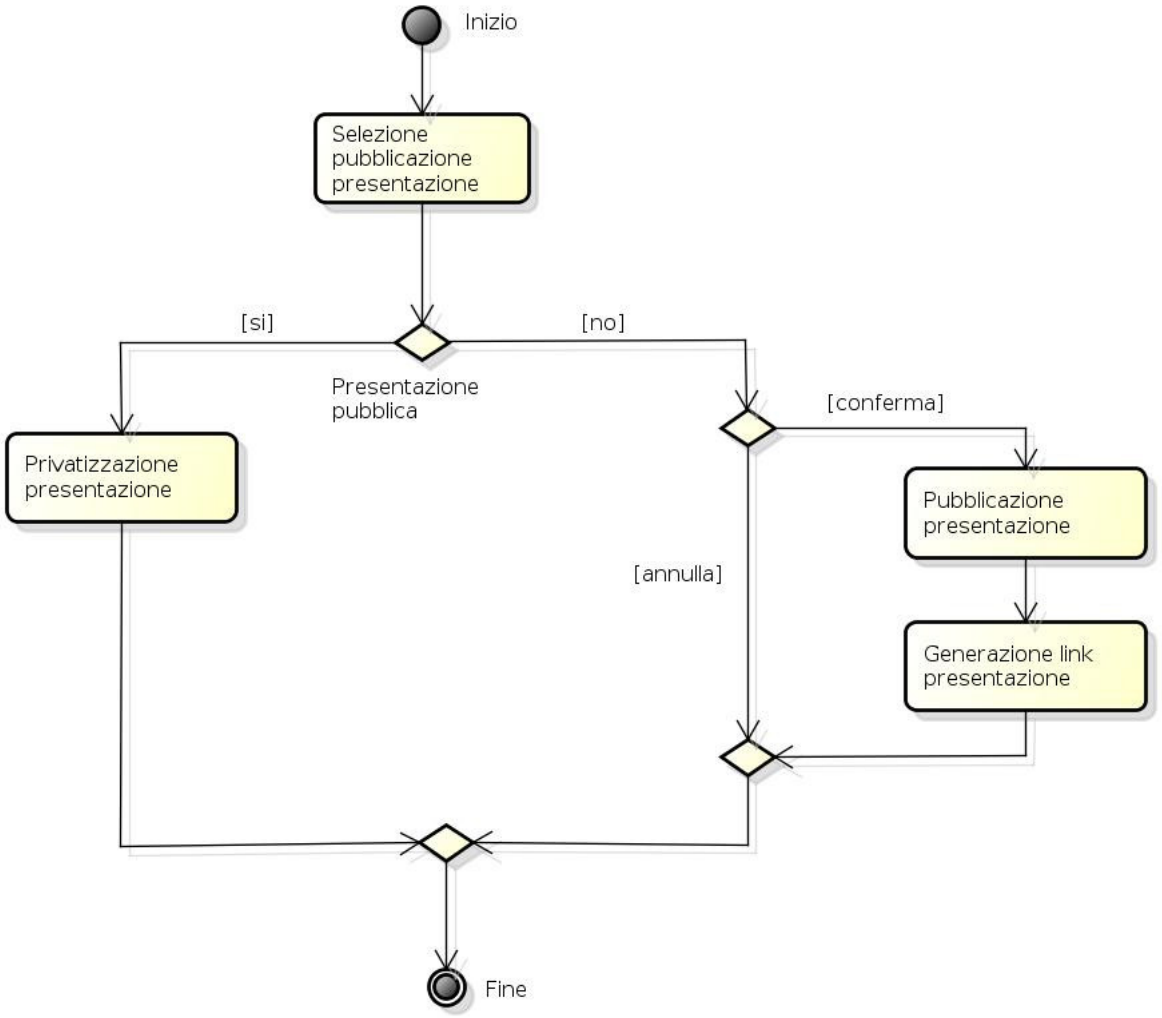

Figura 24: Pubblicazione presentazione

Se la presentazione è già pubblica rende privata la stessa, altrimenti se conferma la pubblicazione la rende visibile al pubblico e viene generato un link da inviare a chi voglia visualizzarla.

6.12 Elimina presentazione

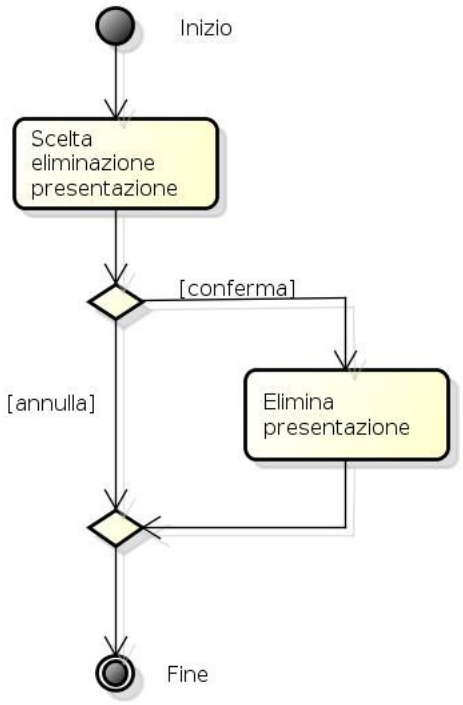

Figura 25: Elimina presentazione

L'utente deve confermare l'eliminazione altrimenti l'operazione viene annullata.

6.13 Esportazione presentazione

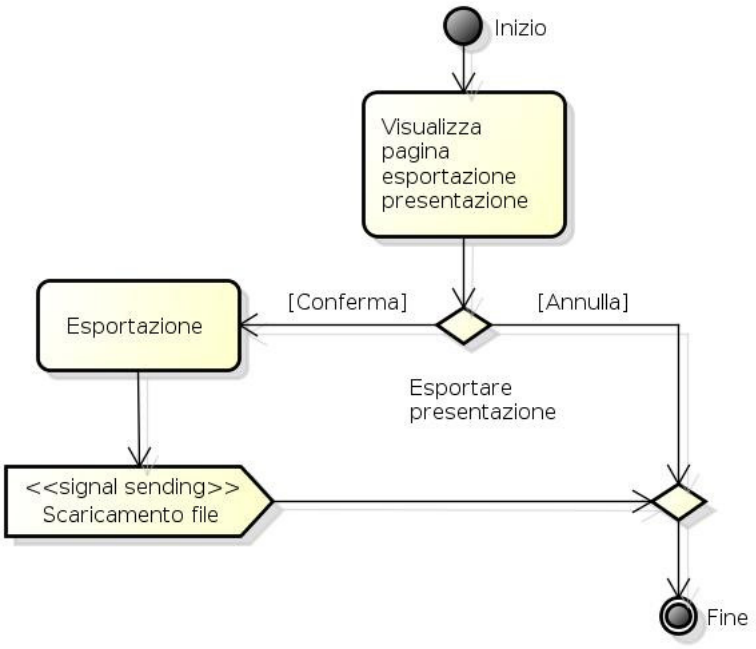

Figura 26: Esportazione presentazione

L'utente può esportare la presentazione in modo da ottenere un poster. Se l'operazione è confermata vengono esportati i dati e si procede allo scaricamento del file. Altrimenti viene annullata.

