



# Specifica Tecnica

## Informazioni sul documento

<b>Nome documento</b>	Specifica Tecnica
<b>Versione</b>	v1.0.0
<b>Data redazione</b>	2014-01-10
<b>Redattori</b>	<ul style="list-style-type: none"><li>• Adami Alberto</li><li>• Feltre Beatrice</li><li>• Luisetto Luca</li><li>• Magnabosco Nicola</li><li>• Scapin Davide</li></ul>
<b>Verificatori</b>	<ul style="list-style-type: none"><li>• Bissacco Nicolò</li><li>• Martignago Jimmy</li></ul>
<b>Approvazione</b>	<ul style="list-style-type: none"><li>• Martignago Jimmy</li></ul>
<b>Lista distribuzione</b>	<ul style="list-style-type: none"><li>• <i>Seven Monkeys</i></li><li>• <i>Prof. Tullio Vardanega</i></li><li>• <i>Prof. Riccardo Cardin</i></li></ul>
<b>Uso</b>	Esterno

## Sommario

Questo documento contiene la specifica tecnica di Romeo. Viene descritta l'architettura generale del sistema con una prima bozza dei componenti del sistema.

## Diario delle Modifiche

Modifica	Autore & Ruolo	Data	Versione
<i>Approvazione del documento</i>	Martignago Jimmy <i>Responsabile di Progetto</i>	2014-02-04	v1.0.0
<i>Verifica del documento prima dell'approvazione</i>	Bissacco Nicolò <i>Verificatore</i>	2014-02-04	v0.3.0
<i>Apportate modifiche a seguito di verifica per i capitoli 6-8 e appendici A e B</i>	Feltre Beatrice <i>Progettista</i>	2014-02-03	v0.2.1
<i>Verifica del documento nei capitoli 6-8 e appendici A e B</i>	Martignago Jimmy <i>Verificatore</i>	2014-02-01	v0.2.0
<i>Stesura Appendice A: Descrizione Design Pattern</i>	Luisetto Luca <i>Progettista</i>	2014-02-01	v0.1.5
<i>Stesura capitolo stime di fattibilità e risorse necessarie</i>	Magnabosco Nicola <i>Responsabile di Progetto</i>	2014-02-01	v0.1.4
<i>Apportate modifiche a seguito di verifica per i capitoli 1-5</i>	Adami Alberto <i>Progettista</i>	2014-01-31	v0.1.3
<i>Stesura e importazione tracciamento requisiti-componenti, componenti-requisiti</i>	Scapin Davide <i>Progettista</i>	2014-01-30	v0.1.2
<i>Stesura capitolo design pattern utilizzati</i>	Scapin Davide <i>Progettista</i>	2014-01-27	v0.1.1
<i>Verifica del documento nei capitoli 1-5</i>	Bissacco Nicolò <i>Verificatore</i>	2014-01-26	v0.1.0
<i>Incremento capitolo componenti e classi, Romeo::Controller</i>	Magnabosco Nicola <i>Progettista</i>	2014-01-25	v0.0.9
<i>Stesura capitolo del database utilizzato da Romeo</i>	Feltre Beatrice <i>Progettista</i>	2014-01-24	v0.0.8
<i>Incremento capitolo componenti e classi, Romeo::View</i>	Luisetto Luca <i>Progettista</i>	2014-01-23	v0.0.7
<i>Inizio Stesura capitolo componenti e classi, Romeo::Model</i>	Luisetto Luca <i>Progettista</i>	2014-01-20	v0.0.6
<i>Stesura capitolo descrizione dell'architettura</i>	Adami Alberto <i>Progettista</i>	2014-01-18	v0.0.5
<i>Stesura Appendice B: Prototipo di UI</i>	Magnabosco Nicola <i>Progettista</i>	2014-01-16	v0.0.4
<i>Stesura capitolo Tecnologie utilizzate</i>	Adami Alberto <i>Progettista</i>	2014-01-15	v0.0.3
<i>Stesura Introduzione e Diagrammi delle Attività</i>	Feltre Beatrice <i>Progettista</i>	2014-01-13	v0.0.2
<i>Creata struttura del documento</i>	Luisetto Luca <i>Progettista</i>	2014-01-10	v0.0.1

