404NotFound

Premi: better than Prezi.

Specifica Tecnica

Versione	2.0
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	26-05-2015
Lista di distribuzione	404NotFound prof. Tullio Vardanega prof. Riccardo Cardin Zucchetti S.p.a.

Registro delle modifiche

Versione	Autore	Data	Descrizione
1.0	Ismaele Gobbo	26-05-2015	Approvazione del documento
0.28	Monica Manuto	26-05-2015	Verifica documento
0.27	Andrea Rettore	20-05-2015	Verifica documento
0.26	Monica Manuto	19-05-2015	Fine stesura tracciamento requisiti-componenti
0.25	Ismaele Gobbo	19-05-2015	Fine stesura descrizione dei componenti
0.24	Andrea Rettore	18-05-2015	Modifica sezione Metodo e Formalismo
0.23	Ismaele Gobbo	15-05-2015	modifica Diagramma Presentation
0.22	Monica Manuto	14-05-2015	Incremento tracciamento requisiti-componenti
0.21	Ismaele Gobbo	09-05-2015	Incremento descrizione dei componenti
0.20	Andrea Rettore	08-05-2015	Incremento della sezione Tecnologie
0.19	Monica Manuto	04-05-2015	Incremento tracciamento requisiti-componenti
0.18	Ismaele Gobbo	02-05-2015	Incremento descrizione dei componenti
0.17	Ismaele Gobbo	25-04-2015	Inizio stesura descrizione dei componenti
0.16	Andrea Rettore	25-04-2015	Inizio stesura tracciamento requisiti-componenti
0.15	Andrea Rettore	25-04-2015	Inizio stesura tracciamento requisiti-componenti
0.14	Enrico De Lazzari	24-04-2015	Fine stesura definizione di prodotto
0.13	Mattia Cossu	19-04-2015	Incremento definizione di prodotto
0.12	Federico Vegro	16-04-2015	Incremento definizione di prodotto
0.11	Andrea Rettore	14-04-2015	Fine stesura tecnologie utilizzate
0.10	Monica Manuto	14-04-2015	Fine stesura stime di fattibilità
0.9	Monica Manuto	14-04-2015	Fine stesura Design Pattern
0.8	Andrea Rettore	12-04-2015	Incremento Design Pattern
0.7	Monica Manuto	13-04-2015	Inizio stesura Design Pattern
0.6	Andrea Rettore	12-04-2015	Incremento tecnologie utilizzate
0.5	Andrea Rettore	11-04-2015	Incremento stime di fattibilità
0.4	Monica Manuto	10-04-2015	Inizio stesura stime di fattibilità
0.3	Monica Manuto	10-04-2015	Inizio stesura tecnologie utilizzate
0.2	Marco Camborata	10-04-2015	Inizio stesura definizione di prodotto
0.1	Marco Camborata	17-02-2015	Stesura scheletro

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 e agli eventi, solitamente utilizzato per la programmazione di siti web lato client ed interpretato dai browser$_G$. Questo aspetto permette di sollevare dal server il peso della computazione la quale viene eseguita dal client.$_G$. Tale particolarità rappresenta un vantaggio 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 diffuso e ormai consolidato, JavaScript$_G$ può essere eseguito dalla maggior parte dei browser$_G$, sia in ambienti 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 all'utilizzo di 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. AngularJS$_G$ 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. AngularJS$_G$ 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.

- AngularJS$_G$ 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, AngularJS$_G$ 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 AngularJS$_G$ 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, ne 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$_G$ 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.

3.3 Server

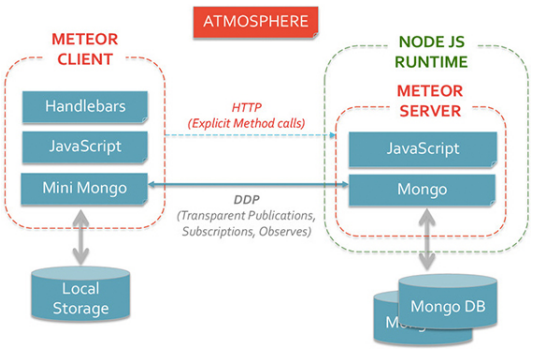

Figura 1: Schema architettura

La comunicazione con la componente server avviene tramite la libreria Node.js$_G$ già implementata in MeteorJS$_G$. Con questa componente l'architettura permette di realizzare un'applicazione asincrona, che a differenza del modello classico client-server permette di eseguire operazioni utili durante l'attesa di ricevere o di modificare un dato richiesto. Nel server sono presenti delle funzioni che permettono al client di reperire i dati dal database MongoDB$_G$ residente nel server. In locale vengono scaricati solo i file richiesti che servono in quel momento. Ogni modifica ai dati viene effettuata prima nel database locale Minimongo$_G$; solo successivamente MeteorJS esegue in automatico in background la sincronizzazione tra Minimongo$_G$ e MongoDB$_G$.

4 Diagramma dei Package

Di seguito vengono rappresentati i componenti principali del sistema e le loro dipendenze.
Package contenenti parti del prodotto relative alla componente *Model* sono stati colorati di rosso; quelli relativi alla componente *Controller* o *ViewModel* sono stati colorati di verde, mentre quelli che racchiudono i template delle *View* sono stati colorati di arancio. Questa scelta ha il solo scopo di facilitare la comprensione della struttura del sistema e non si basa su alcun standard UML$_G$.

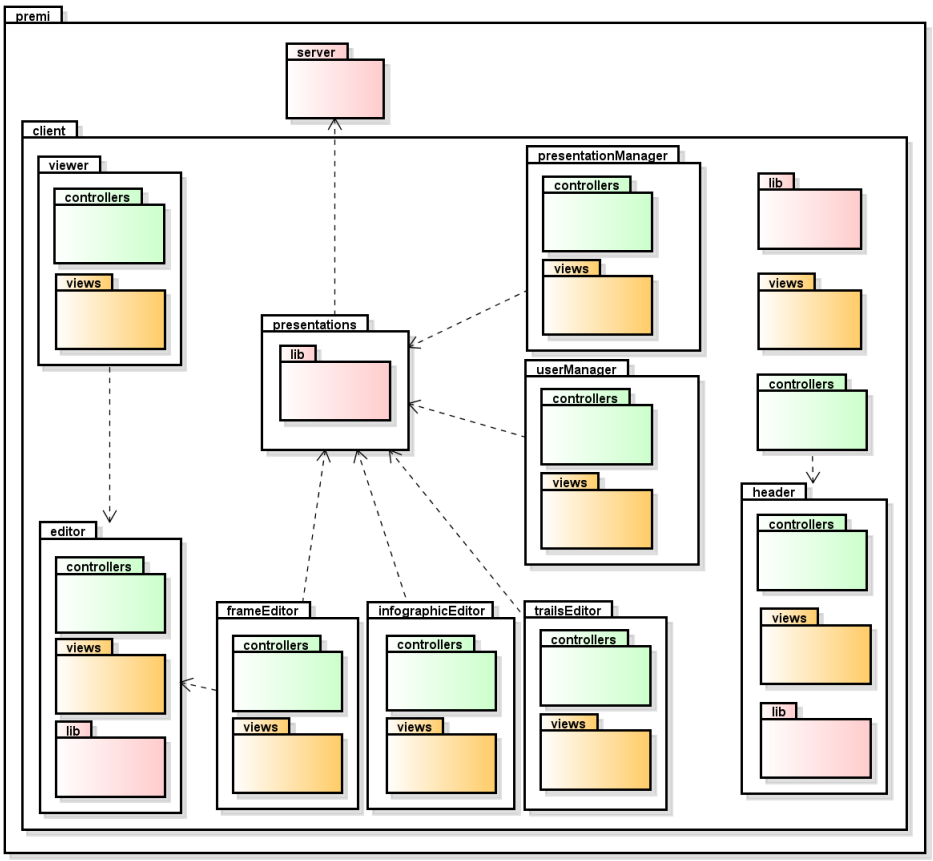

Figura 2: Diagramma dei package di Premi.

L'applicazione è costituita da un solo package$_G$ principale chiamato `premi`; al suo interno sono presenti:

- `premi.server` contiene una libreria di metodi per l'inserimento, l'aggiornamento e la rimozione dei dati presenti nel database$_G$;
- `premi.client` è il package principale che gestisce le funzionalità offerte all'utente. Contiene al suo interno tutti i package relativi al lato client dell'applicazione, tra cui anche il template$_G$ ed il controller$_G$ della vista principale, e le librerie esterne adottate per la realizzazione di alcune parti dell'applicazione;
- `premi.client.header` contiene i files necessari alla creazione e alla visualizzazione dell'header$_G$ dell'applicazione;

- `premi.client.presentation` contiene un'interfaccia per l'utilizzo dei metodi di inserimento, aggiornamento e rimozione dei dati presenti nel server; contiene anche delle classi per la gestione delle presentazioni dell'utente;
- `premi.client.presentationManager` consente all'utente di creare, modificare o eliminare le presentazioni;
- `premi.client.editor` è lo scheletro dell'editor delle presentazioni. Al suo interno sono presenti le classi che modellano gli elementi che compongono la presentazione;
- `premi.client.frameEditor` è la parte di editor che si occupa di creare, modificare o cancellare i Frame$_G$ contenuti nella presentazione;
- `premi.client.infographicEditor` è a parte di editor che permette il posizionamento dei Frame$_G$ o di altri elementi all'interno di un poster;
- `premi.client.trailsEditor` è la parte di editor che permette all'utente di ordinare i Frame per la creazione di uno o più percorsi di presentazione;
- `premi.client.userManager` è il package$_G$ di gestione dei dati dell'utente; fornisce le procedure per la registrazione, il login, il cambio di password, ecc;
- `premi.client.viewer` racchiude gli elementi necessari alla visualizzazione della presentazione nei vari contesti previsti(presentazione live, pubblica e privata).

5 Descrizione dei singoli componenti

5.0.1 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.1 premi/server

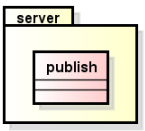

Figura 3: Diagramma del package premi/server

5.1.1 premi/server/publish

Nome: publish

Tipo: class

Package: premi/server

Descrizione: questa classe fornisce al lato client dell'applicazione dei metodi per l'inserimento, l'aggiornamento e la rimozione dei dati del database, e pubblica all'utente solo ed esclusivamente le informazioni a cui lui ha accesso

5.2 premi/client

5.2.1 premi/client/views/home.ng

Nome: home.ng

Tipo: template

Package: premi/client/views

Descrizione: template della pagina principale dell'applicazione

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/controllers/homeCtrl per l'aggiornamento in tempo reale dei dati visualizzati e modificati

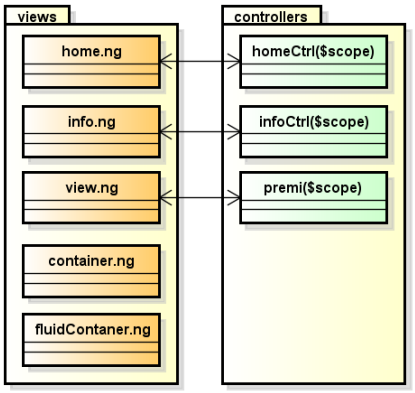

Figura 4: Diagramma dei package views e controllers di premi

5.2.2 premi/client/views/info.ng

Nome: info.ng

Tipo: template

Package: premi/client/views

Descrizione: template della view che mostra all'utente informazioni sull'applicazione nella pagina principale

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/controllers/infoCtrl per l'aggiornamento in tempo reale dei dati visualizzati e modificati

5.2.3 premi/client/views/view.ng

Nome: view.ng

Tipo: template

Package: premi/client/views

Descrizione: template della view di supporto all'header della pagina principale dell'applicazione

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/controllers/premi per la gestione dell'header

5.2.4 premi/client/views/container.ng

Nome: container.ng

Tipo: template

Package: premi/client/views

Descrizione: template contenitore di supporto ad altre viste

5.2.5 premi/client/views/fluidContainer.ng

Nome: fluidContainer.ng

Tipo: template

Package: premi/client/views

Descrizione: template contenitore di supporto ad altre viste

5.2.6 premi/client/controllers/homeCtrl

Nome: homeCtrl

Tipo: controller

Package: premi/client/controllers

Descrizione: controller di premi/client/views/home.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/views/home.ng

5.2.7 premi/client/controllers/infoCtrl

Nome: infoCtrl

Tipo: controller

Package: premi/client/controllers

Descrizione: controller di premi/client/views/info.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/views/info.ng

5.2.8 premi/client/controllers/premi

Nome: premi

Tipo: controller

Package: premi/client/controllers

Descrizione: controller di premi/client/views/view.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/views/info.ng e dipende anche da:

- *premi/client/header/lib/Header* per la gestione dell'header della pagina principale

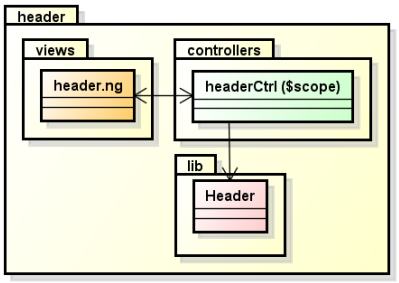

Figura 5: Diagramma del package premi/client/header

5.3 premi/client/header

5.3.1 premi/client/header/views/header.ng

Nome: header.ng

Tipo: template

Package: premi/client/header/views

Descrizione: template dell'header della pagina principale dell'applicazione

Relazioni con altri componenti: a view generata da questo template è collegata allo *$scope* di premi/client/header/controllers/headerCtrl per l'aggiornamento in tempo reale dell'header

5.3.2 premi/client/header/controllers/headerCtrl

Nome: headerCtrl

Tipo: controller

Package: premi/client/header/controllers

Descrizione: controller di premi/client/header/views/header.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/header/views/header.ng e dipende anche da:

- *premi/client/header/lib/Header* per la creazione dell'header della pagina principale

5.3.3 premi/client/header/lib/Header

Nome: Header

Tipo: class

Package: premi/client/header/lib

Descrizione: classe dell'header della pagina principale di Premi. Fornisce dei metodi per gestire la propria visualizzazione e per quella dello username, dello stato dell'applicazione e della barra di navigazione

5.4 premi/client/presentation

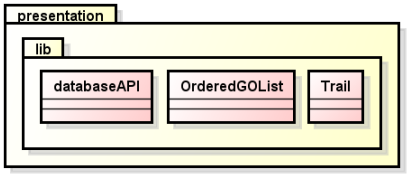

Figura 6: Diagramma del package premi/client/presentation

5.4.1 premi/client/presentation/lib/databaseAPI

Nome: databaseAPI

Tipo: class

Package: premi/client/presentation/lib/

Descrizione: estende i metodi di premi/server/publish e li specializza per i bisogni del client

Relazioni con altri componenti: dipende da:

- *premi/server/publish* per derivare i metodi di gestione dei dati lato client

5.4.2 premi/client/presentation/lib/OrderedGOList

Nome: OrderedGoList

Tipo: class

Package: premi/client/presentation/lib/

Descrizione: classe che modella una lista ordinata di oggetti grafici

5.4.3 premi/client/presentation/lib/Trail

Nome: Trail

Tipo: class

Package: premi/client/presentation/lib/

Descrizione: classe che modella un Trail, ossia un percorso di presentazione. Deve poter fornire i metodi per scorrere la presentazione e inserire o rimuovere Frame e checkpoint

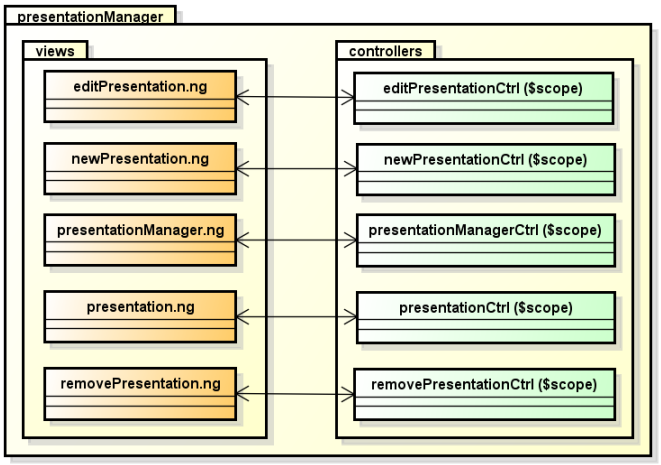

Figura 7: Diagramma del package premi/client/presentation

5.5 premi/client/presentationManager

5.5.1 premi/client/presentationManager/views/editPresentation.ng

Nome: editPresentation.ng

Tipo: template

Package: premi/client/presentationManager/views/

Descrizione: template della parte di pagina che offre all'utente la possibilità di modificare una presentazione

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/presentationManager/controllers/editPresentationCtrl per la modifica di una presentazione

5.5.2 premi/client/presentationManager/views/newPresentation.ng

Nome: newPresentation.ng

Tipo: template

Package: premi/client/presentationManager/views/

Descrizione: template della parte di pagina che offre all'utente la possibilità di creare una nuova presentazione

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/presentationManager/controllers/newPresentationCtrl per l'aggiunta di una presentazione vuota nel database di proprietà dell'utente

5.5.3 premi/client/presentationManager/views/presentationManager.ng

Nome: presentationManager.ng

Tipo: template

Package: premi/client/presentationManager/views/

Descrizione: template dello scheletro della pagina di gestione delle presentazioni dell'utente

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/presentationManager/controllers/newPresentationCtrl

5.5.4 premi/client/presentationManager/views/presentations.ng

Nome: presentations.ng

Tipo: template

Package: premi/client/presentationManager/views/

Descrizione: template della parte di pagina che mostra all'utente la lista delle sue presentazioni

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/presentationManager/controllers/presentationsCtrl per accedere alla lista delle presentazioni

5.5.5 premi/client/presentationManager/views/removePresentation.ng

Nome: removePresentation.ng

Tipo: template

Package: premi/client/presentationManager/views/

Descrizione: template della parte di pagina che offre all'utente la possibilità di eliminare una presentazione

Relazioni con altri componenti: la view generata da questo template è collegata allo *$scope* di premi/client/presentationManager/controllers/removePresentationCtrl per rimuovere una presentazione dal database

5.5.6 premi/client/presentationManager/controllers/editPresentationCtrl

Nome: editPresentationCtrl

Tipo: controller

Package: premi/client/presentationManager/controllers

Descrizione: controller di premi/client/presentationManager/views/editPresentation.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/presentationManager/views/editPresentation.ng e dipende anche da:

- *premi/client/presentation/lib/databaseAPI* per l'accesso al database per la modifica dei campi dati della presentazione

5.5.7 premi/client/presentationManager/controllers/newPresentationCtrl

Nome: newPresentationCtrl

Tipo: controller

Package: premi/client/presentationManager/controllers/

Descrizione: controller di premi/client/presentationManager/views/newPresentation.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/presentationManager/views/newPresentation.ng e dipende anche da:

- *premi/client/presentation/lib/databaseAPI* per l'accesso al database per l'aggiunta di una nuova presentazione

5.5.8 premi/client/presentationManager/controllers/PresentationManagerCtrl

Nome: presentationManagerCtrl

Tipo: controller

Package: premi/client/presentationManager/controllers

Descrizione: controller di premi/client/presentationManager/views/presentationManager.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/presentationManager/views/presentationManager.ng

5.5.9 premi/client/presentationManager/controllers/presentationsCtrl

Nome: presentationsCtrl

Tipo: controller

Package: premi/client/presentationManager/controllers

Descrizione: controller di premi/client/presentationManager/views/presentationManager.ng, fornisce alla vista la lista delle presentazioni dell'utente pubblicate dal server

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/presentationManager/views/presentations.ng

5.5.10 premi/client/presentationManager/controllers/removePresentationCtrl

Nome: removePresentationCtrl

Tipo: controller

Package: premi/client/presentationManager/controllers

Descrizione: controller di premi/client/presentationManager/views/removePresentation.ng, fornisce alla vista dei metodi per la rimozione della presentazione selezionata

Relazioni con altri componenti: modella lo *$scope* per interagire con la view generata da premi/client/presentationManager/views/removePresentation.ng e dipende anche da:

- *premi/client/presentation/lib/databaseAPI* per l'accesso al database per la rimozione della presentazione selezionata

5.6 Premi/client/editor

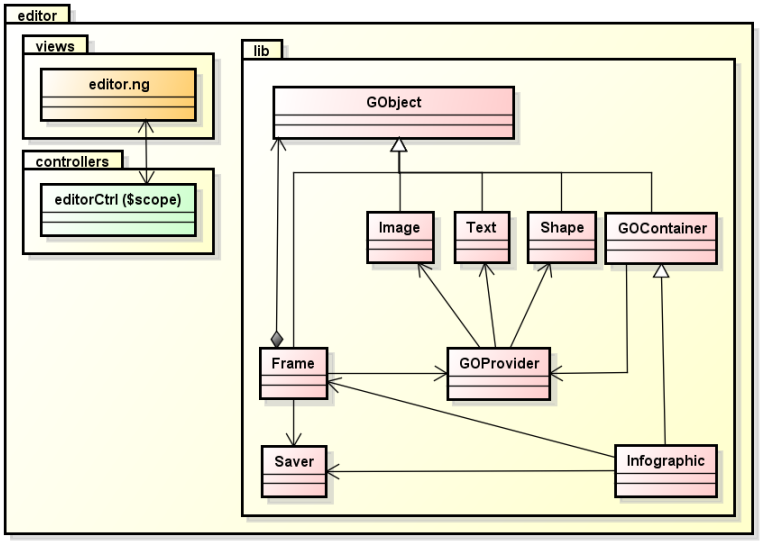

Figura 8: Diagramma del package premi/client/editor

5.6.1 Premi/client/editor/lib/gObject

Nome: gObject

Tipo: *abstract class*

Package: Premi/client/editor/lib

Descrizione: rappresenta gli oggetti grafici nella presentazione

5.6.2 Premi/client/editor/lib/gOContainer

Nome: GOContainer

Tipo: *abstract class*

Package: Premi/client/editor/lib

Descrizione: rappresenta gli oggetti grafici che possono essere contenuti in un frame

Relazioni con altri componenti: estende `Premi/client/editor/gObject` e dipende anche da:

- *Premi/client/editor/lib/image* per interagire con gli oggetti di tipo image;
- *Premi/client/editor/lib/text* per interagire con gli oggetti di tipo testo;
- *Premi/client/editor/lib/shape* per interagire con gli oggetti di tipo shape;

5.6.3 Premi/client/editor/lib/text

Nome: text

Tipo: class

Package: Premi/client/editor/lib

Descrizione: rappresenta un'area di testo nella presentazione

Relazioni con altri componenti: estende `Premi/client/editor/gObject`

5.6.4 Premi/client/editor/lib/image

Nome: image

Tipo: class

Package: Premi/client/editor/lib

Descrizione: rappresenta un'immagine nella presentazione

Relazioni con altri componenti: estende `Premi/client/editor/gObject`

5.6.5 Premi/client/editor/lib/shape

Nome: shape

Tipo: class

Package: Premi/client/editor/lib

Descrizione: rappresenta una figura nella presentazione. Uno shape può avere forme diverse come un quadrato, un cerchio, una freccia. Può diventare un elemento di abbellimento o di aumento dell' informazione che si vuole rappresentare.

Relazioni con altri componenti: estende `Premi/client/editor/gObject`

5.6.6 Premi/client/editor/lib/frame

Nome: frame

Tipo: class

Package: Premi/client/editor/lib

Descrizione: rappresenta un frame$_G$ nella presentazione

Relazioni con altri componenti: estende `Premi/client/editor/gObject` e contiene un insieme di oggetti `Premi/client/editor/gObject`. Nonostante la classe frame$_G$ estenda gObject, un frame$_G$ non può contenere altri frame$_G$. Tuttavia si è scelto di lasciare che un frame$_G$ possa contenere tutti gli gObject perchè si prevede in futuro che un frame$_G$ potrà contenere altri frame$_G$. Dipende anche da:

- *Premi/client/editor/lib/image* per interagire con gli oggetti di tipo image;
- *Premi/client/editor/lib/text* per interagire con gli oggetti di tipo testo;
- *Premi/client/editor/lib/shape* per interagire con gli oggetti di tipo shape;
- *Premi/client/editor/lib/saver* per interagire con il database e salvare e modificare gli oggetti di un frame;

5.6.7 Premi/client/editor/lib/infographic

Nome: infographic

Tipo: class

Package: Premi/client/editor/lib

Descrizione: rappresenta l'infografica di una presentazione

Relazioni con altri componenti: estende `Premi/client/editor/gOContainer` e dipende da:

- *Premi/client/editor/lib/frame* per interagire con gli oggetti di tipo frame;
- *Premi/client/editor/lib/saver* per interagire con il database, salvare e modificare gli oggetti dell'infografica;

5.6.8 Premi/client/editor/lib/saver

Nome: saver

Tipo: class

Package: Premi/client/editor/lib

Descrizione: classe che permette di effettuare le modifiche degli oggetti sul database

Relazioni con altri componenti: Dipende da:

- *Premi/client/editor/lib/saver* per interagire con il database.

5.7 Premi/client/frameEditor

5.7.1 Premi/client/frameEditor/views/frame.ng

Nome: frame.ng

Tipo: class

Package: Premi/client/frameEditor/views

Descrizione: template della parte di pagina che permette la modifica dei Frame_G e degli oggetti in esso contenuti.

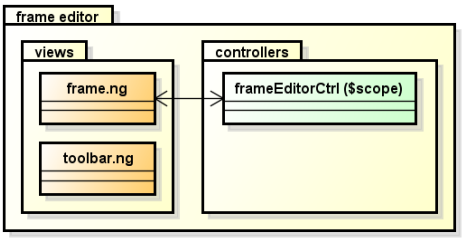

Figura 9: Diagramma del package premi/client/frameEditor

5.7.2 Premi/client/frameEditor/views/toolbar.ng

Nome: toolbar.ng

Tipo: class

Package: Premi/client/frameEditor/views

Descrizione: template della parte di pagina che mostra la toolbar che contiene tutte le funzionalità disponibili per la modifica di un frame e degli oggetti in esso contenuti.

5.7.3 Premi/client/frameEditor/controllers/frameEditorCtrl

Nome: FrameEditorCtrl

Tipo: class

Package: Premi/client/frameEditor/controllers

Descrizione: controller di premi/client/frameEditor/views/toolbar.ng e di premi/client/frameEditor/views/frame.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con le views generate da premi/client/frameEditor/views/toolbar.ng e premi/client/frameEditor/views/frame.ng dipende anche da:

- *Premi/client/presentation/lib/databaseAPI* per interagire con il database;
- *Premi/client/editor/lib/interactInit* per interagire con la libreria Interactjs;
- *Premi/client/editor/lib/frame* per interagire con la libreria frame e usare i metodi per la modifica, aggiunta, cancellazione di un frame;
- *Premi/client/editor/lib/Observer* per interagire con la libreria observer;
- *Premi/presentation/lib/orderedGOList* per interagire con la libreria orderedGOList per ordinare la lista dei frame.

5.8 Premi/client/infographicEditor

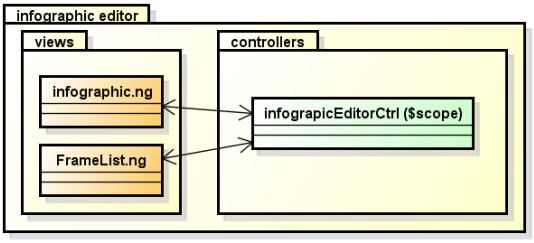

Figura 10: Diagramma del package premi/client/infographicEditor

5.8.1 Premi/client/infographicEditor/views/infographic.ng

Nome: infographic.ng

Tipo: class

Package: Premi/client/infographicEditor/views

Descrizione: template della parte di pagina per la creazione o modifica dell'infografica$_G$

5.8.2 Premi/client/infographicEditor/views/frameList.ng

Nome: frameList.ng

Tipo: class

Package: Premi.client.InfographicEditor.views

Descrizione: template della parte di pagina che mostra la lista dei frame creati che possono essere inseriti nell'infografica.

5.8.3 Premi/client/infographicEditor/controllers/infographicEditorCtrl

Nome: infographicEditorCtrl

Tipo: class

Package: Premi/client/infographicEditor/controllers

Descrizione: controller di premi/client/infographicEditor/views/frameList.ng e premi/client/infographicEditor/views/infographic.ng

Relazioni con altri componenti: modella lo *$scope* per interagire con le views generate da Premi.client.InfographicEditor.views.frame.ng e di Premi.client.InfographicEditor.vi e dipende da:

Premi/client/presentation/lib/databaseAPI per interagire con il database;

Premi/client/editor/lib/interactInit per interagire con la libreria interactjs;

Premi/client/editor/lib/infographic per interagire con la libreria infografica e usare i metodi per la gestione dell'infografica;

Premi/client/editor/lib/observer per interagire con la libreria observer;

Premi/presentation/lib/orderedGOList per interagire con la libreria orderedGOList per ordinare la lista dei frame.