6.14 Rendi portable

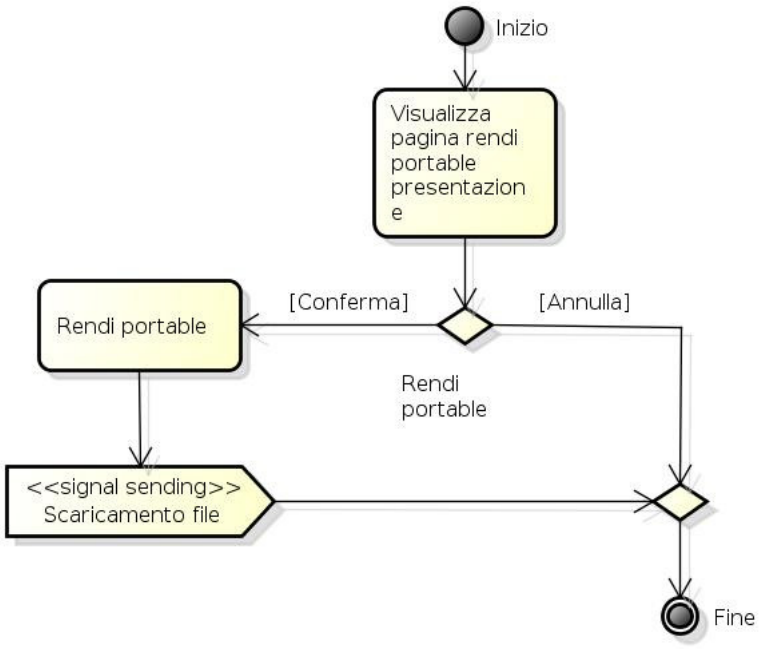

Figura 27: Rendi portable

L'utente può rendere portabile la presentazione in modo da vederla offline. Se l'operazione è confermata la presentazione viene resa portabile e si procede allo scaricamento dei file. Altrimenti viene annullata.

6.15 Modifica presentazione

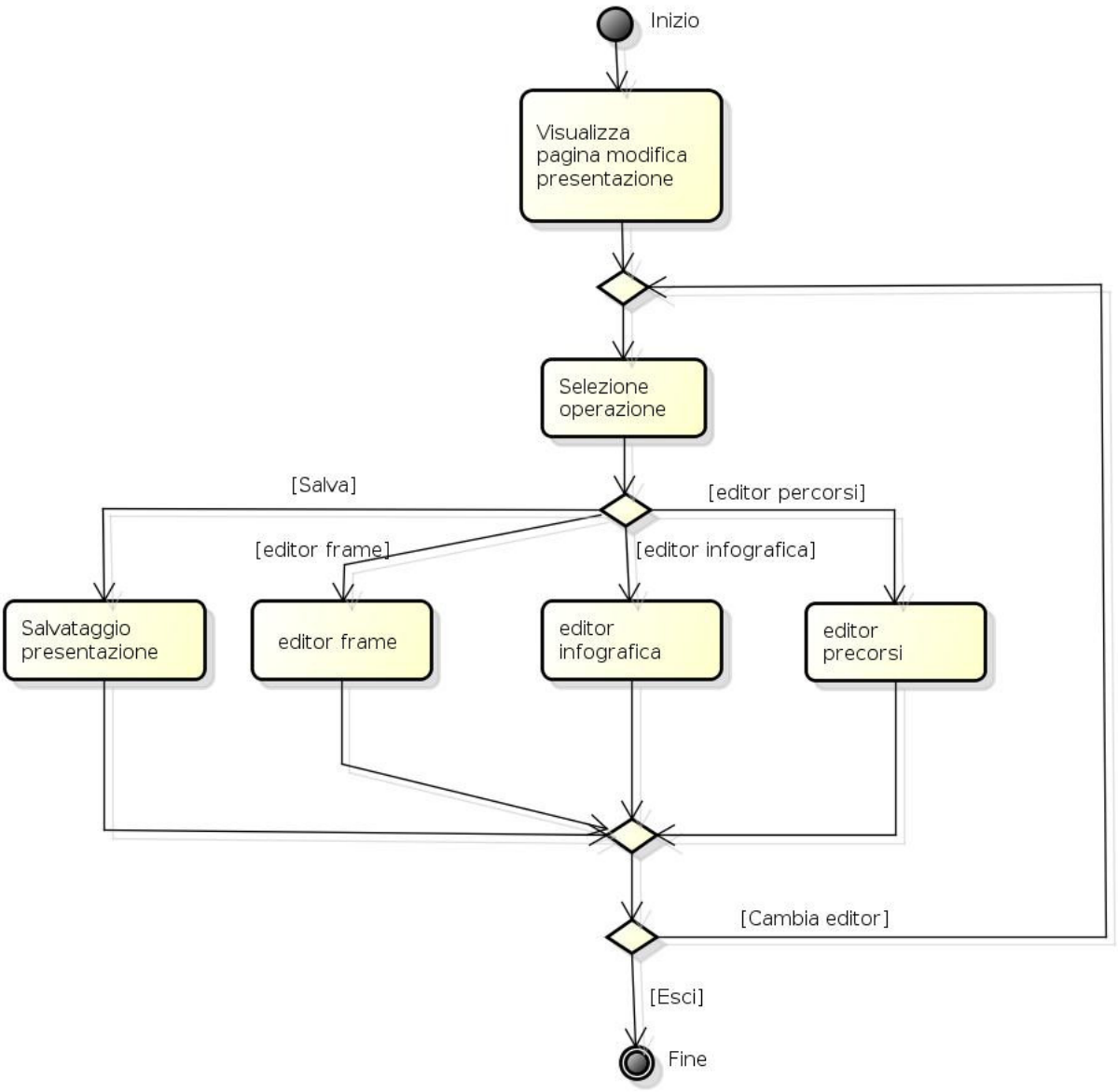

Figura 28: Modifica presentazione

L'utente può scegliere di: salvare la presentazione, andare nel frame editor, andare nell'infografica editor o nell'editor percorsi. L'utente può in ogni momento cambiare editor.

