UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Dipartimento di Ingegneria e di Scienze dell'Informazione e Matematica

Object Oriented Software Design

BIBLIOTECA DIGITALE

Gruppo di lavoro:

- Amicosante Andrea Matr.(246790) andrea.amicosante@student.univaq.it
- D'Alfonso Angelo Matr.(249069) angelo.dalfonso@student.univaq.it
- Ravanetti Stefano Matr.(248035) stefano.ravanetti@student.univaq.it

A.A. 2017/2018

INDICE

1. REQUIREMENTS

Il sistema offre una biblioteca digitale di testi e studi che contribuiscono alla formazione della cultura all'interno dell'Università degli Studi dell'Aquila. Lo scopo del progetto è di consentire la consultazione di manoscritti che devono essere digitalizzati. Gli utenti possono consultare il catalogo digitale dei manoscritti e visualizzare un'opera di loro interesse a seguito di una ricerca. Gli utenti con privilegi possono scaricare le opere. Ogni utente può richiedere di diventare trascrittore, che può aggiungere il testo in formato digitale di una immagine.

1.1 DOCUMENTO DEI REQUISITI:

Viewer

- Consultazione opere
  Gli Utenti possono visualizzare le opere presenti nel catalogo.
- Ricerca nel catalogo
  Gli Utenti possono cercare le opere nel catalogo per: titolo, autore, contenuto del testo.
- Download dell'opera
  Utenti con Privilegi possono scaricare le opere.
- Richiesta per essere collaboratore del sistema (Trascrittore)
  Un Utente può far richiesta per diventare Trascrittore.
- Accesso dati personali
  Gli Utenti possono visualizzare il proprio profilo contente i propri dati.

Uploader

- Upload opera nel sistema (immagini)
  Gli Uploader caricano immagini delle pagine scansionate e vengono inserite nel catalogo dopo l'approvazione di un revisore upload
- Caricamento metadati opera
  Gli Uploader definiscono i dati della/e pagina/e caricata/e.

Transcriber

- Trasformazione opera in un testo digitale (formato TEI)
  I trascrittori trascrivono le opere in formato cartaceo sul catalogo attraverso un editor di testo (TEI) presente nel programma.

Manager

- Gestione assegnazioni
  I revisori delle trascrizioni assegnano delle parti di opera a uno o più trascrittori.
- Riassegnazione pagine
  Le pagine assegnate ai Trascrittori possono essere riassegnate.
- Revisione delle trascrizioni
  I Revisori delle trascrizioni controllano le trascrizioni e a seguito le validano se adatte alla pubblicazione.
- Gestione livelli di esperienza

I Revisori delle trascrizioni gestiscono i livelli di esperienza dei trascrittori. Il livello va da 1 a 5 e viene scelto in base alla sua esperienza.

- Revisiona acquisizione immagini
  I Revisori degli Upload validano le immagini caricate prima di aggiungerle al catalogo.