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Scopo del documento	1
1.2	Scopo del prodotto	1
1.3	Glossario	1
1.4	Riferimenti	1
1.4.1	Normativi	1
1.4.2	Informativi	1
<b>2</b>	<b>Tecnologie utilizzate</b>	<b>2</b>
2.1	C++	2
2.2	Qt	2
2.2.1	Signals & Slots	2
2.2.2	<i>The Meta-Object System</i>	3
2.2.3	Licenza	3
2.3	ITK	4
2.4	VTK	4
2.5	SQLite	5
<b>3</b>	<b>Descrizione architettura</b>	<b>6</b>
3.1	Premesse	6
3.2	Architettura generale	7
<b>4</b>	<b>Componenti e classi</b>	<b>8</b>
4.1	Romeo	8
4.1.1	Informazioni sul package	8
4.1.2	Descrizione	8
4.1.3	Package contenuti	8
4.1.4	Relazioni d'uso tra i componenti	8
4.2	Romeo::Model	9
4.2.1	Informazioni sul package	9
4.2.2	Descrizione	9
4.2.3	Package contenuti	9
4.2.4	Relazioni d'uso tra i componenti	9
4.2.5	Classi contenute	9
4.3	Romeo::Model::Core	10
4.3.1	Informazioni sul package	10
4.3.2	Descrizione	10
4.3.3	Package contenuti	10
4.3.4	Relazioni d'uso tra i componenti	10
4.3.5	Classi contenute	10
4.4	Romeo::Model::Core::Adapters	12
4.4.1	Informazioni sul package	12
4.4.2	Descrizione	12
4.4.3	Package contenuti	12
4.5	Romeo::Model::Core::Adapters::Features	12
4.5.1	Informazioni sul package	12
4.5.2	Descrizione	13
4.5.3	Relazioni d'uso tra i componenti	13
4.5.4	Classi contenute	13

4.6	Romeo::Model::Core::Adapters::Algorithms . . . . .	14
4.6.1	Informazioni sul package . . . . .	14
4.6.2	Descrizione . . . . .	15
4.6.3	Relazioni d'uso tra i componenti . . . . .	15
4.6.4	Classi contenute . . . . .	15
4.7	Romeo::Model::Util . . . . .	15
4.7.1	Informazioni sul package . . . . .	15
4.7.2	Descrizione . . . . .	16
4.7.3	Package contenuti . . . . .	16
4.8	Romeo::Model::Util::DAO . . . . .	16
4.8.1	Informazioni sul package . . . . .	16
4.8.2	Descrizione . . . . .	16
4.8.3	Classi contenute . . . . .	16
4.9	Romeo::Model::Util::ExporterModel . . . . .	18
4.9.1	Informazioni sul package . . . . .	18
4.9.2	Descrizione . . . . .	18
4.9.3	Classi contenute . . . . .	18
4.10	Romeo::Model::Util::Log . . . . .	19
4.10.1	Informazioni sul package . . . . .	19
4.10.2	Descrizione . . . . .	19
4.10.3	Classi contenute . . . . .	19
4.11	Romeo::Model::Help . . . . .	19
4.11.1	Informazioni sul package . . . . .	19
4.11.2	Descrizione . . . . .	20
4.11.3	Classi contenute . . . . .	20
4.12	Romeo::View . . . . .	21
4.12.1	Informazioni sul package . . . . .	21
4.12.2	Descrizione . . . . .	21
4.12.3	Package contenuti . . . . .	21
4.12.4	Relazioni tra i componenti . . . . .	21
4.13	Romeo::View::Window . . . . .	22
4.13.1	Informazioni sul package . . . . .	22
4.13.2	Descrizione . . . . .	22
4.13.3	Relazioni tra i componenti . . . . .	22
4.13.4	Classi contenute . . . . .	22
4.14	Romeo::View::Dialog . . . . .	25
4.14.1	Informazioni sul Package . . . . .	25
4.14.2	Descrizione . . . . .	25
4.14.3	Classi contenute . . . . .	25
4.15	Romeo::View::Component . . . . .	26
4.15.1	Informazioni sul package . . . . .	26
4.15.2	Descrizione . . . . .	26
4.15.3	Classi contenute . . . . .	26
4.16	Romeo::Controller . . . . .	27
4.16.1	Informazioni sul package . . . . .	27
4.16.2	Descrizione . . . . .	27
4.16.3	Relazioni tra i componenti . . . . .	27
4.16.4	Classi contenute . . . . .	27
<b>5</b>	<b>Database Romeo . . . . .</b>	<b>31</b>