6.16 Frame editor

L'utente procede alla creazione del frame oppure può selezionarne uno. Una volta selezionato viene visualizzato nell'editor e a questo punto può scegliere se eliminarlo o modificarlo.

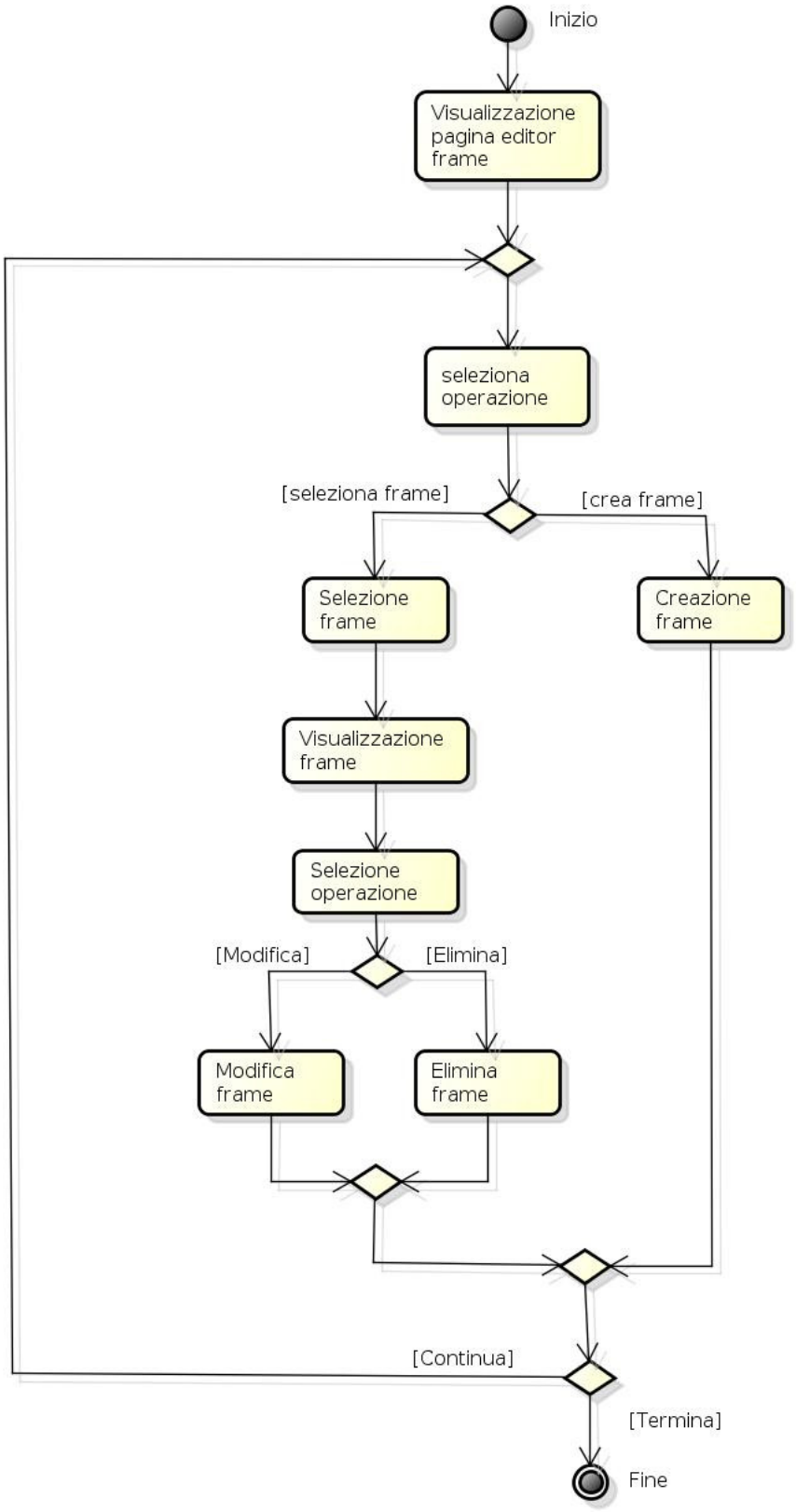

Figura 29: Frame editor

6.17 Modifica frame

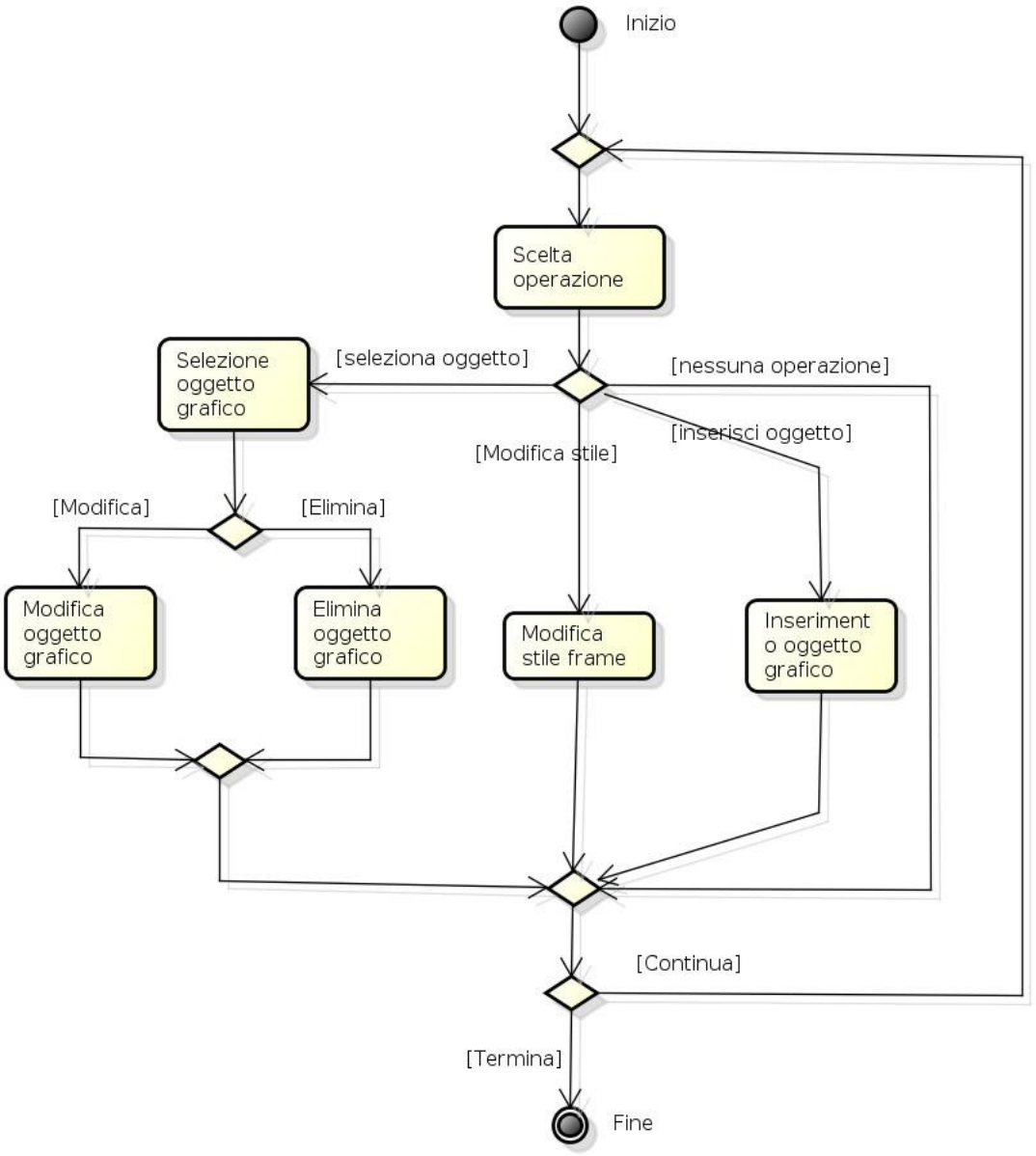

Figura 30: Modifica frame

L'utente può effettuare le seguenti operazioni:

- Selezione oggetto grafico: una volta selezionato l'oggetto può essere eliminato o modificato;