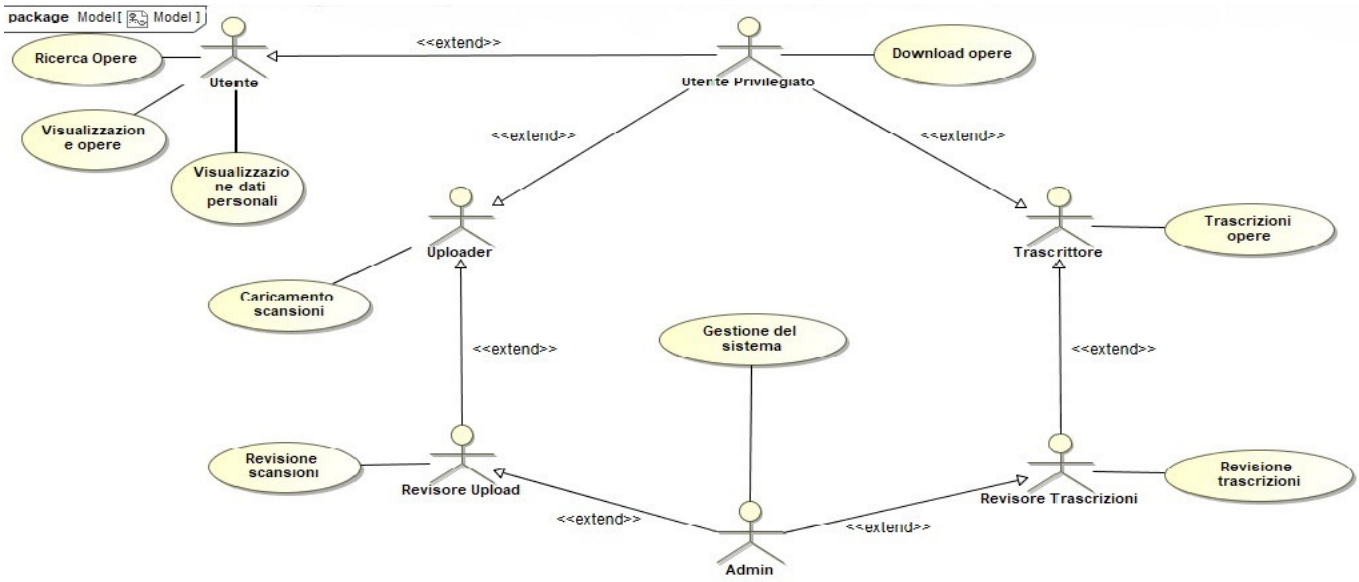
Administrator

- Gestione del sistema
  L'Amministratore gestisce tutto il sistema.

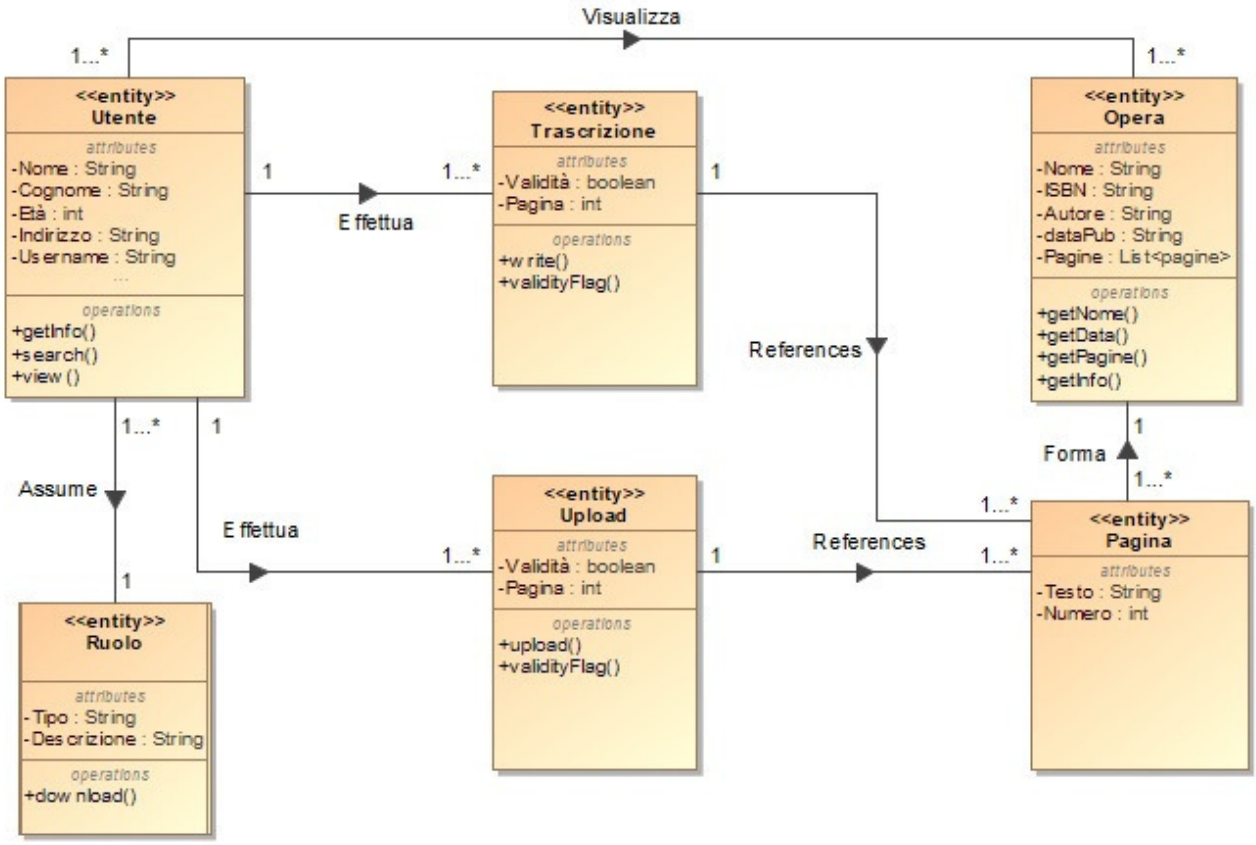
1.2 MODELLI UML USE CASE

Sono stati identificati i seguenti attori del sistema:

- **Amministratore**
  Gestione del sistema
- **Utente base**
  Consultazione opere (visualizzazione e ricerca)
- **Utente privilegiato**
  Utente base con possibilità di download dell'opera
- **Trascrittore**
  Utente che può trascrivere opere nell'editor (TEI)
- **Revisori Trascrizioni**
  Revisione e validazione della trascrizione
- **Uploader**
  Caricamento immagini
- **Revisori Upload**
  Revisione immagini caricate

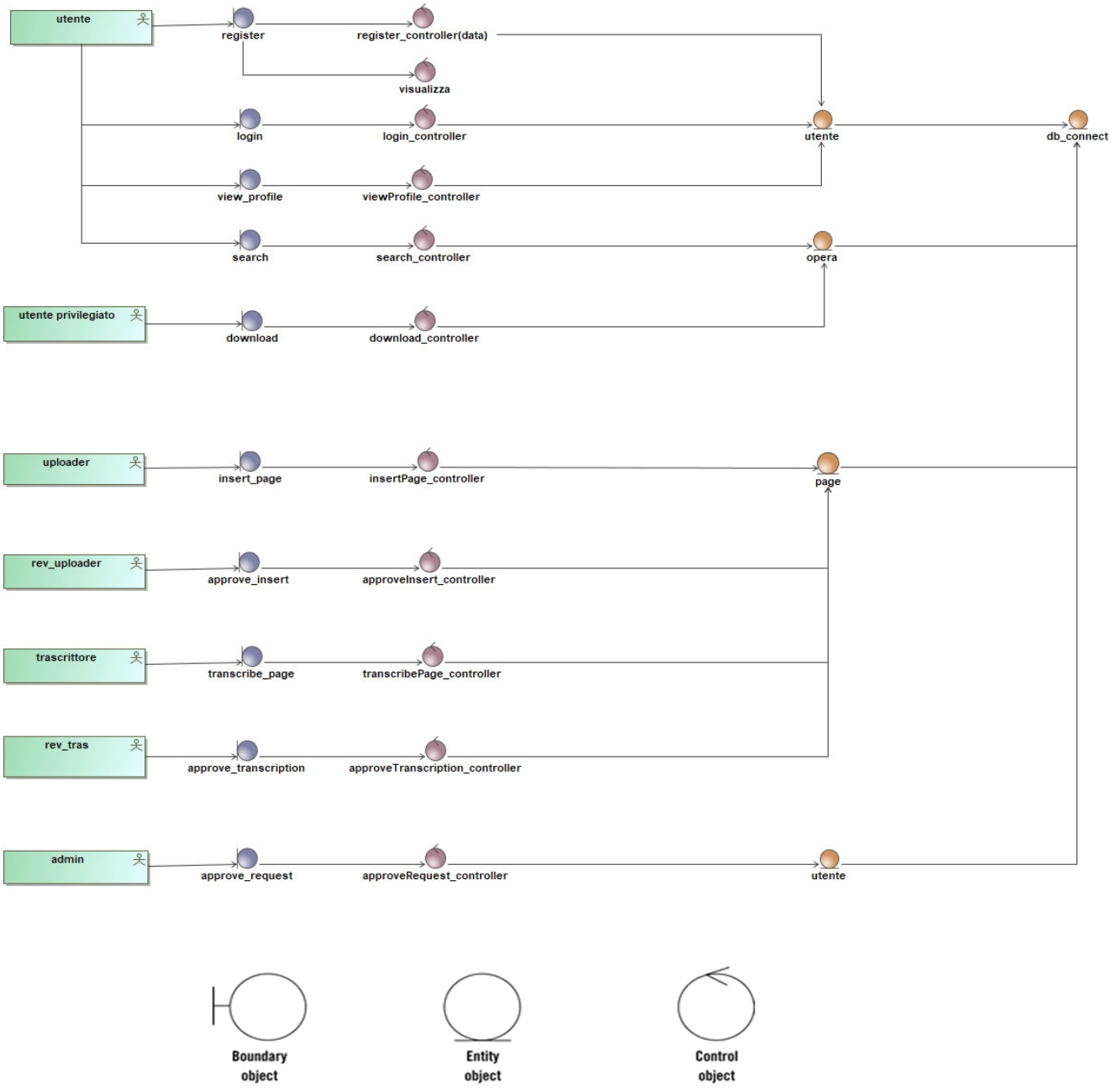


1.3 MODELLI DI DOMINIO



Nel nostro modello di dominio abbiamo descritto le varie entità, che fanno parte e hanno rilevanza nel nostro sistema e le loro relazioni, mettendo a fuoco i concetti fondamentali.

1.4 ANALISI FINALIZZATA ALL'INDIVIDUAZIONE DELLE CLASSI ENTITY, BOUNDARY E CONTROLLER



- Le **entity** sono oggetti che rappresentano i dati di sistema
- I **boundary** sono oggetti che si interfacciano con gli attori di sistema