5.1	Descrizione testuale delle classi . . . . .	32
5.1.1	Subject . . . . .	32
5.1.2	Group . . . . .	32
5.1.3	Dataset . . . . .	32
5.1.4	Protocol . . . . .	32
5.1.5	ClusterAlgorithm . . . . .	33
5.1.6	Feature . . . . .	33
5.1.7	Analysis . . . . .	33
5.1.8	Result . . . . .	33
5.2	Descrizione delle associazioni . . . . .	34
5.3	Progettazione logica . . . . .	34
<b>6</b>	<b>Diagrammi delle attività . . . . .</b>	<b>36</b>
6.1	Attività principali . . . . .	36
6.2	Creazione nuovo Subject . . . . .	38
6.2.1	Caricare un File . . . . .	39
6.3	Creare nuovo gruppo di Subjects . . . . .	40
6.4	Creare nuovo Protocol . . . . .	41
6.5	Creare nuovo Dataset . . . . .	42
6.6	Visualizzare i Subject . . . . .	43
6.7	Gestire i gruppi di Subject . . . . .	44
6.8	Gestire i Protocol . . . . .	45
6.9	Gestire i Dataset . . . . .	46
6.10	Avviare un'analisi . . . . .	47
6.10.1	Visualizzare i risultati dell'analisi . . . . .	50
6.11	Visualizzare le analisi effettuate . . . . .	51
6.12	Aprire la guida . . . . .	52
<b>7</b>	<b>Design pattern . . . . .</b>	<b>53</b>
7.1	Design pattern architetturali . . . . .	53
7.1.1	MVC . . . . .	53
7.2	Design pattern creazionali . . . . .	53
7.2.1	Factory . . . . .	53
7.2.2	Singleton . . . . .	54
7.3	Design pattern strutturali . . . . .	55
7.3.1	Adapter . . . . .	55
7.3.2	DAO (Data Access Object) . . . . .	56
7.3.3	Facade . . . . .	57
7.4	Design pattern comportamentali . . . . .	58
7.4.1	Strategy . . . . .	58
<b>8</b>	<b>Stime di fattibilità e risorse necessarie . . . . .</b>	<b>59</b>
<b>9</b>	<b>Tracciamento . . . . .</b>	<b>60</b>
9.1	Tracciamento componenti-requisiti . . . . .	60
9.2	Tracciamento requisiti-componenti . . . . .	64
<b>A</b>	<b>Descrizione dei design pattern . . . . .</b>	<b>72</b>
A.1	Design pattern architetturali . . . . .	72
A.1.1	MVC . . . . .	72
A.2	Design pattern creazionali . . . . .	74