5.8.4 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.5 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.6 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.7 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.8 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.9 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

5.8.10 premi/client

Nome:

Tipo:

Package:

Descrizione:

Relazioni con altri componenti:

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

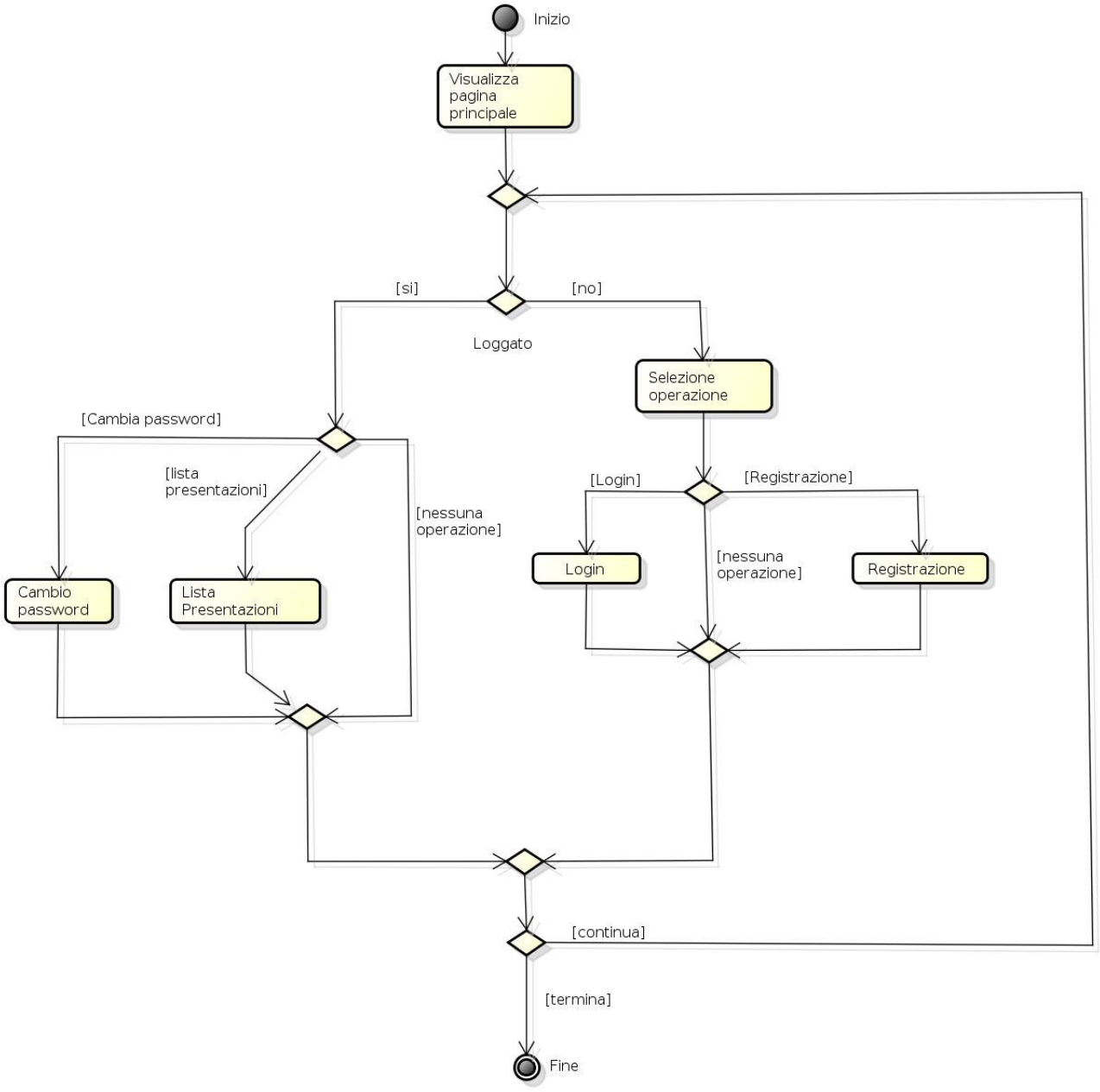

Figura 11: Attività principali

L'utente nel momento in cui accede al 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

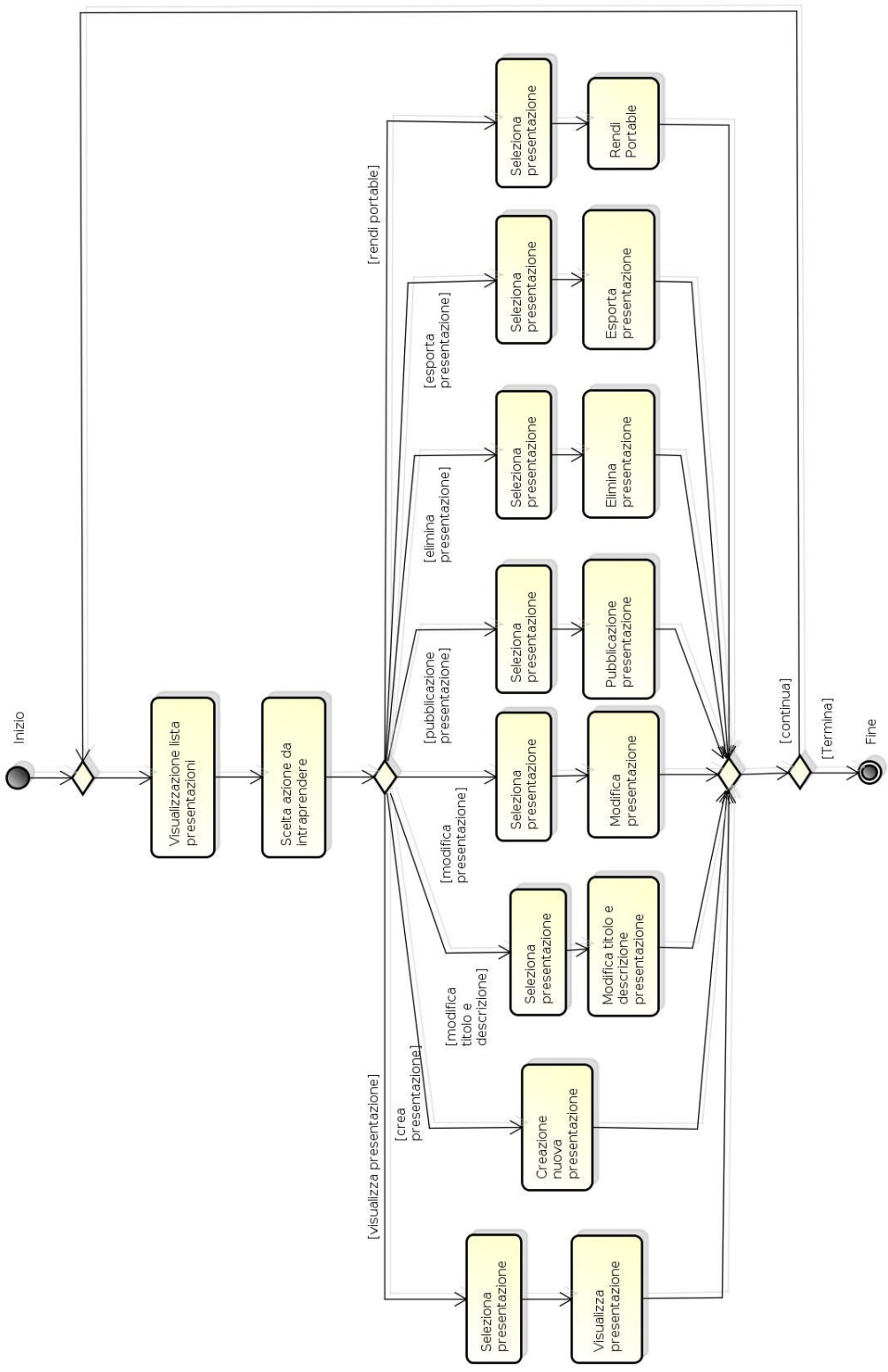

Figura 12: Lista presentazioni

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

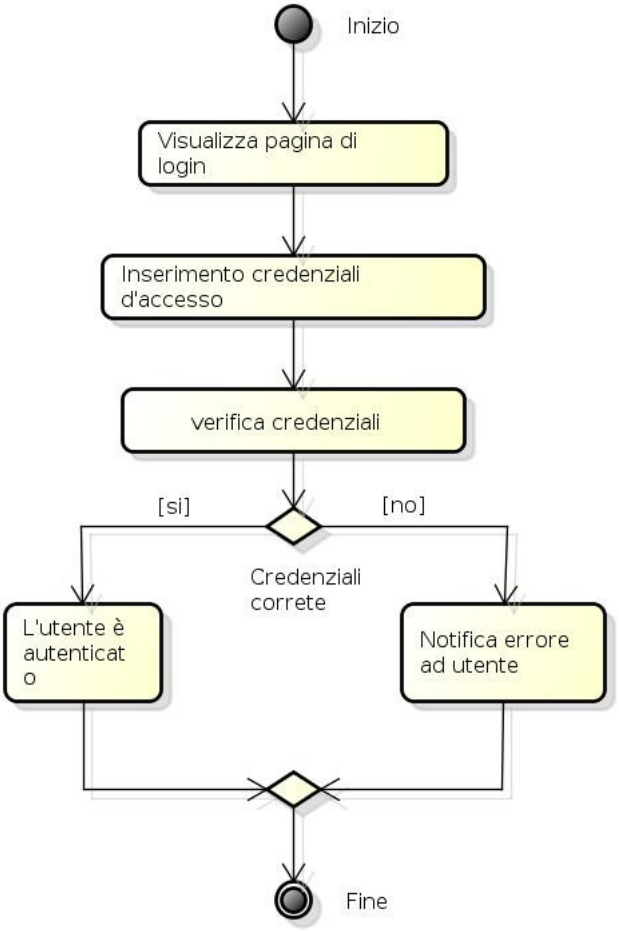

Figura 13: Login utente

L'utente quando accede alla 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

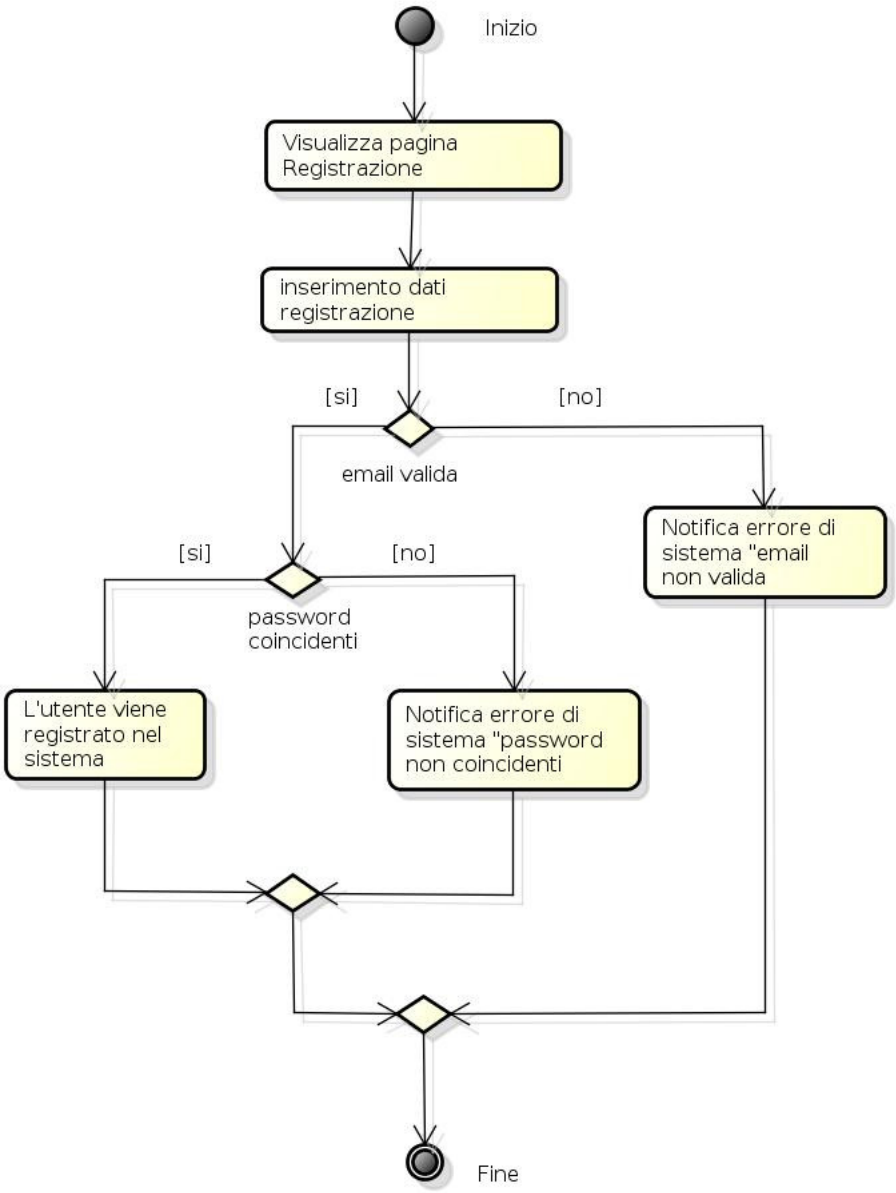

Figura 14: Registrazione utente

L'utente quando accede alla parte di registrazione inserisce l'email, la password e la conferma di quest'ultima. 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

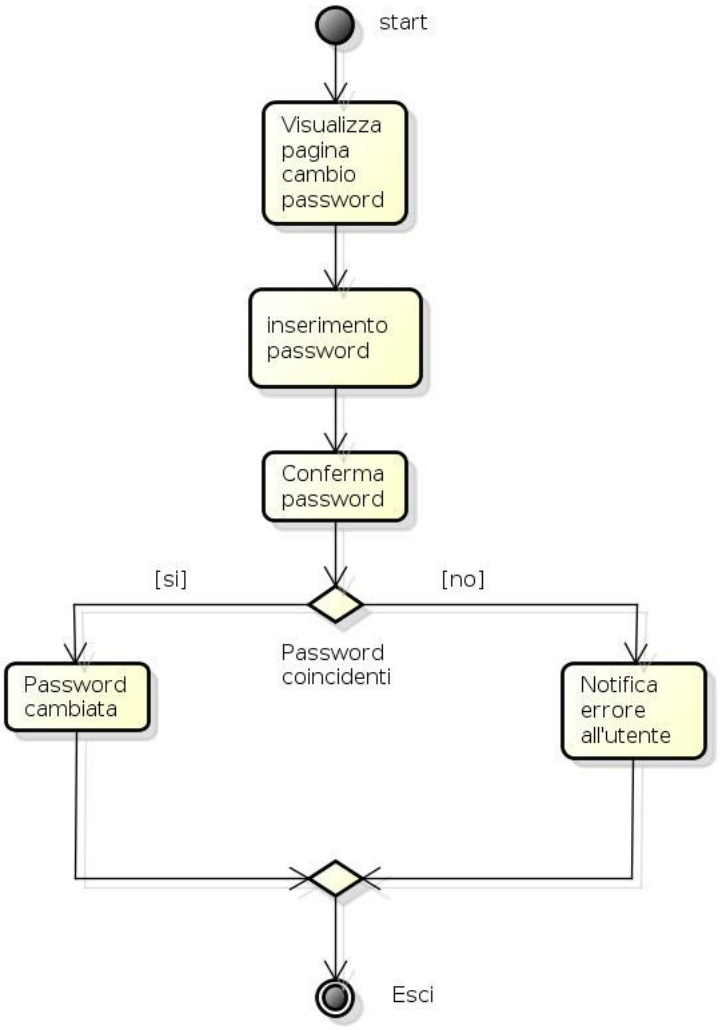

Figura 15: Cambio password

Per effettuare il cambio password l'utente inserisce la nuova password e la sua conferma. Se le due password coincidono viene effettuato il cambio password altrimenti viene notificato un errore all'utente.