- Modificare lo stile del frame;
- Inserire un oggetto grafico nel frame;
- Non effettuare nessuna operazione.

6.18 Infografica editor

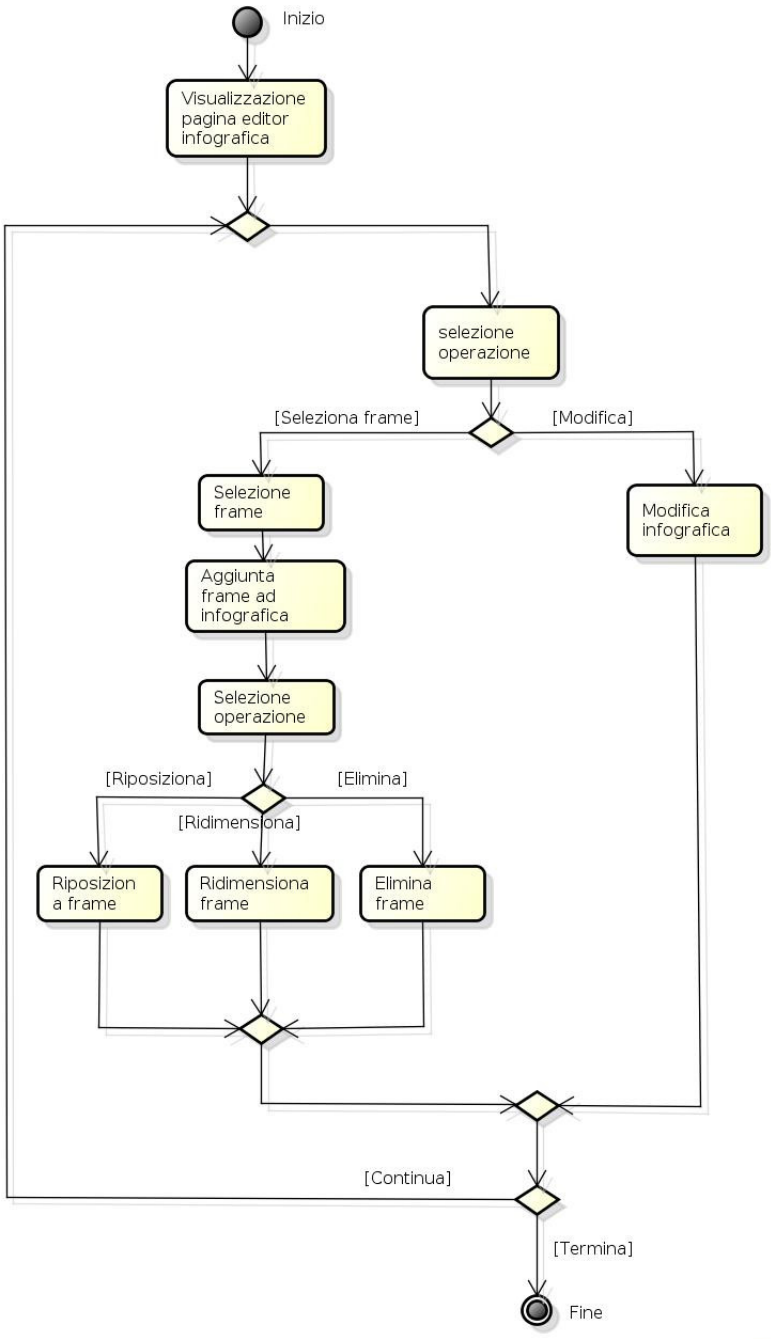

Figura 31: Infographic editor

L'utente può scegliere di:

- modificare l'infografica
- selezionare un frame e successivamente aggiungierlo all'infografica, eliminarlo dall'infografica o cambiargli la posizione e la grandezza e altezza.