- I **controller** sono oggetti che mediano tra i boundary e le entity. Questi servono da collante tra boundary ed entity, implementando la logica necessaria per gestire i vari elementi e le loro interazioni

2. SYSTEM DESIGN

2.1 MODELLO DELL'ARCHITETTURA SOFTWARE

MVC:

Il modello architetturale scelto è stato il pattern **MVC**, il quale utilizza un design pattern che suddivide il sistema in più moduli indipendenti e renderà più facili le modifiche e gli aggiornamenti.

Abbiamo usato il pattern **MVC** perché:

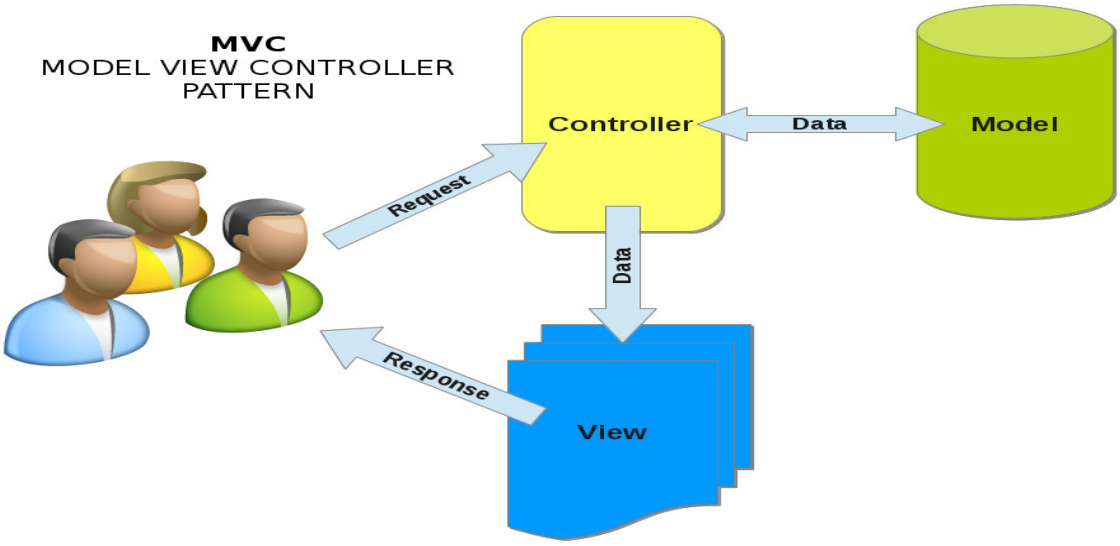
- È più facile riutilizzare il codice, quindi lo sviluppo è più veloce
- Il codice è più organizzato, quindi è più facile da capire e mantenere
- È più facile testare il codice.
- è più sicuro

Il pattern architetturale è diviso in tre parti:

1. **Model**
   - Incapsula lo stato dell'applicazione
   - Deve permettere di accedere ai dati
   - Deve notificare i cambiamenti di stato