6.6 Visualizzatore

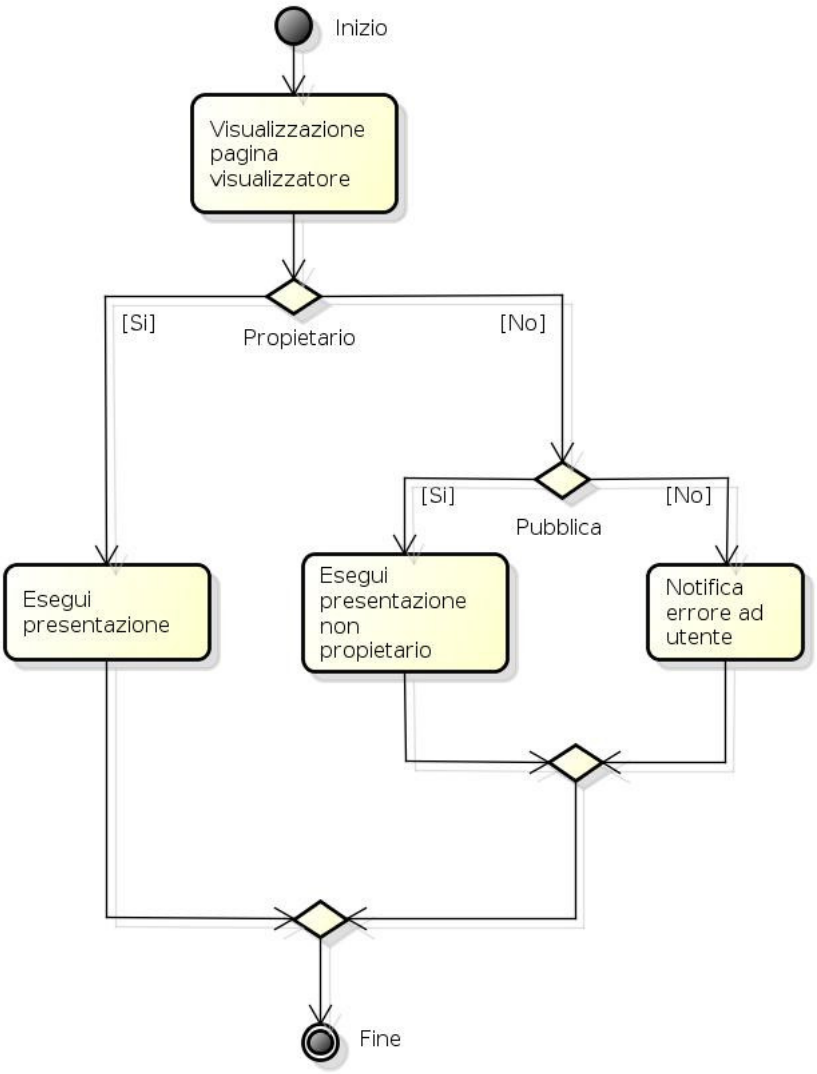

Figura 16: Visualizzatore

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 è importante il controllo sullo stato della presentazione (se pubblica o privata) per verificare la validità del link ad essa associato.

6.7 Esegui presentazione proprietario

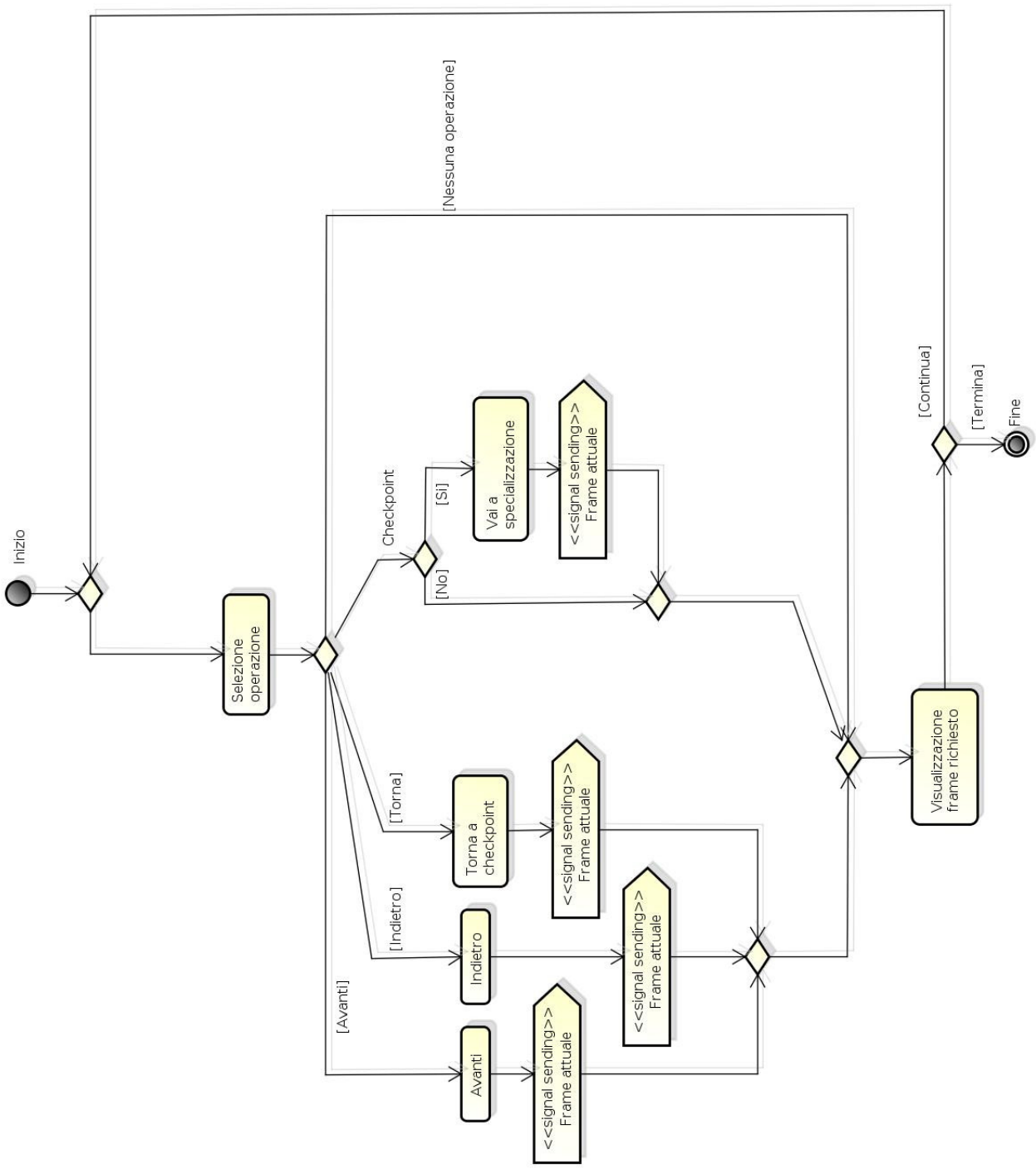

Figura 17: Esegui presentazione proprietario

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

- avanti: per andare avanti di un 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

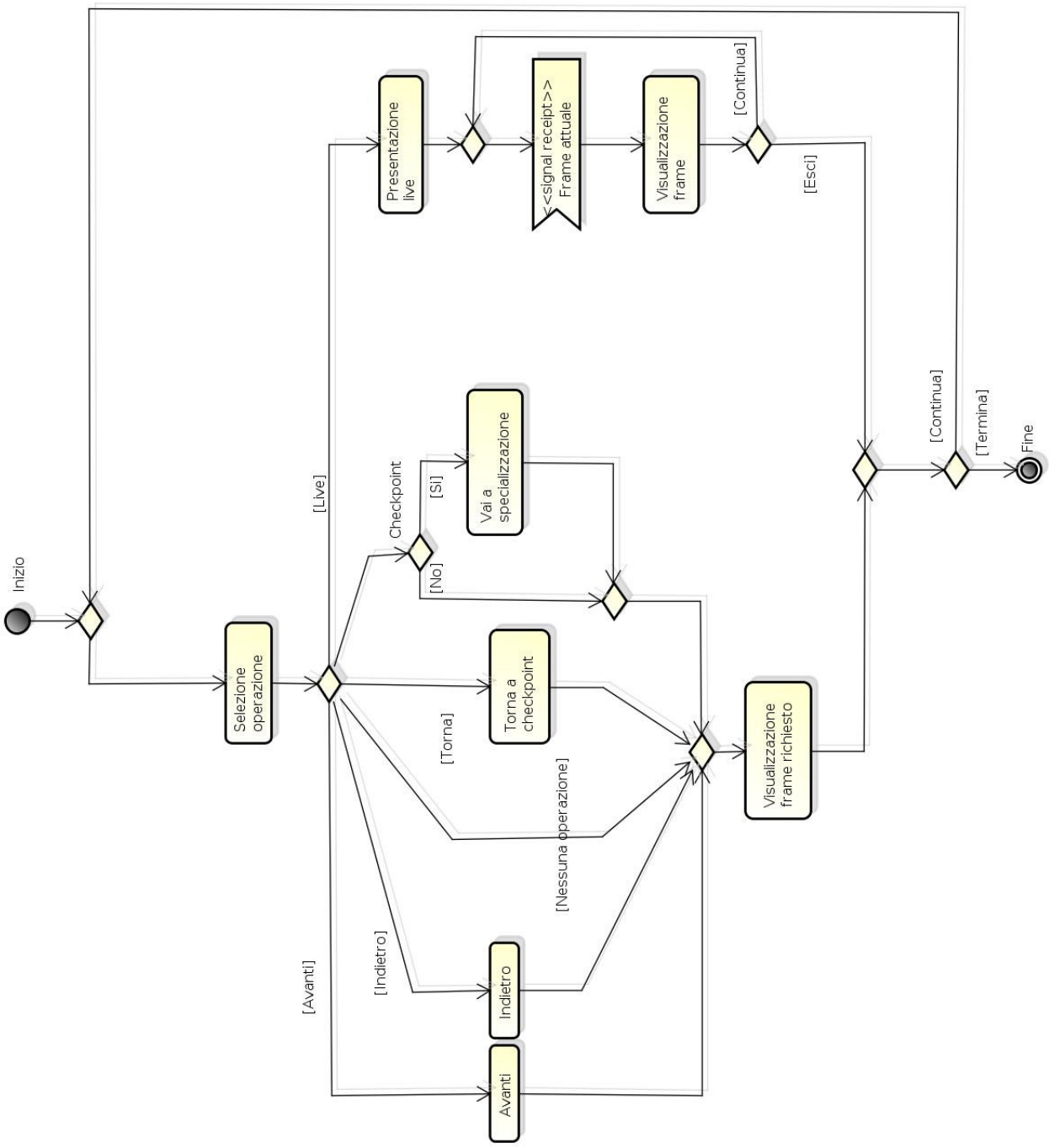

Figura 18: Esegui presentazione non proprietario

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

- avanti: per andare avanti di un 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