6.19 Modifica infografica

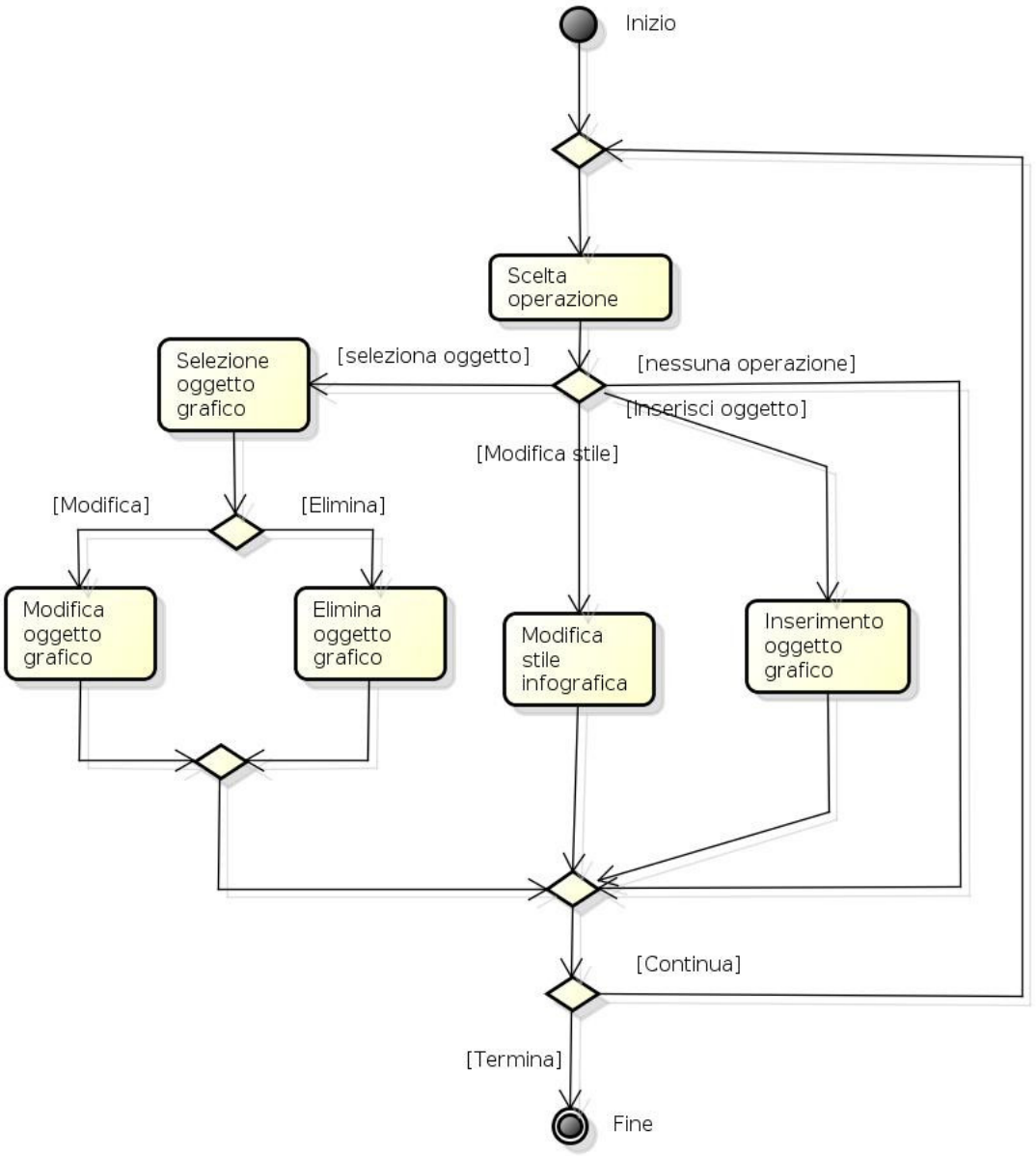

Figura 32: Modifica infografica

L'utente può effettuare le seguenti operazioni:

- Selezione oggetto grafico: una volta selezionato l'oggetto può essere eliminato o modificato;
- Modificare lo stile dell'infografica;
- Inserire un oggetto grafico nell'infografica;
- Non effettuare nessuna operazione.