2. **View**
   - Mostra il modello
   - Gestisce l'interazione con l'utente

3. **Controller**
   - Rappresenta la logica applicativa
   - Collega le azioni dell'utente con modifiche dello stato
   - Sceglie cosa essere mostrato



2.2 DESCRIZIONE DELL'ARCHITETTURA

Il sistema è stato suddiviso in tre componenti distaccate seguendo il pattern architetturale MVC. Le componenti e il loro utilizzo sono state descritte qui di seguito:

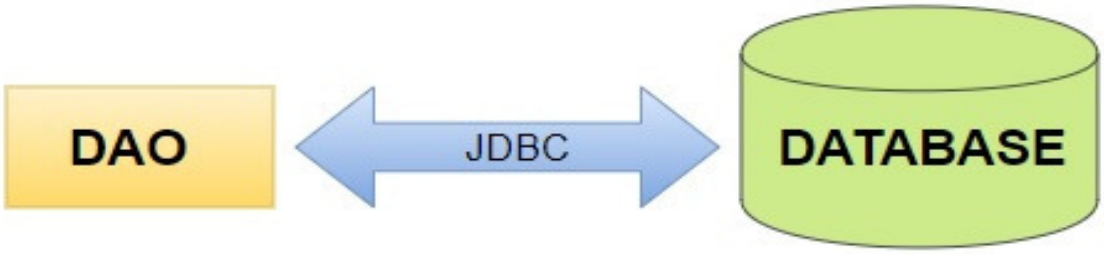
View

Il package view è stato suddiviso in due sottopackage, FXML e FrontController. **FXML** contiene dei file con omonima estensione (file XML generati con interfaccia grafica) creati da SceneBuilder che abbiamo utilizzato per gestire l'interfaccia grafica. **FrontController** contiene dei Controller per l'esclusiva gestione delle view di FXML, comunicano con il package Controller per l'utilizzo delle funzioni del sistema, garantendo così una perfetta modularità tra i package dell'MVC.

Model

Nel package model sono inoltre stati utilizzati altri due design pattern per la gestione dei dati:

Il **DAO** (*Data Access Object*) servirà per collegare un Database al sistema che gestisce tutti i dati degli utenti e delle opere nel sistema. Il DAO incapsula tutti gli accessi ai dati conservandoli in un posto unico (DB). Per gestire al meglio i dati da inserire nel DB è stato utilizzato il **VO** (*Value Object*) che incapsula i business data (*il modello business coopera con il database con una relazione associativa*) per migliorare la loro gestione limitando il passaggio di oggetti pesanti attraverso il sistema. I VO sono rappresentazioni "leggere" delle classi di dominio dell'analisi del modello.



Controller

Questo componente ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model, ma il Controller non rappresenta un semplice "ponte" tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione e gestisce tutte le operazioni effettuate da GUI (Graphic user interface), attraverso varie funzioni corrispondenti a ogni parte dell'interfaccia.

2.3 DESCRIZIONE DELLE SCELTE E STRATEGIE ADOTTATE

Come **design patterns** abbiamo usato:

- ***Abstract Factory*** (RAGGIO DI AZIONE: OBJECT, considera relazioni tra oggetti che sono modificate a run-time e sono più dinamiche), che rientra nei Pattern creazionali, riguardanti i processi di creazioni di oggetti delegando parte del processo di creazione di un oggetto ad altri oggetti. Questo pattern ci ha permesso di avere diverse modalità di presentazione e comportamento per gli elementi.

- ***Façade*** (*RAGGIO DI AZIONE OBJECT, considera relazioni tra oggetti che sono modificate a run-time e sono più dinamiche*) rientra nei Pattern strutturali e focalizzano attenzione su composizione di classi e oggetti, utilizzano l'ereditarietà descrivendo modi per raggruppare oggetti. Questo pattern ci ha permesso si suddividere un sistema in sottosistemi aiutandoci a ridurre la complessità e in più ha minimizzato le comunicazioni e le dipendenze tra i sottosistemi.

3.0 SOFTWARE/OBJECT DESIGN

3.1 UML CLASS DIAGRAMS

Un **Class Diagram** descrive:

- Classi
- Relazioni tra classi:
  - associazione (*uso*);
  - generalizzazione (*ereditarietà*);
  - aggregazione (*contenimento*).

Definisce la visione statica del sistema, ed è il modello principe di UML, perché definisce gli elementi base del sistema da sviluppare.