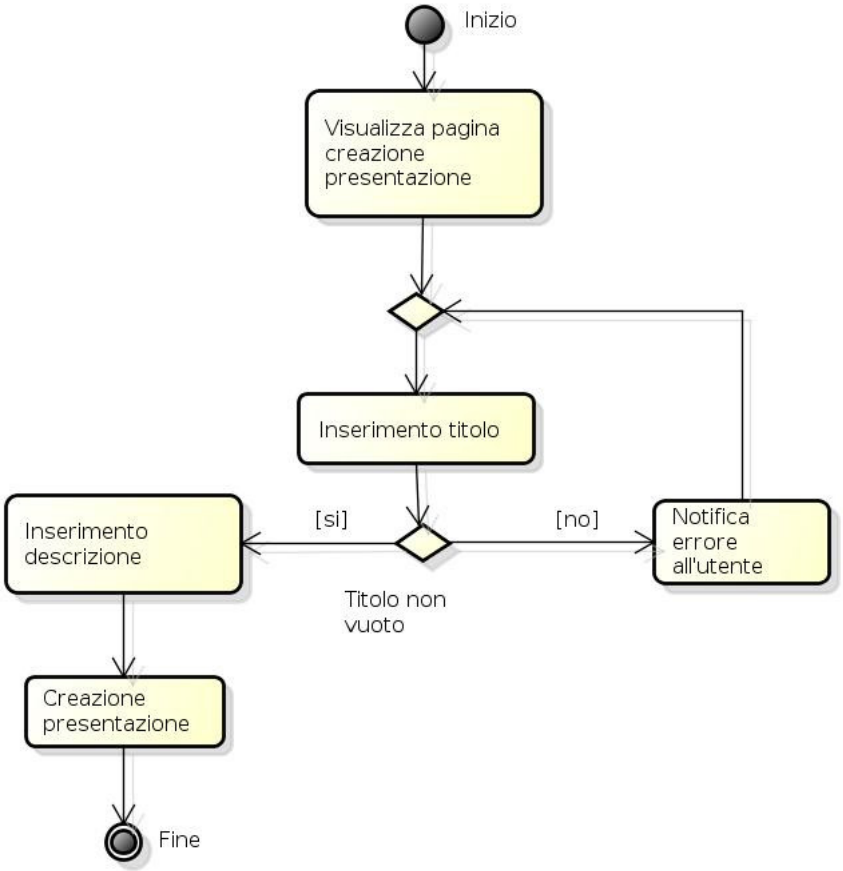

Figura 19: Creazione presentazione

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

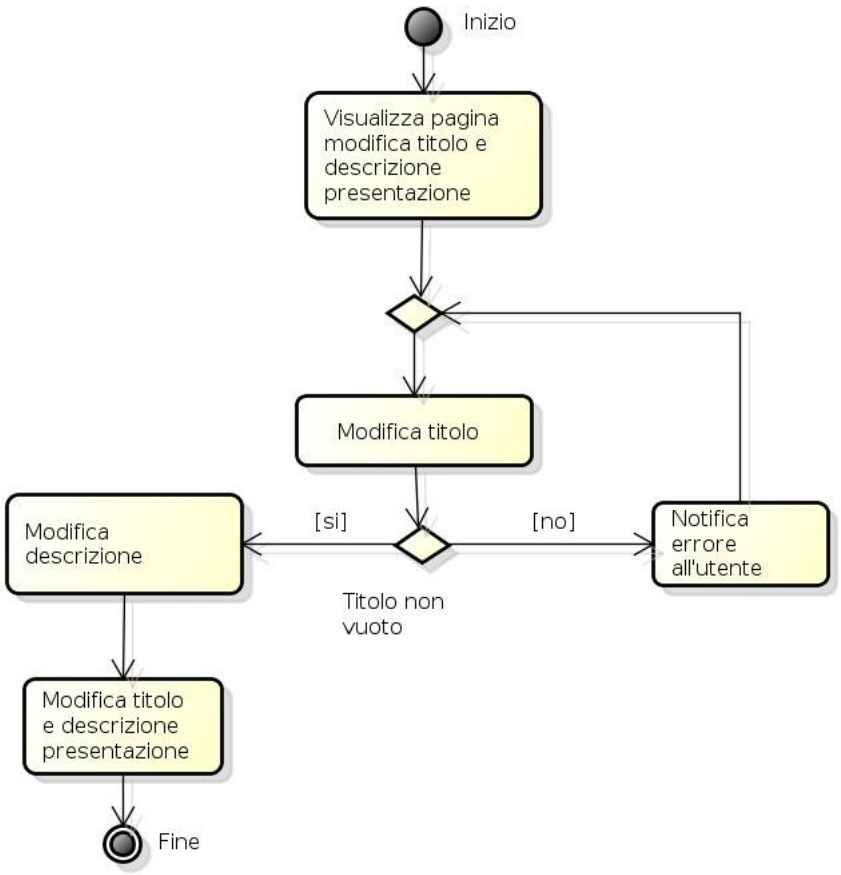

Figura 20: Modifica titolo e descrizione della presentazione

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

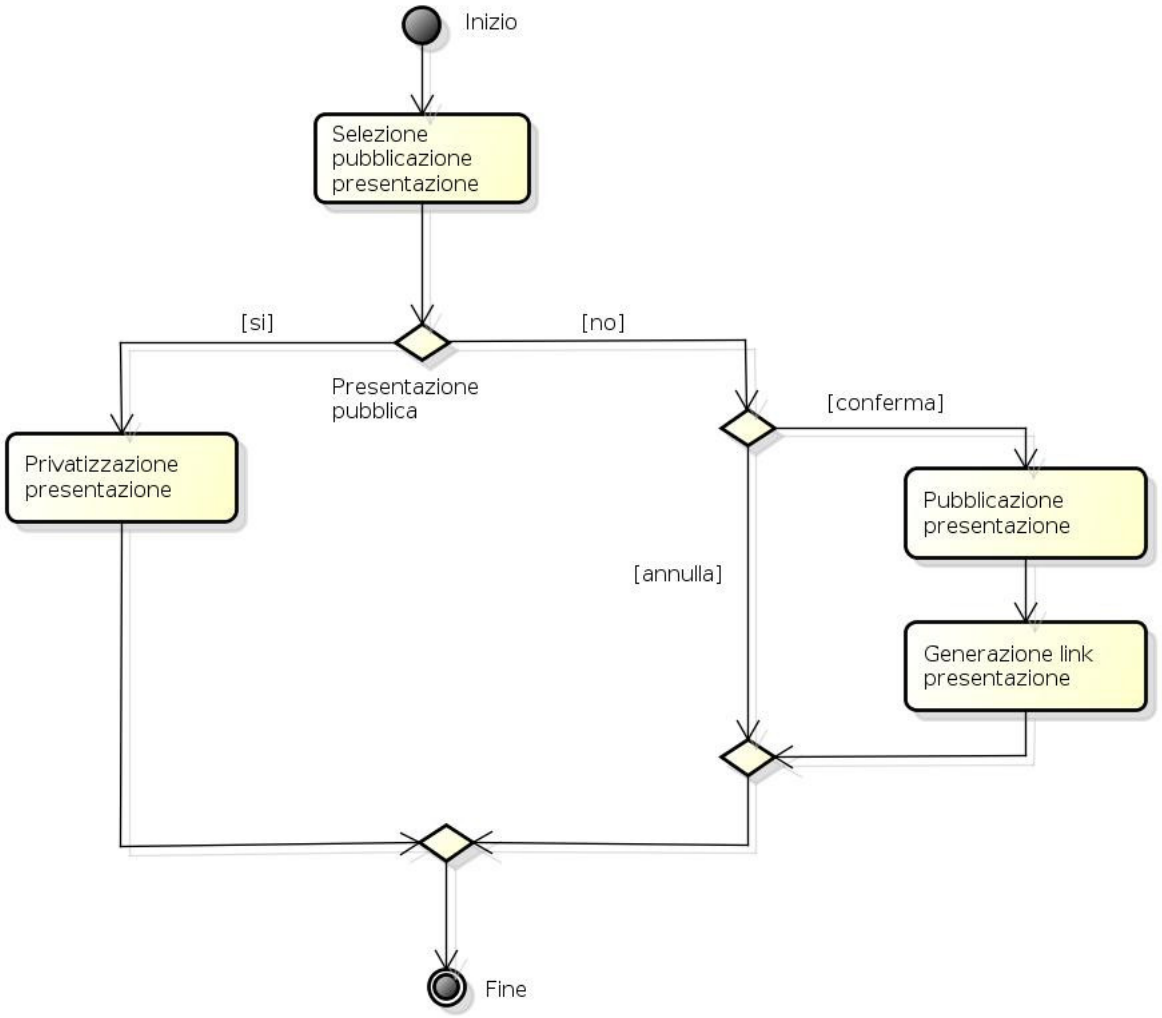

Figura 21: Pubblicazione presentazione

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

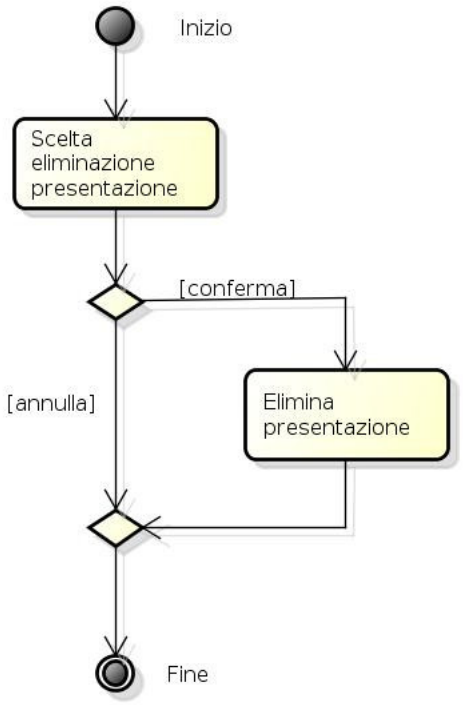

Figura 22: Elimina presentazione

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

6.13 Esportazione presentazione

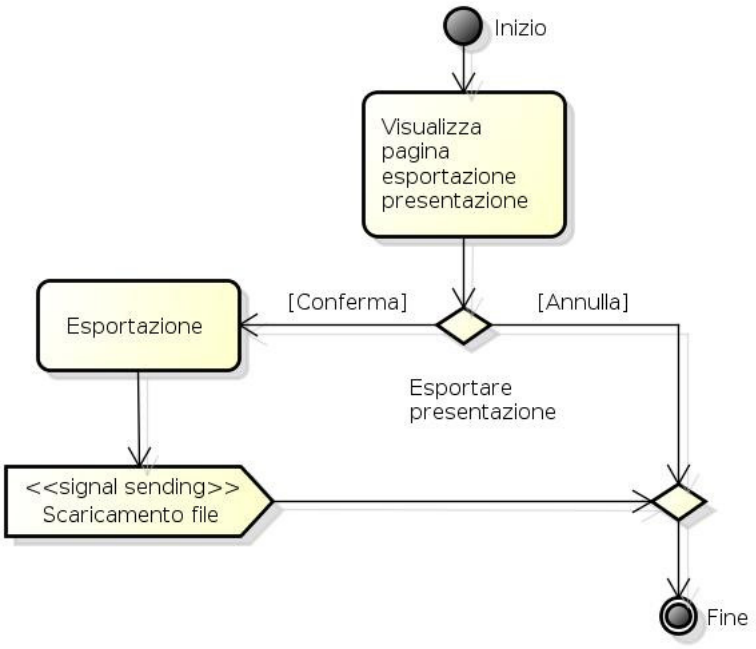

Figura 23: Esportazione presentazione

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

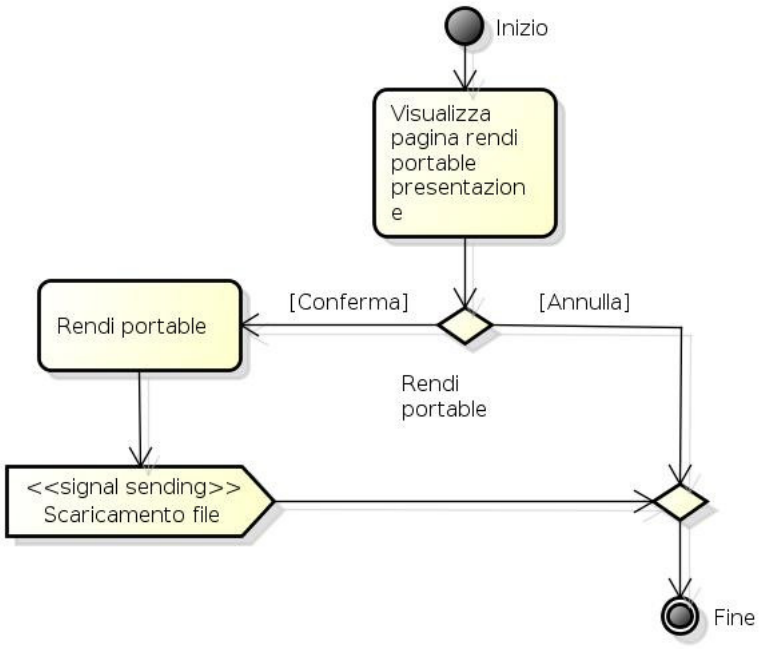

Figura 24: Rendi portable

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

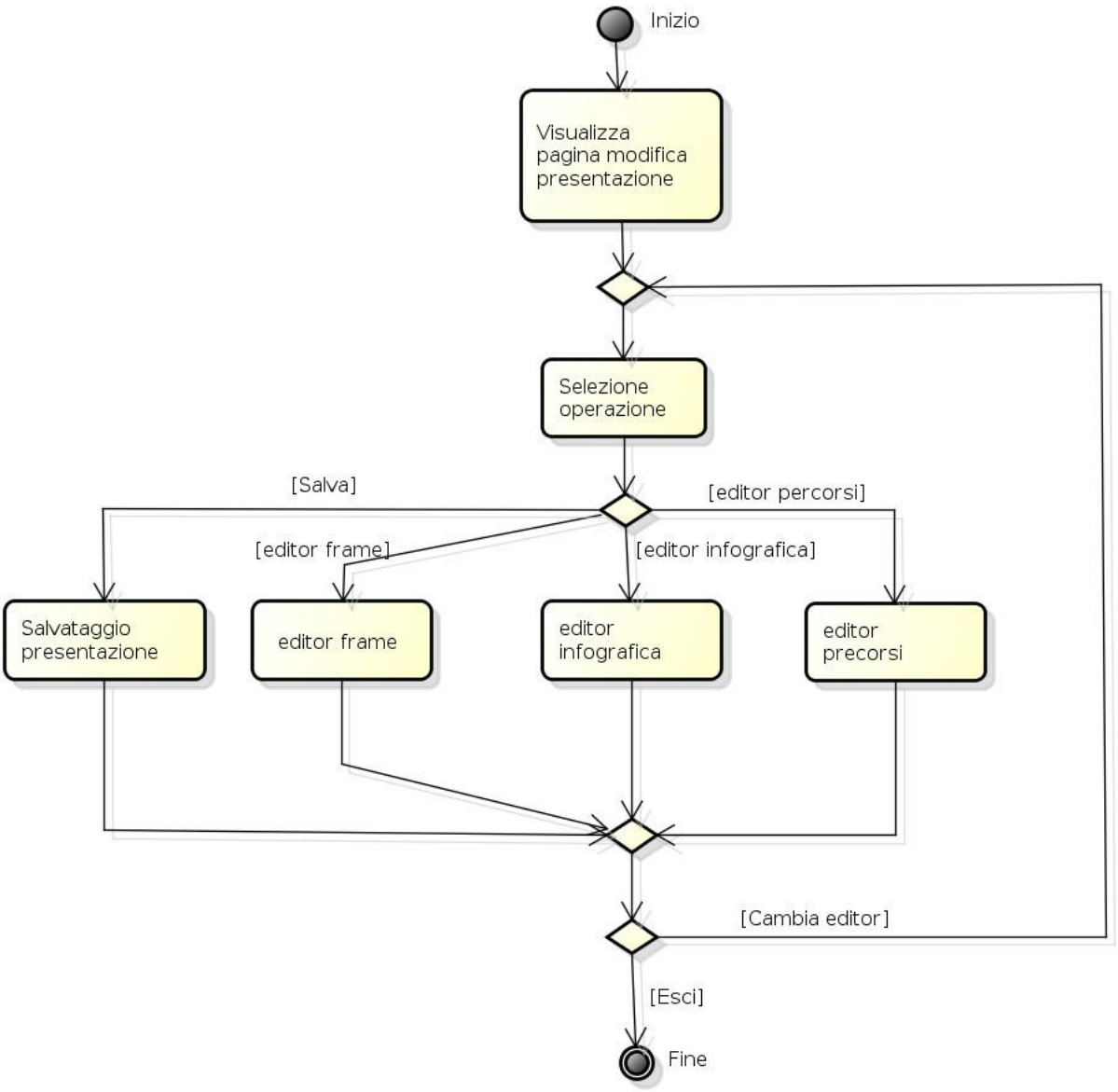

Figura 25: Modifica presentazione

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

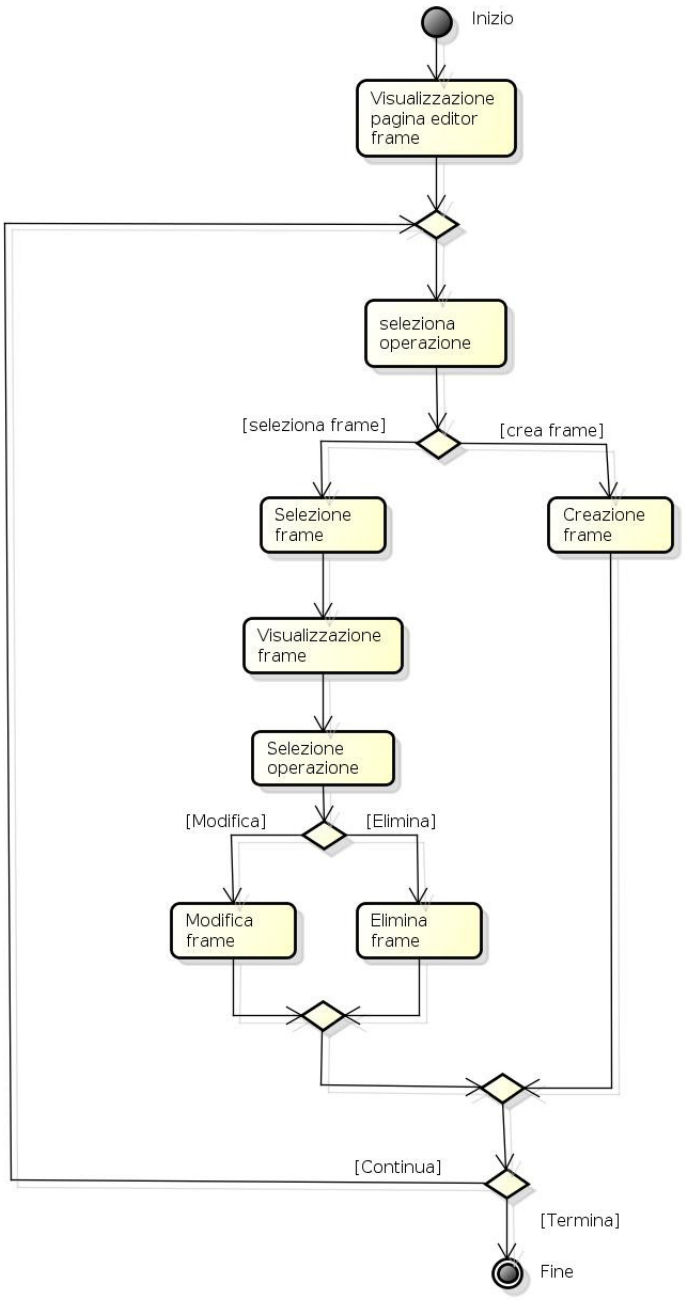

Figura 26: Frame editor

L'utente può procedere alla creazione di un novo frame oppure selezionarne uno esistente. Una volta selezionato un frame questo viene visualizzato nell'editor e a questo punto si può scegliere se eliminarlo o modificarlo.