6.20 Editor percorsi

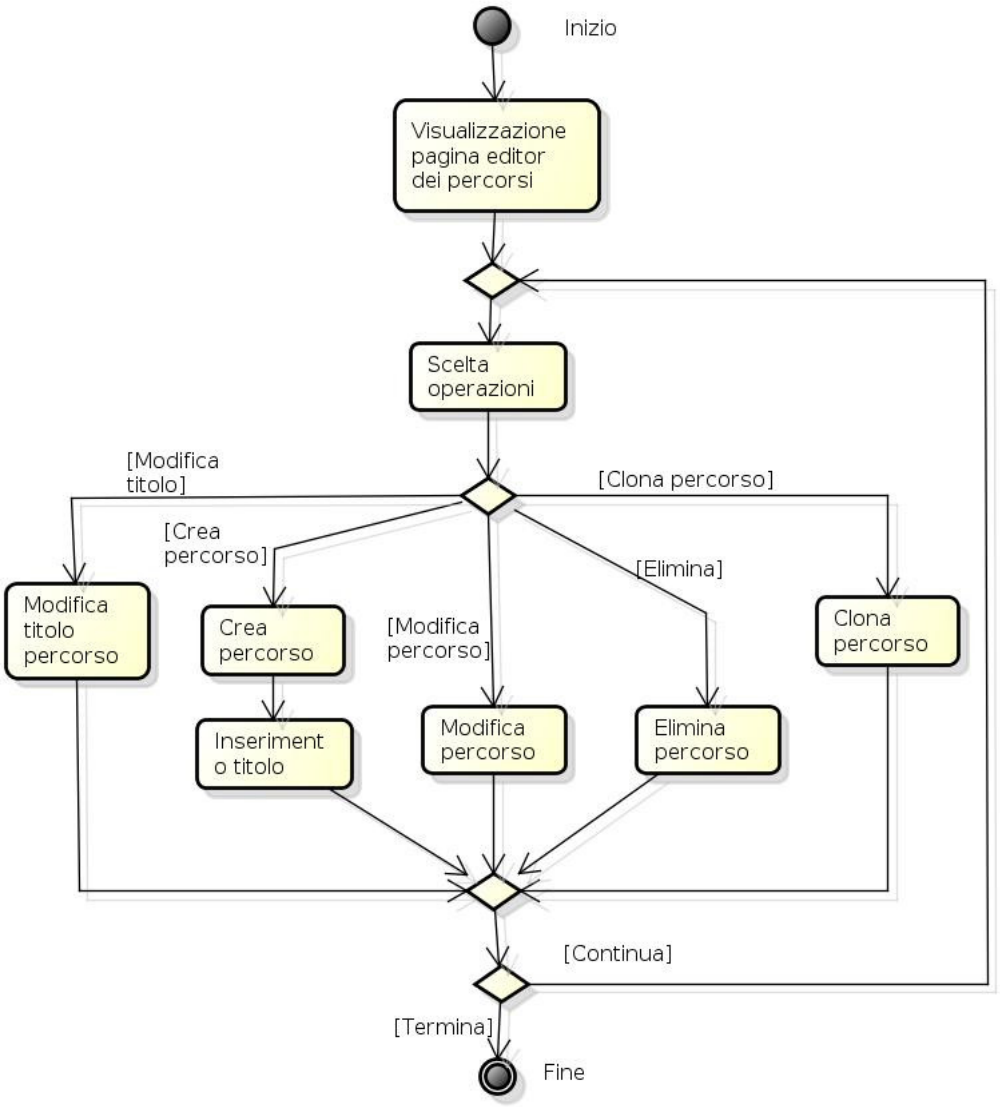

Figura 33: Editor percorsi

Le operazioni che può fare l'utente sono:

- Modificare il titolo del percorso selezionato;
- Creare un nuovo percorso inserendo il titolo;
- Modificare il percorso selezionato;
- Eliminare il percorso selezionato;
- Clonare il percorso selezionato.

6.21 Modifica percorso

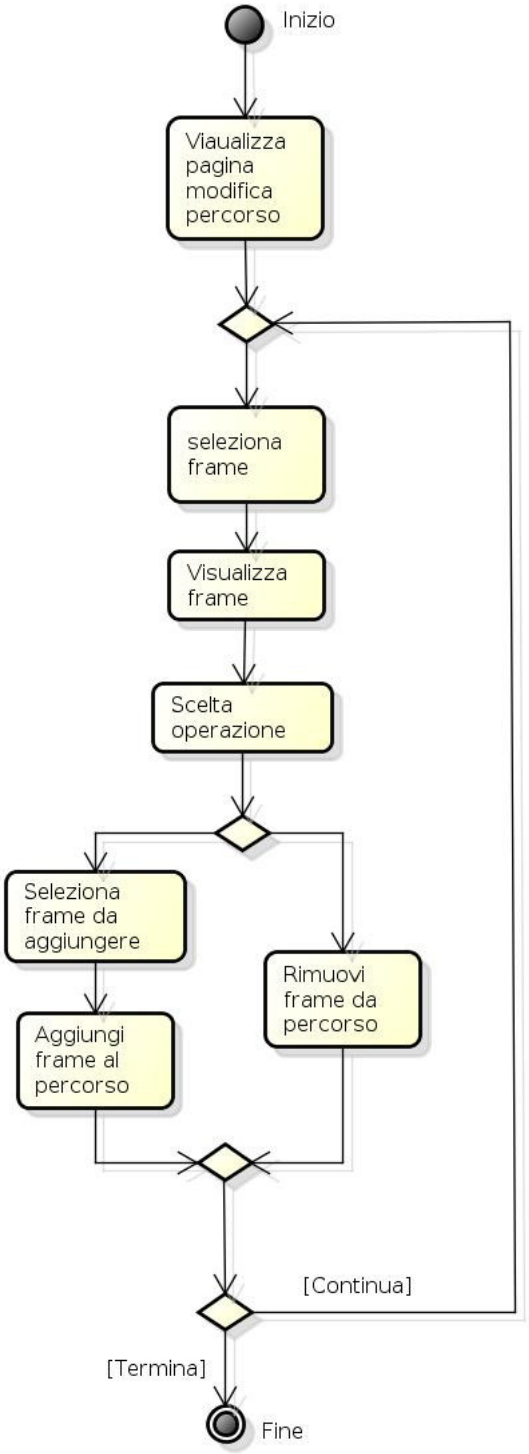

Figura 34: Modifica percorso

Per modificare il percorso l'utente seleziona un frame e questo viene visualizzato nell'editor. Dopodicchè ha la possibilità di rimuovere il frame dal percorso o di selezionarne uno da aggiungerlo al percorso.

6.22 Aggiungi frame

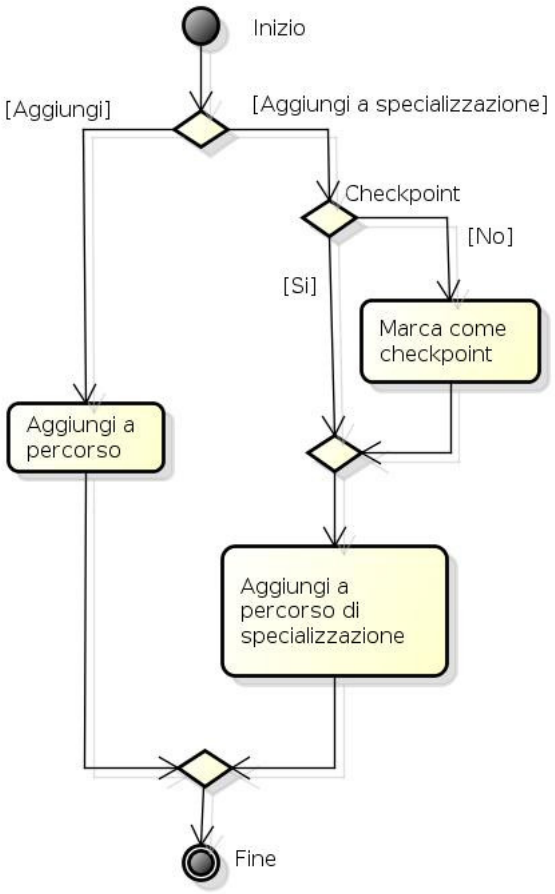

Figura 35: Aggiungi frame

Per aggiungere un frame al percorso si hanno due possibilità:

- Aggiungere il frame in coda al frame visualizzato nell'editor;
- Marcare il frame corrente visualizzato come checkpoint, se non già marcato, e aggiungerlo al percorso di specializzazione.