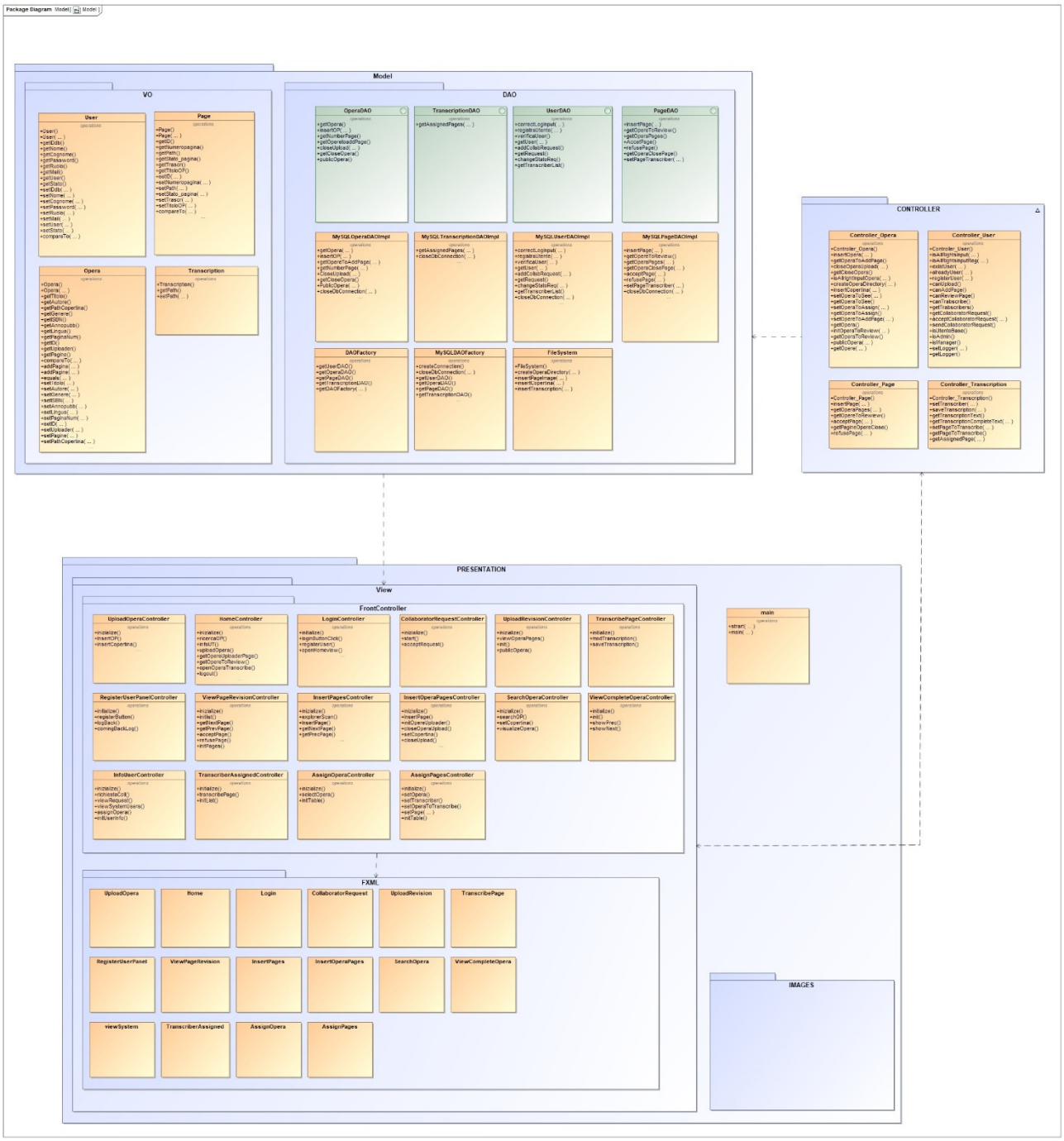
Un class diagram può essere definito a livello:

1. concettuale (o di Analisi):

- studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione

2. di specifica implementativa:

- studia il software da un punto di vista implementativo, specificando come va sviluppato il sistema;
- è un raffinamento del precedente.



Per una migliore comprensione dell'immagine sono state allegate tutte quelle originali nel GitHub.