6.17 Modifica frame

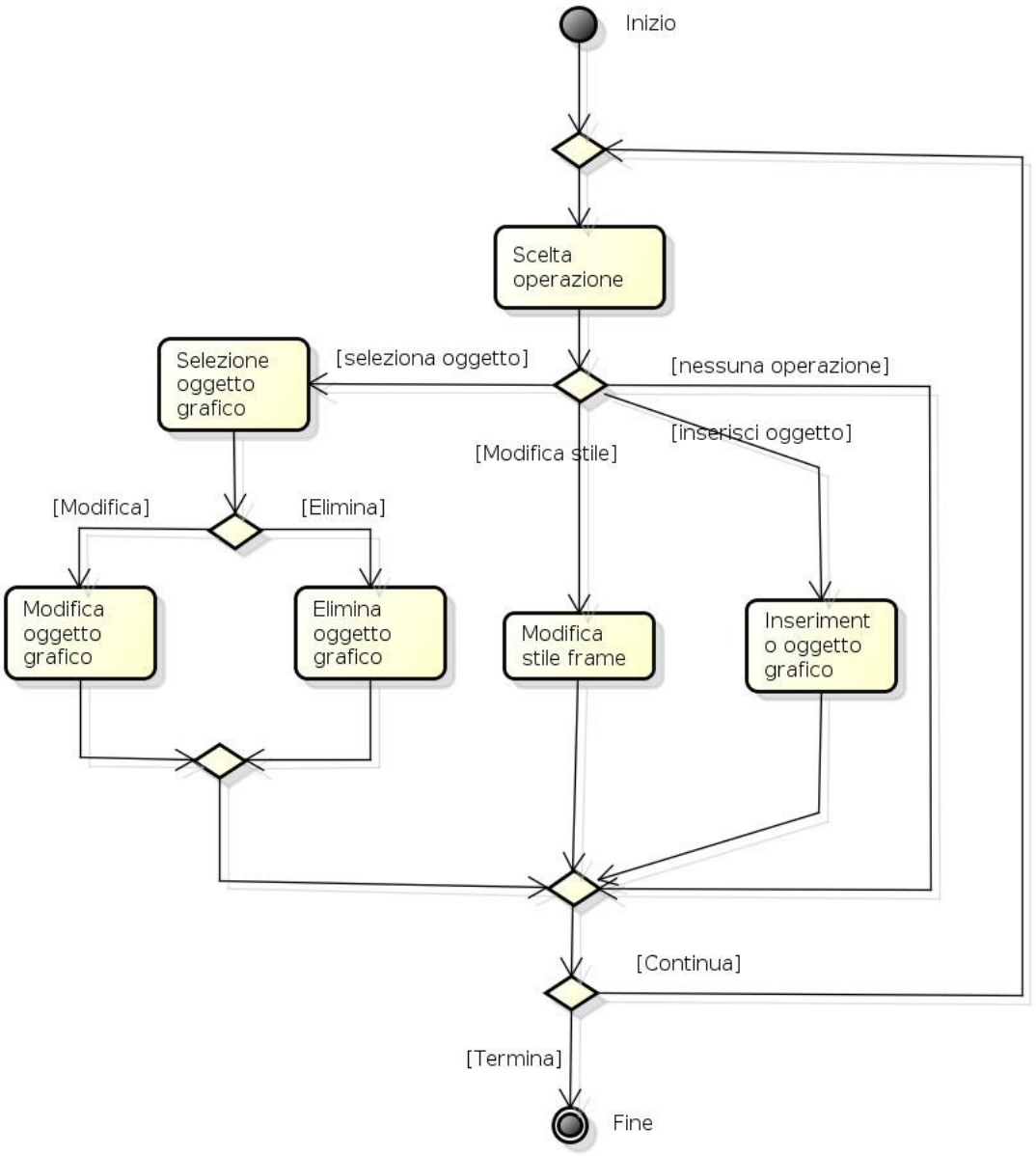

Figura 27: Modifica frame

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

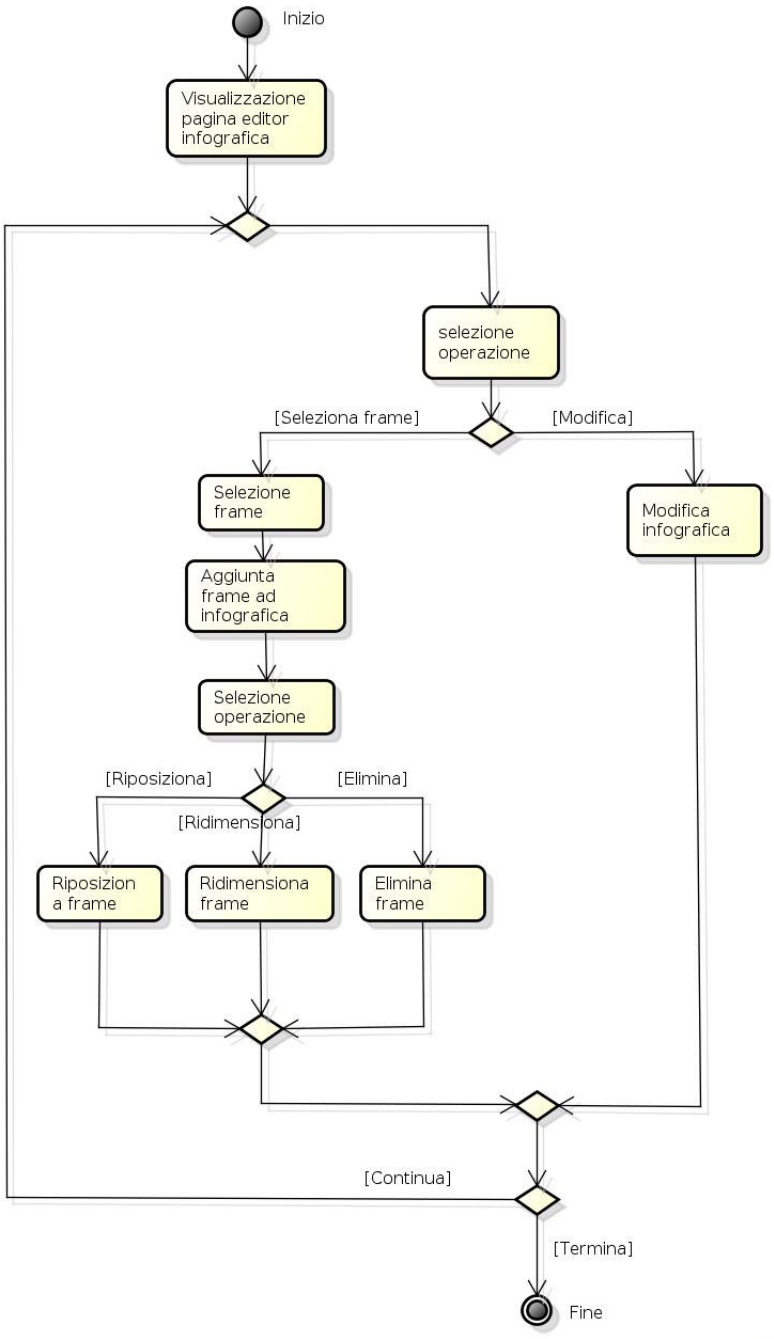

Figura 28: Infographic editor

L'utente può scegliere di:

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

6.19 Modifica infografica

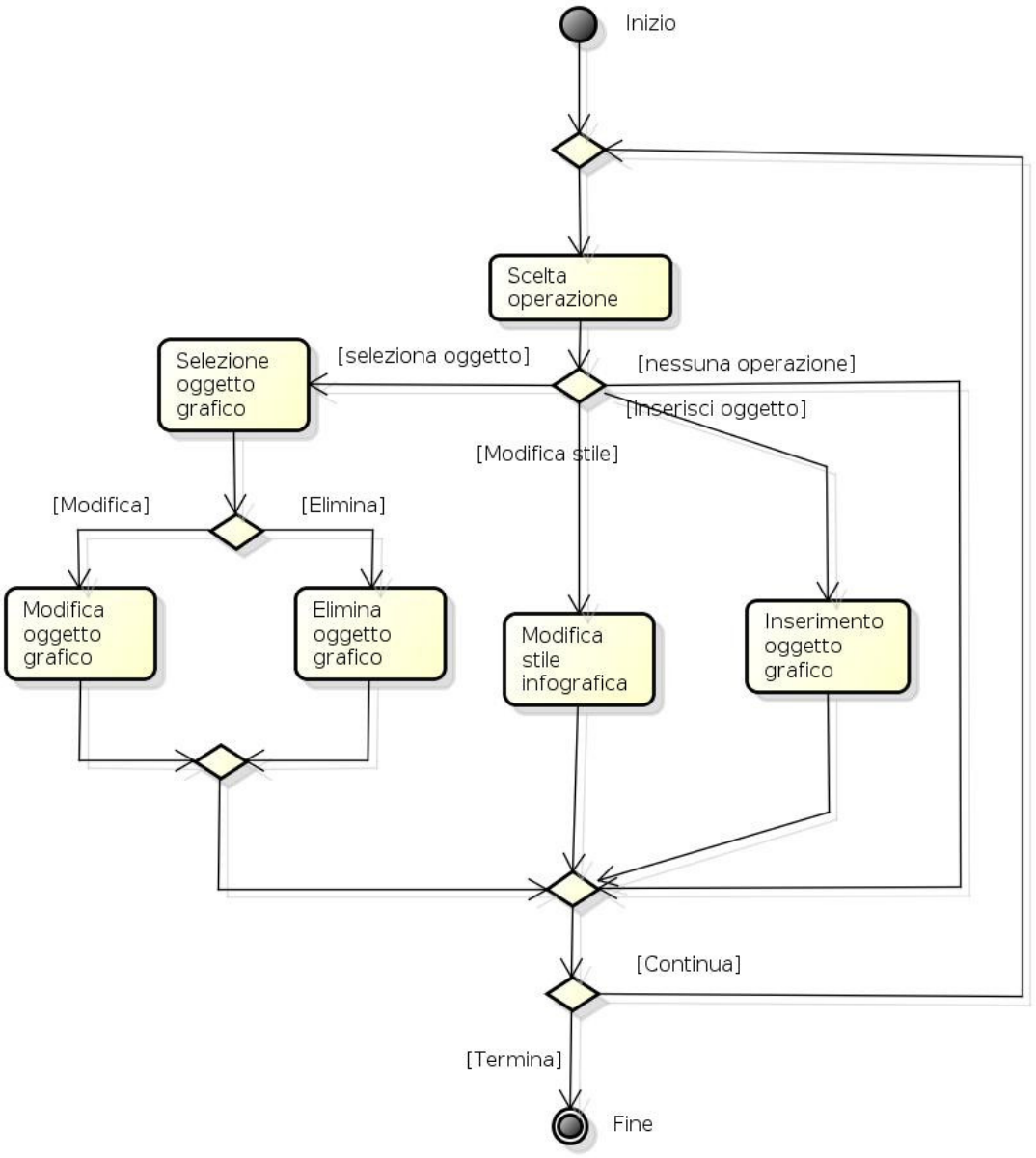

Figura 29: Modifica infografica

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

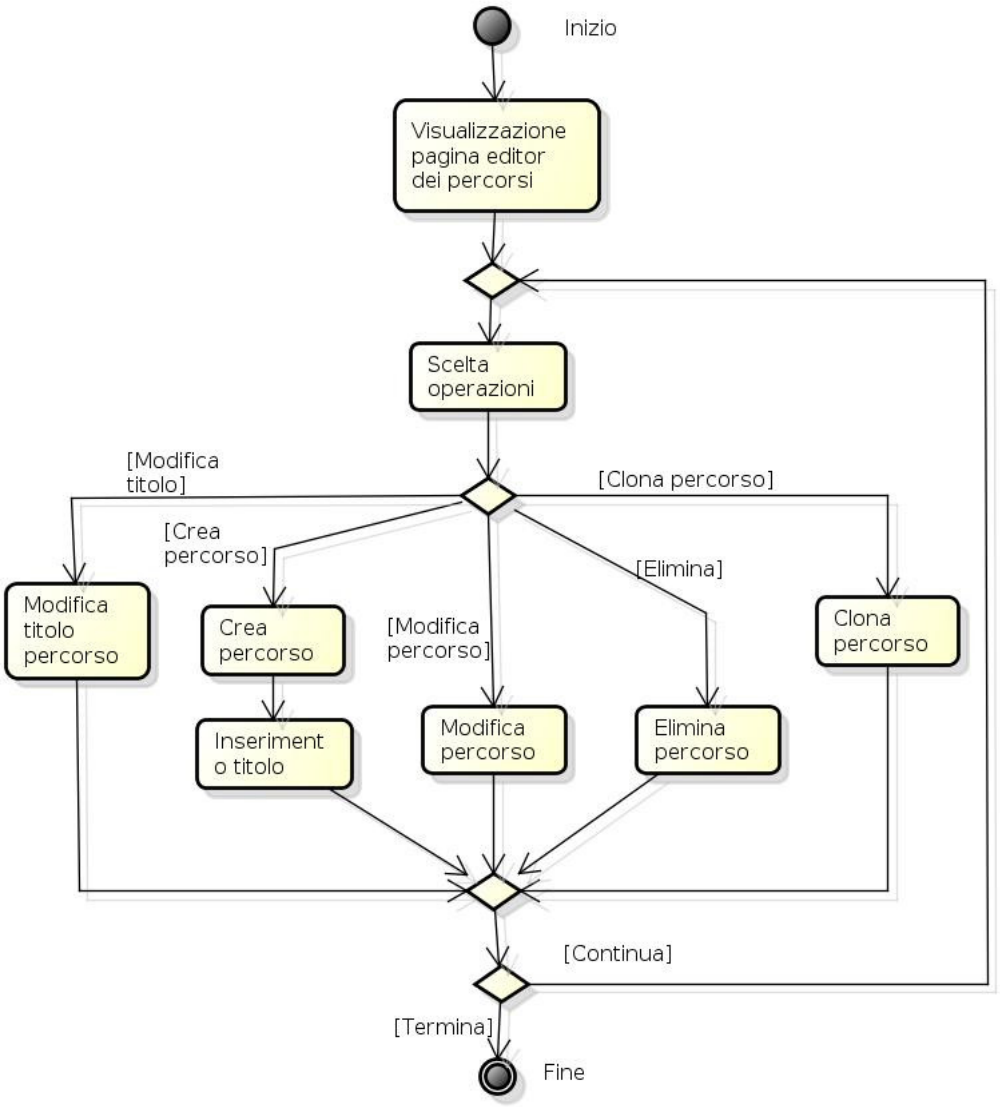

Figura 30: Editor percorsi

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

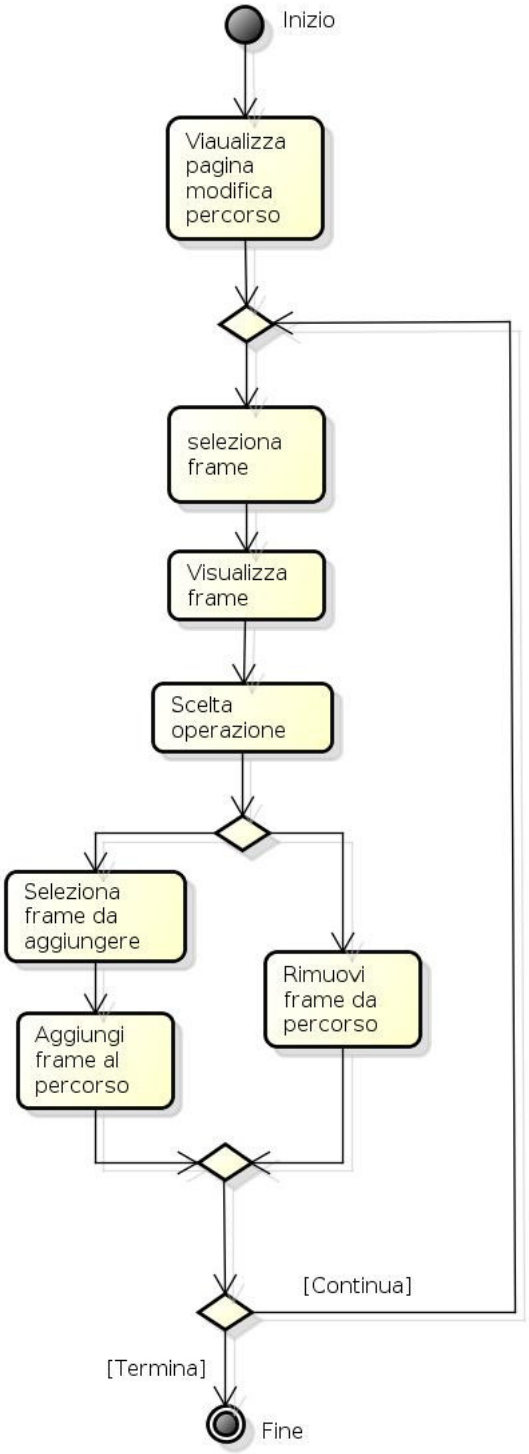

Figura 31: Modifica percorso

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

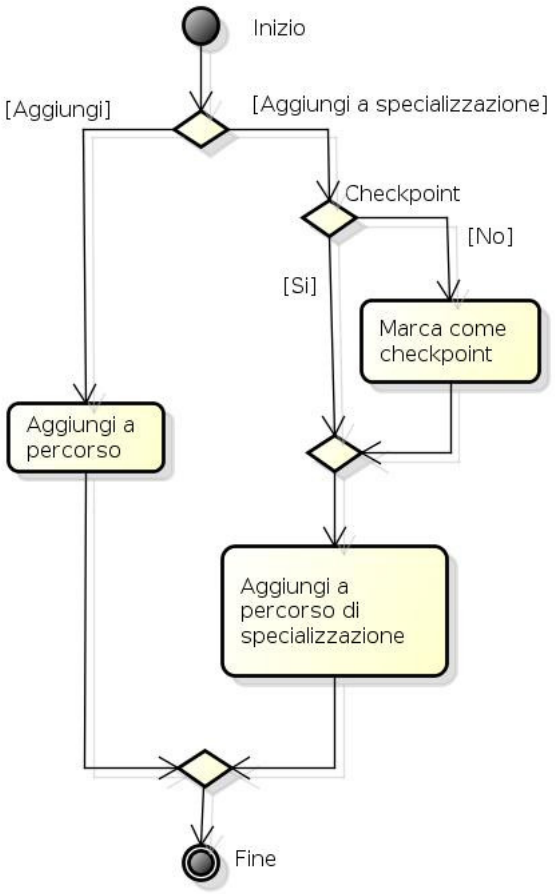

Figura 32: Aggiungi frame

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

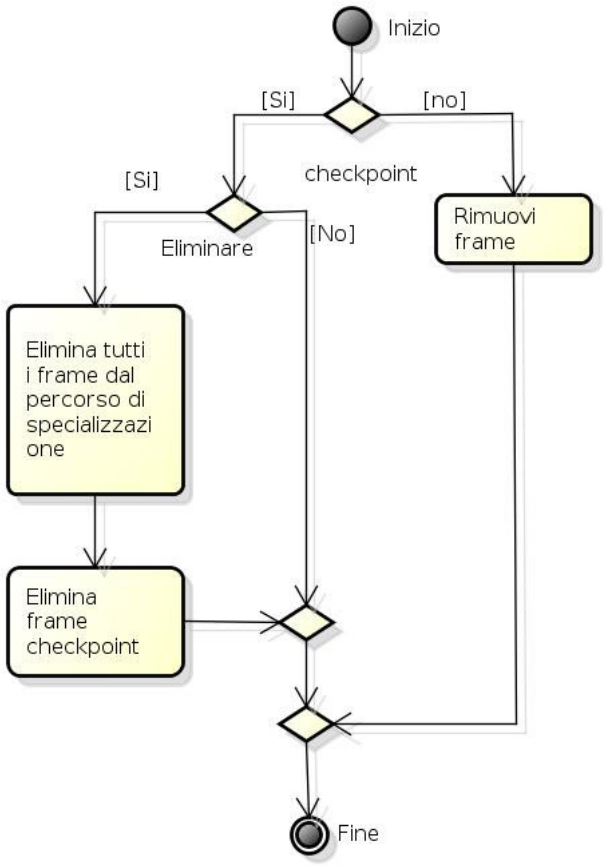

Figura 33: Rimuovi frame dal percorso

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

Dopo un'analisi delle caratteristiche del prodotto risulta chiaro che le tecnologie adottate siano adeguate per portare a termine il progetto. Le tecnologie e gli strumenti usati sono:

- I linguaggi HTML5$_G$ e CSS3$_G$ verranno utilizzati per creare l'interfaccia grafica. Questo garantisce una buona compatibilità con molti browser in particolare con Chrome$_G$;
- I framework AngularJS$_G$ e MeteorJS$_G$ basati sul linguaggio Javascript$_G$ ci permettono di interagire con le interfacce grafiche e con la gestione dei dati interni. MeteorJS$_G$ ci permette di interagire facilmente con la componente server$_G$, mentre AngularJS$_G$ ci permette di gestire la componente client$_G$;
- La libreria InteractJS$_G$, fornisce una serie di metodi che permettono di interagire facilmente con gli oggetti grafici presenti nell'interfaccia grafica;
- La libreria ImpressJS$_G$, fornisce una serie di metodi che permette di scorrere i frame quando si visualizza la presentazione all'utente;
- Per il salvataggio dei dati viene utilizzato il database MongoDB$_G$ che garantisce semplicità nella gestione degli stessi.

  Questi strumenti garantiscono di coprire la realizzazione di tutti i componenti del progetto.

8 Tracciamento requisiti-componenti

Requisito	Descrizione	Componenti
FOb1	l'utente deve poter creare una presentazione	Premi.-PresentationManager.- NewView Premi.-PresentationManager.- NewController
FOb1.1	l'utente deve poter scegliere un titolo per una presentazione	Premi.-PresentationManager.- NewView Premi.-PresentationManager.- NewController
FOb1.2	l'utente deve poter inserire una descrizione della presentazione	Premi.-PresentationManager.- NewView Premi.-PresentationManager.- NewController
FOp10	l'utente deve poter rendere live una presentazione pubblica	Premi.-PresentationManager.- View Premi.-PresentationManager.- Presentation-ManagerCtrl

FOp11	l'utente deve poter esportare una presentazione	Premi.-PresentationManager.- Export Premi.-PresentationManager.- Portable
FOp11.1	l'utente deve poter esportare la presentazione come poster	Premi.-PresentationManager.- Export Premi.-PresentationManager.- Portable
FOp11.2	l'utente deve poter esportare la presentazione in formato portabile	Premi.-PresentationManager.- Export
FOp12	l'utente deve poter partecipare ad una presentazione resa live	Premi.-Viewer.-View Premi.-Viewer.-ViewerCtrl

FOb13	l'utente deve poter registrarsi al sistema	Premi.-UserManager.-User Premi.-UserManager.-View Premi.-UserManager.-UserManagerCtrl Premi.-UserManager.-RegistrationView Premi.-UserManager.-Registration-Controller
FOb14	l'utente deve potersi autenticare	Premi.-UserManager.-User Premi.-UserManager.-View Premi.-UserManager.-UserManagerCtrl Premi.-UserManager.-LoginView Premi.-UserManager.-LoginController
FOb15	l'utente deve sapere quando ha commesso un errore	Premi.-View Premi.-PremiCtrl

FOb16	l'utente deve poter modificare la propria password	Premi.- UserManager.- User Premi.- UserManager.- View Premi.- UserManager.- UserManagerCtrl Premi.- UserManager.- ChangePasswordView Premi.- UserManager.- ChangePas- swordController
FOb2	l'utente deve poter selezionare una sua presentazione	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl Premi.- Presentation.- Presentation Premi.- PresentationManager.- ListPresView Premi.- PresentationManager.- ListPresController
FOb3	l'utente deve poter eseguire una sua presentazione	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl

FDe3.1	l'utente deve poter scegliere un percorso presentativo precedentemente creato	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl
FOb3.2	l'utente deve poter avanzare nel percorso presentativo scelto	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl
FOb3.3	l'utente deve poter retrocedere nel percorso presentativo scelto	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl
FDe3.4	l'utente deve poter seguire un percorso di approndimento	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl
FOb3.5	l'utente deve poter tornare ad un checkpoint	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl
FOb3.6	l'utente deve poter interrompere l'esecuzione della presentazione	Premi.- Viewer.- View Premi.- Viewer.- ViewerCtrl

FOb4	l'utente deve poter modificare una presentazione	Premi.- Editor.- View Premi.- Editor.- EditorCtrl
FOb4.1	l'utente deve poter inserire un oggetto grafico	Premi.- Editor.- View Premi.- Editor.- EditorCtrl
FOb4.1.1	l'utente deve poter inserire un'area di testo	Premi.- Presentation.- GraphicObject
FOb4.1.1.1	l'utente deve poter inserire del testo	Premi.- Presentation.- Text Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.1.1.2	l'utente deve poter scegliere il tipo di font per il testo	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.1.1.3	l'utente deve poter scegliere il colore del testo	Premi.- Editor.- TextView Premi.- Editor.- TextController

FOb4.1.1.4	l'utente deve poter scegliere la dimensione del testo	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.1.2	l'utente deve poter inserire un frame nella presentazione	Premi.- Presentation.- Text Premi.- Editor.- TextView Premi.- Editor.- TextController
FOp4.1.2.1	l'utente deve poter scegliere la forma del frame	Premi.- Presentation.- Frame
FOb4.1.3	l'utente deve poter inserire un immagine nella presentazione	Premi.- Presentation.- Frame
FOb4.1.3.1	l'utente deve poter scegliere un'immagine da filesystem	Premi.- Presentation.- Image Premi.- Editor.- ImageView Premi.- Editor.- ImageController
FOb4.1.4	l'utente deve poter inserire uno shape nella presentazione	Premi.- Presentation.- Image Premi.- Editor.- ImageView Premi.- Editor.- ImageController

FOb4.1.4.1	l'utente deve poter scegliere la forma di uno shape	Premi.- Presentation.- Shape Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController
FOb4.10	l'utente deve poter modificare la descrizione di una presentazione	Premi.- Presentation.- GraphicObject
FOb4.2	l'utente deve poter selezionare un oggetto grafico	Premi.- PresentationManager.- EditTitleDescrView Premi.- PresentationManager.- EditTitleDescrController
FOb4.3	l'utente deve poter modificare un oggetto grafico nella presentazione	Premi.- Presentation.- GraphicObject
FOb4.3.1	l'utente deve poter modificare un frame	Premi.- Presentation.- GraphicObject
FOb4.3.1.1	l'utente deve poter ridimensionare un frame della presentazione	Premi.- Presentation.- Frame Premi.- Editor.- FrameEditor.- View Premi.- Editor.- FrameEditor.- FrameEditorCtrl

FOb4.3.1.2	l'utente deve poter riposizionare un frame nella presentazione	Premi.- Editor.- InfographicEditor.- View Premi.- Editor.- InfographicEditor.- InfographicEditorCtrl
FOb4.3.1.3	l'utente deve poter modificare lo stile di un frame della presentazione	Premi.- Editor.- InfographicEditor.- View Premi.- Editor.- InfographicEditor.- InfographicEditorCtrl
FOb4.3.2	l'utente deve poter modificare il contenuto di un'area di testo della presentazione	Premi.- Presentation.- Frame Premi.- Editor.- FrameEditor.- View Premi.- Editor.- FrameEditor.- FrameEditorCtrl
FOb4.3.2.1	l'utente deve poter ridimensionare un area di testo della presentazione	Premi.- Presentation.- Text Premi.- Editor.- TextView Premi.- Editor.- TextController