6.23 Rimuovi frame da percorso

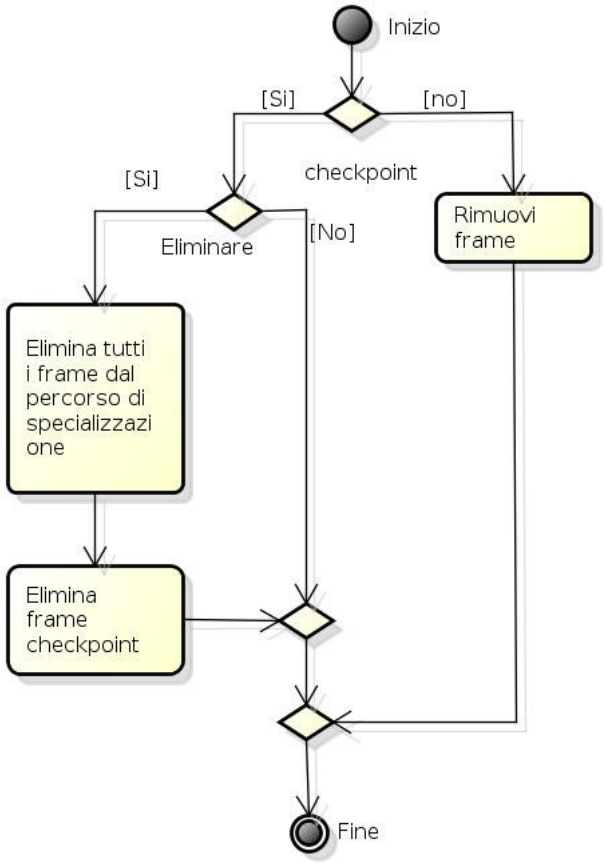

Figura 36: Rimuovi frame dal percorso

Quando l'utente rimuove un frame viene controllato se è un checkpoint. In caso negativo si rimuove il frame dal percorso, viceversa se c'è la conferma dell'utente si eliminano prima tutti i frame del percorso di specializzazione e successivamente si elimina il frame dal percorso.

7 Stime di fattibilità e di bisogno di risorse

8 Tracciamento della relazione componenti - requisiti

9 Design Pattern

9.1 Design Pattern Architetturali

9.1.1 MVC - Model View Controller

- **Descrizione:** Il design pattern$_G$ MVC permette un disaccoppiamento totale della View dalle logiche di manipolazione del Modello tramite l'introduzione di

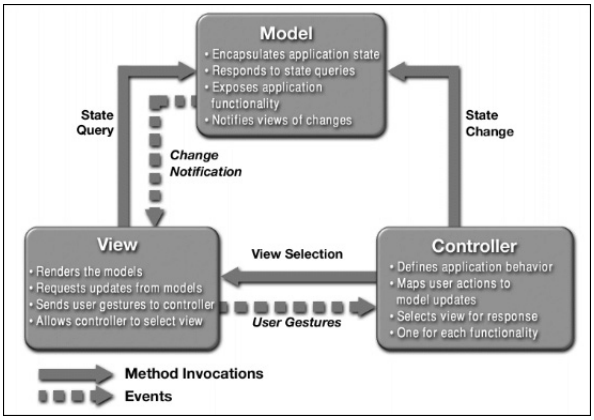

Figura 37: Diagramma del design pattern MVC

un componente, il Controller, che funga da intermediario e da coordinatore in risposta alle interazioni con l'utente. Si individuano tre componenti:

  - Model: dati di business e regole di accesso;
  - View: rappresentazione grafica. Visualizza i dati contenuti nel model e raccoglie gli input dell'utente;
  - Controller: reazioni della UI agli input utente. Interagisce con il model in base ai comandi dell'utente (attraverso la View);

- **Motivazione:** Lo scopo di molte applicazioni è quello di recuperare dati e visualizzarli in maniera opportuna a seconda delle esigenze degli utenti. Poiché il flusso chiave di informazione avviene tra il dispositivo su cui sono memorizzati i dati e l'interfaccia utente, si è portati a legare insieme queste due parti per ridurre la quantità di codice e migliorare le performance dell'applicazione. Questo approccio, apparentemente naturale, presenta alcuni problemi significativi; uno di questi è che l'interfaccia utente tende a cambiare più in fretta rispetto al sistema di memorizzazione dei dati. C'è la necessità, quindi, di rendere modulari le funzionalità dell'interfaccia utente in maniera tale da poter facilmente modificare le singole parti. L'intento del pattern MVC è di disaccoppiare il più possibile tra loro le parti dell'applicazione adibite al controllo, all'accesso ai dati e alla presentazione, apportando diversi vantaggi:

  - indipendenza tra i business data (model) la logica di presentazione (view) e quella di controllo (controller);
  - separazione dei ruoli e delle relative interfacce;
  - viste diverse per il medesimo model;
  - semplice il supporto per nuove tipologie di client: bisogna scrivere la vista ed il controller appropriati riutilizzando il model esistente.

- **Applicablità:** Il pattern MVC può essere utilizzato nei seguenti casi:

  - Quando si vuole trattare un gruppo di oggetti come un oggetto singolo;

- Quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro;
- Quando si vogliono agganciare più View a un Model per fornire più rappresentazioni del Model stesso.