FOb4.3.2.2	l'utente deve poter riposizionare un area di testo nella presentazione	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.3.2.3	l'utente deve poter modificare lo stile dell'area di testo della presentazione	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.3.2.4	l'utente deve poter modificare il contenuto di un'area di testo della presentazione	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.3.2.5	l'utente deve poter cambiare il livello di un'area di testo	Premi.- Editor.- TextView Premi.- Editor.- TextController
FOb4.3.3	l'utente deve poter modificare una shape	Premi.- Presentation.- Text Premi.- Editor.- TextView Premi.- Editor.- TextController

FOb4.3.3.1	l'utente deve poter riposizionare una shape	Premi.- Presentation.- Shape Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController
FOb4.3.3.2	l'utente deve poter cambiare lo stile di una shape	Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController
FOb4.3.3.3	l'utente deve poter ridimensionare una shape	Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController
FOb4.3.3.4	l'utente deve poter cambiare livello ad una shape	Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController
FOb4.3.4	l'utente deve poter modificare un'immagine della presentazione	Premi.- Presentation.- Shape Premi.- Editor.- ShapeView Premi.- Editor.- ShapeController

FOb4.3.4.1	l'utente deve poter riposizionare un'immagine nella presentazione	Premi.- Presentation.- Image Premi.- Editor.- ImageView Premi.- Editor.- ImageController
FOb4.3.4.2	l'utente deve poter ridimensionare un'immagine della presentazione	Premi.- Editor.- ImageView Premi.- Editor.- ImageController
FOb4.3.4.3	l'utente deve poter cambiare il livello di un immagine	Premi.- Editor.- ImageView Premi.- Editor.- ImageController
FOb4.4	l'utente deve poter eliminare un oggetto grafico dalla presentazione	Premi.- Presentation.- GraphicObject
FDe4.5	l'utente deve poter creare un percorso per la presentazione	Premi.- Editor.- TextView
FDe4.5.1	l'utente deve poter inserire un titolo per un percorso	Premi.- Presentation.- Trail Premi.- Editor.- TrailsEditor.- NewView Premi.- Editor.- TrailsEditor.- NewController

FDe4.5.2	l'utente deve poter clonare un percorso esistente della presentazione	Premi.- Editor.- TrailsEditor.- NewView Premi.- Editor.- TrailsEditor.- NewController
FDe4.6	l'utente deve poter selezionare un percorso della presentazione	Premi.- Presentation.- Trail Premi.- Editor.- TrailsEditor.- NewView Premi.- Editor.- TrailsEditor.- NewController
FOb4.7	l'utente deve poter modificare un percorso della presentazione	Premi.- Presentation.- Trail Premi.- Editor.- TrailsEditor.- View Premi.- Editor.- TrailsEditor.- TrailsEditorCtrl
FDe4.7.1	l'utente deve poter modificare il titolo del percorso	Premi.- Presentation.- Trail Premi.- Editor.- TrailsEditor.- View Premi.- Editor.- TrailsEditor.- TrailsEditorCtrl

FOb4.7.2	l'utente deve poter aggiungere un passo ad un cammino della presentazione	Premi.- Editor.- TrailsEditor.- EditTitleView Premi.- Editor.- TrailsEditor.- EditTitleController
FOb4.7.3	l'utente deve poter modificare l'ordine dei frame nel percorso	Premi.- Editor.- TrailsEditor.- FrameListController Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController
FDe4.7.4	l'utente deve poter rendere checkpoint un frame	Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController
FDe4.7.5	l'utente deve poter rimuovere la marcatura a checkpoint di un frame	Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController

FDe4.7.5.1	l'utente deve confermare l'eliminazione di una marcatura a checkpoint	Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController
FDe4.7.5.2	l'utente deve poter annullare la rimozione di una marcatura a checkpoint	Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController
FOb4.7.6	l'utente deve poter selezionare un frame del percorso	Premi.- Editor.- TrailsEditor.- EditTrailView Premi.- Editor.- TrailsEditor.- EditTrailController
FOb4.8	l'utente deve poter modificare il titolo di una presentazione	Premi.- Presentation.- Trail Premi.- Editor.- TrailsEditor.- View Premi.- Editor.- TrailsEditor.- TrailsEditorCtrl

FDe4.9	l'utente deve poter eliminare un percorso	Premi.-PresentationManager.- EditTitleDescrView Premi.-PresentationManager.- EditTitleDescrController
FOb5	l'utente deve poter salvare una presentazione	Premi.-Utility
FOb6	l'utente deve poter eliminare una presentazione	Premi.-PresentationManager.- RemoveView Premi.-PresentationManager.- RemoveController
FOb6.1	l'utente deve poter scegliere di annullare l'operazione di eliminazione di una presentazione	Premi.-PresentationManager.- RemoveView Premi.-PresentationManager.- RemoveController
FOp7	l'utente deve poter rendere pubblica una presentazione	Premi.-PresentationManager.- PublicView Premi.-PresentationManager.- PublicController

FOp8	il sistema deve poter generare un link per una presentazione live	Premi.-PresentationManager.- View Premi.-PresentationManager.- Presentation-ManagerCtrl
FOp9	l'utente deve poter rendere privata una presentazione publica	Premi.-PresentationManager.- PublicView Premi.-PresentationManager.- PublicController

Tabella 2: Tracciamento requisiti-componeni

9 Tracciamento componenti-requisiti

Componente	Requisiti
Premi	
Premi.Utility	FOb5
Premi.View	FOb15
Premi.PremiCtrl	FOb15
Premi.UserManager	
Premi.UserManager.User	FOb13 FOb14 FOb16
Premi.UserManager.View	FOb13 FOb14 FOb16
Premi.UserManager.UserManagerCtrl	FOb13 FOb14 FOb16
Premi.UserManager.LoginView	FOb14
Premi.UserManager.LoginController	FOb14
Premi.UserManager.RegistrationView	FOb13
Premi.UserManager.RegistrationController	FOb13
Premi.UserManager.ChangePasswordView	FOb16
Premi.UserManager.ChangePasswordController	FOb16
Premi.Viewer	
Premi.Viewer.View	FOp12 FOb2 FOb3 FDe3.1 FOb3.2 FOb3.3 FDe3.4 FOb3.5 FOb3.6

Premi.Viewer.ViewerCtrl	FOp12 FOb2 FOb3 FDe3.1 FOb3.2 FOb3.3 FDe3.4 FOb3.5 FOb3.6
Premi.Presentation	
Premi.Presentation.GraphicObject	FOb4.1 FOb4.2 FOb4.3
Premi.Presentation.GoContent	
Premi.Presentation.Text	FOb4.1.1 FOb4.3.2
Premi.Presentation.Image	FOb4.1.3 FOb4.3.4
Premi.Presentation.Shape	FOb4.1.4 FOb4.3.3
Premi.Presentation.Frame	FOb4.1.2 FOp4.1.2.1 FOb4.3.1
Premi.Presentation.Trail	FDe4.5 FDe4.6 FOb4.7
Premi.Presentation.Presentation	FOb2
Premi.PresentationManager	
Premi.PresentationManager.Export	FOp11 FOp11.1
Premi.PresentationManager.Portable	FOp11 FOp11.2
Premi.PresentationManager.View	FOp10 FOp8

Premi.PresentationManager.PresentationManagerCtrl	FOp10 FOp8
Premi.PresentationManager.RemoveView	FOb6 FOb6.1
Premi.PresentationManager.RemoveController	FOb6 FOb6.1
Premi.PresentationManager.PublicView	FOp7 FOp9
Premi.PresentationManager.PublicController	FOp7 FOp9
Premi.PresentationManager.EditTitleDescrView	FOb4.10 FOb4.8
Premi.PresentationManager.EditTitleDescrController	FOb4.10 FOb4.8
Premi.PresentationManager.NewView	FOb1 FOb1.1 FOb1.2
Premi.PresentationManager.NewController	FOb1 FOb1.1 FOb1.2
Premi.PresentationManager.ListPresView	FOb2
Premi.PresentationManager.ListPresController	FOb2
Premi.Editor	
Premi.Editor.View	FOb4
Premi.Editor.EditorCtrl	FOb4

Premi.Editor.TextView	FOb4.1.1 FOb4.1.1.1 FOb4.1.1.2 FOb4.1.1.3 FOb4.1.1.4 FOb4.3.2 FOb4.3.2.1 FOb4.3.2.2 FOb4.3.2.3 FOb4.3.2.4 FOb4.3.2.5 FOb4.4
Premi.Editor.TextController	FOb4.1.1 FOb4.1.1.1 FOb4.1.1.2 FOb4.1.1.3 FOb4.1.1.4 FOb4.3.2 FOb4.3.2.1 FOb4.3.2.2 FOb4.3.2.3 FOb4.3.2.4 FOb4.3.2.5
Premi.Editor.ShapeView	FOb4.1.4 FOb4.1.4.1 FOb4.3.3 FOb4.3.3.1 FOb4.3.3.2 FOb4.3.3.3 FOb4.3.3.4
Premi.Editor.ShapeController	FOb4.1.4 FOb4.1.4.1 FOb4.3.3 FOb4.3.3.1 FOb4.3.3.2 FOb4.3.3.3 FOb4.3.3.4

Premi.Editor.ImageView	FOb4.1.3 FOb4.1.3.1 FOb4.3.4 FOb4.3.4.1 FOb4.3.4.2 FOb4.3.4.3
Premi.Editor.ImageController	FOb4.1.3 FOb4.1.3.1 FOb4.3.4 FOb4.3.4.1 FOb4.3.4.2 FOb4.3.4.3
Premi.Editor.StyleView	FOb4.3.1.3
Premi.Editor.StyleController	FOb4.3.1.3
Premi.Editor.FrameListView	
Premi.Editor.FrameEditor	
Premi.Editor.FrameEditor.View	FOb4.3.1
Premi.Editor.FrameEditor.FrameEditorCtrl	FOb4.3.1
Premi.Editor.InfographicEditor	
Premi.Editor.InfographicEditor.View	FOb4.3.1.1 FOb4.3.1.2
Premi.Editor.InfographicEditor.InfographicEditorCtrl	FOb4.3.1.1 FOb4.3.1.2
Premi.Editor.InfographicEditor.FrameListController	
Premi.Editor.TrailsEditor	
Premi.Editor.TrailsEditor.View	FDe4.6 FOb4.7
Premi.Editor.TrailsEditor.TrailsEditorCtrl	FDe4.6 FOb4.7
Premi.Editor.TrailsEditor.FrameListController	FOb4.7.2

Premi.Editor.TrailsEditor.EditTrailView	FOb4.7.2 FOb4.7.3 FDe4.7.4 FDe4.7.5 FDe4.7.5.1 FDe4.7.5.2
Premi.Editor.TrailsEditor.EditTrailController	FOb4.7.2 FOb4.7.3 FDe4.7.4 FDe4.7.5 FDe4.7.5.1 FDe4.7.5.2
Premi.Editor.TrailsEditor.NewView	FDe4.5 FDe4.5.1 FDe4.5.2
Premi.Editor.TrailsEditor.NewController	FDe4.5 FDe4.5.1 FDe4.5.2
Premi.Editor.TrailsEditor.EditTitleView	FDe4.7.1
Premi.Editor.TrailsEditor.EditTitleController	FDe4.7.1
Premi.Editor.TrailsEditor.RemoveView	FDe4.9
Premi.Editor.TrailsEditor.RemoveController	FDe4.9

Tabella 3: Tracciamento componenti-requisiti

10 Design Pattern

10.1 Design Pattern Architetturali

10.1.1 MVC - Model View Controller

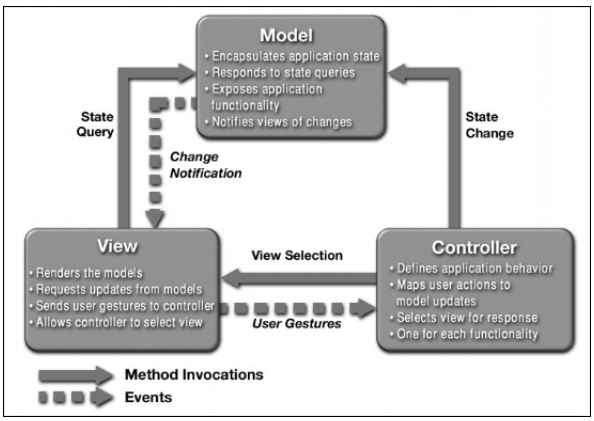

Figura 34: Diagramma del design pattern MVC

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

10.1.2 MVVM - Model View ViewModel

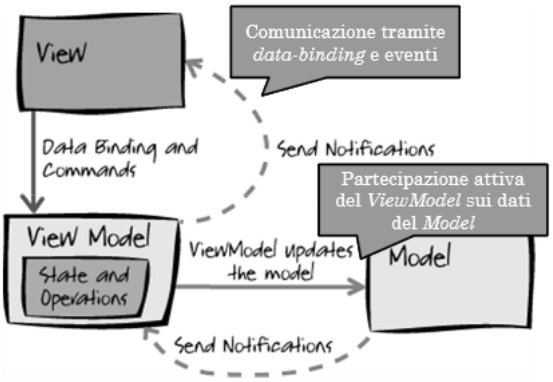

Figura 35: Diagramma del design pattern MVVM

- **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 qui 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 i suoi comportamenti. E' compito del ViewModel, 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;

10.1.3 Dependency Injection

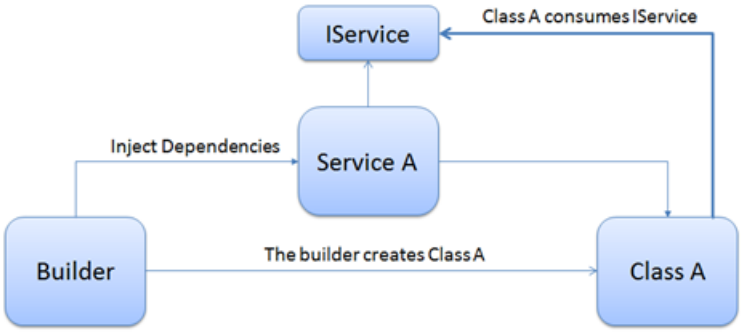

Figura 36: Diagramma del design pattern Dependency Injection

- **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* della 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.

10.2 Design Pattern Creazionali

10.2.1 Factory Method

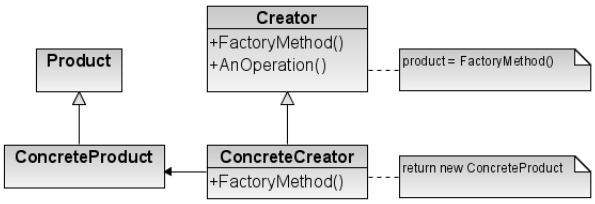

Figura 37: Diagramma del design pattern Factory Method

- **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.