9.1.2 MVVM - Model View VievModel

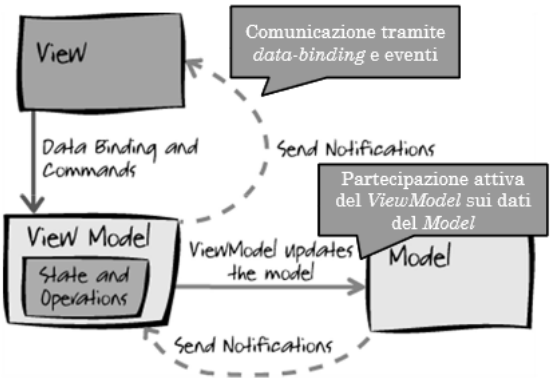

Figura 38: Diagramma del design pattern MVVM

- **Descrizione:** Il design pattern$_G$ MVVM è una variante del pattern MVC che propone un ruolo più attivo della View, la quale è in grado di gestire eventi, eseguire operazioni ed effettuare il data-binding. In questo contesto, quindi, alcune delle funzionalità del Controller vengono inglobate nella View, la quale si appoggia su un'estensione del Model: il ViewModel. Come per il pattern MVC, anche quì si individuano tre componenti:
  - Model: dati di business e regole di accesso;
  - View: rappresentazione grafica. Visualizza i dati contenuti nel model e raccoglie gli input dell'utente;
  - ViewModel: Model esteso con funzionalità per la manipolazione dei dati e per l'interazione con la View.
- **Motivazione:** Il cuore del funzionamento di questo pattern è la creazione di un componente (ViewModel) che rappresenta, in modo astratto, tutte le informazioni e i comportamenti della corrispondente View; quest'ultima si limita a visualizzare graficamente quanto esposto dal ViewModel, a riflettere i propri cambi di stato nel ViewModel stesso oppure ad attivare suoi comportamenti. L'uso di questo pattern richiede la presenza, nella tecnologia di UI, di un'ottimo meccanismo di data-binding (tipicamente bidirezionale) o, in alternativa, di un strato di sincronizzazione fra la View e il ViewModel. E' compito del ViewModel, però, offrire alla View una "superficie esterna" il più possibile ben fruibile, in modo che la sincronizzazione dello stato possa essere fatta senza introdurre logiche decisionali che rendano necessario un test specifico.

- **Applicablità:** Il pattern MVVM può essere utilizzato nei seguenti casi:
  - Quando si vuole trattare un gruppo di oggetti come un oggetto singolo;
  - Quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro;

9.1.3 Dependecy Injection

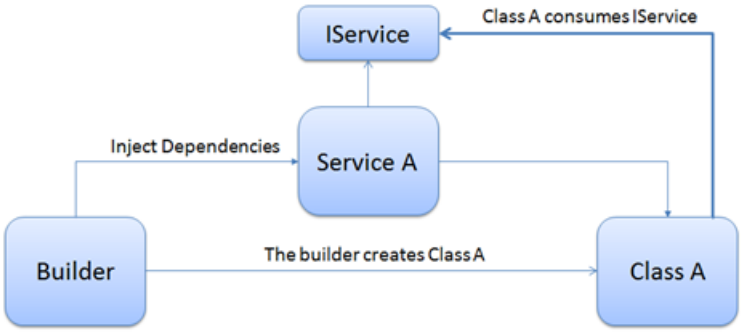

Figura 39: Diagramma del design pattern Dependency Injection

- **Descrizione:** Il design pattern$_G$ Dependecy Injection ha lo scopo di semplificare lo sviluppo e migliorare la testabilità del software, permettendo la separazione del comportamento di una componente dalla risoluzione delle sue dipendenze; Il pattern Dependency Injection coinvolge almeno tre elementi:
  - una componente *dipendente*;
  - la dichiarazione delle *dipendenze* del componente, definite come *interface contracts*;
  - un *injector* (chiamato anche *provider* o *container*) che crea, a richiesta, le istanze delle classi che implementano delle *dependency interfaces*.
- **Motivazione:** Il collegamento di due o più componenti in modo esplicito ne aumenta l'accoppiamento, causando una scarsa manutenibilità del software e complicando le fasi di unit testing. Inoltre un componente soggetto a dipendenze risulta meno predisposto al riutilizzo dello stesso. La dependency injection prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze. Normalmente, senza l'utilizzo di questa tecnica, se un oggetto necessita di accedere ad un particolare servizio, l'oggetto stesso si prende la responsabilità di gestirlo, o avendo un diretto riferimento al servizio, o individuandolo con un Service Locator che gli restituisce un riferimento ad una specifica implementazione del servizio. Con l'utilizzo della dependency injection, l'oggetto ha in sé solamente una proprietà che può ospitare un riferimento a quel servizio e, quando l'oggetto viene istanziato, un riferimento ad una implementazione di questo servizio gli viene iniettata dal framework$_G$ esterno, senza che il programmatore che crea l'oggetto sappia nulla sul posizionamento del servizio o altri dettagli dello stesso.

- **Applicablità:** Il pattern Dependency Injection può essere utilizzato nei seguenti casi:
  - Quando si ha la necessità di collegare più componenti cercando di minimizzare il livello di accoppiamento;
  - Quando si lavora su progetti basati sul Test Driven.

9.2 Design Pattern Creazionali

9.2.1 Factory Method

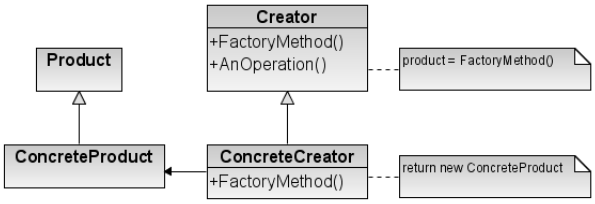

Figura 40: Diagramma del design pattern Factory Method

- **Descrizione:** il design pattern$_G$ Factory Method indirizza il problema della creazione di oggetti senza specificarne l'esatta classe, fornendo un'interfaccia per creare un oggetto, ma lasciando che le sottoclassi decidano quale oggetto istanziare. Definisce un'interfaccia (Creator) per ottenere una nuova istanza di un oggetto (Product). Delega ad una classe derivata (ConcreteCreator) la scelta di quale classe istanziare (ConcreteProduct);
- **Motivazione:** la creazione di un oggetto può, spesso, richiedere processi complessi la cui collocazione all'interno della classe di composizione potrebbe non essere appropriata. Esso può, inoltre, comportare duplicazione di codice, richiedere informazioni non accessibili alla classe di composizione, o non fornire un sufficiente livello di astrazione. Il Factory Method indirizza questi problemi definendo un metodo separato per la creazione degli oggetti. Tale metodo può essere ridefinito dalle sottoclassi per definire il tipo derivato di prodotto che verrà effettivamente creato;
- **Applicablità:** Il pattern Factory Method si può utilizzare nei seguenti casi:
  - Quando si desidera che la creazione di un oggetto non precluda il suo riuso senza una significativa duplicazione di codice;
  - Quando si desidera che la creazione di un oggetto non richieda l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione;
  - Quando si desidera che la gestione del ciclo di vita degli oggetti gestiti debba essere centralizzata in modo da assicurare un comportamento consistente all'interno dell'applicazione.

Riferimenti bibliografici