A.2.1	Factory . . . . .	74
A.2.2	Singleton . . . . .	75
A.3	Design pattern strutturali . . . . .	76
A.3.1	Adapter . . . . .	76
A.3.2	DAO (Data Access Object) . . . . .	77
A.3.3	Facade . . . . .	78
A.4	Design pattern comportamentali . . . . .	79
A.4.1	Strategy . . . . .	79
<b>B</b>	<b>Prototipo di UI . . . . .</b>	<b>80</b>
B.1	Welcome Page . . . . .	80
B.2	Pagine di creazione . . . . .	81
B.2.1	Creazione di un nuovo Subject . . . . .	81
B.2.2	Creazione di un gruppo di Subject . . . . .	81
B.2.3	Creazione di un Protocol . . . . .	82
B.2.4	Creazione di un Dataset . . . . .	82
B.3	Pagine di visualizzazione e modifica . . . . .	83
B.3.1	Visualizzazione dei Subject . . . . .	83
B.3.2	Visualizzazione dei gruppi di Subject . . . . .	83
B.3.3	Visualizzazione dei Protocol . . . . .	84
B.3.4	Visualizzazione dei Dataset . . . . .	84
B.4	Analisi e visualizzazione risultati . . . . .	85
B.4.1	Avvio analisi . . . . .	85
B.4.2	Esecuzione analisi . . . . .	85
B.4.3	Visualizzazione risultati . . . . .	86
B.4.4	Visualizzazione dettaglio risultati . . . . .	86

## Elenco delle figure

1	Schema interazione oggetti tramite <i>Signals &amp; Slots</i> . . . . .	3
2	Vista package dell'architettura di Romeo . . . . .	7
3	Componente Romeo . . . . .	8
4	Componente Romeo::Model . . . . .	9
5	Componente Romeo::Model::Core . . . . .	10
6	Componente Romeo::Model::Core::Adapters . . . . .	12
7	Componente Romeo::Model::Core::Adapters::Features . . . . .	12
8	Componente Romeo::Model::Core::Adapters::Algorithms . . . . .	14
9	Componente Romeo::Model::Util . . . . .	15
10	Componente Romeo::Model::Util::DAO . . . . .	16
11	Componente Romeo::Model::Util::ExporterModel . . . . .	18
12	Componente Romeo::Model::Util::Log . . . . .	19
13	Componente Romeo::Model::Help . . . . .	19
14	Componente Romeo::View . . . . .	21
15	Componente Romeo::View::Window . . . . .	22
16	Componente Romeo::View::Dialog . . . . .	25
17	Componente Romeo::View::Window . . . . .	26
18	Componente Romeo::Controller . . . . .	27
19	Struttura del database di Romeo . . . . .	31
20	Diagramma Attività - Attività principali dell'applicativo Romeo . . .	37
21	Diagramma Attività - Creazione nuovo Subject . . . . .	38
22	Diagramma Attività - Caricamento di un file . . . . .	39
23	Diagramma Attività - Creazione nuovo gruppo di Subject . . . . .	40
24	Diagramma Attività - Creazione di un nuovo Protocol . . . . .	41
25	Diagramma Attività - Creazione di un nuovo Dataset . . . . .	42
26	Diagramma Attività - Visualizzazione dei Subject . . . . .	43
27	Diagramma Attività - Gestione dei gruppi di Subject . . . . .	44
28	Diagramma Attività - Gestione dei Protocol . . . . .	45
29	Diagramma Attività - Gestione dei Dataset . . . . .	46
30	Diagramma Attività - Avvio di un'analisi . . . . .	47
31	Diagramma Attività - Esecuzione analisi per ogni Protocol . . . . .	48
32	Diagramma Attività - Esecuzione analisi per ogni Protocol . . . . .	49
33	Diagramma Attività - Visualizzazione dei risultati dell'analisi effettuata	50
34	Diagramma Attività - Visualizzazione di tutte le analisi effettuate . .	51
35	Diagramma Attività - Apertura della guida . . . . .	52
36	Utilizzo di Factory in Romeo . . . . .	53
37	Utilizzo di Singleton in Romeo . . . . .	54
38	Utilizzo di Adapter in Romeo . . . . .	55
39	Utilizzo di DAO in Romeo . . . . .	56
40	Utilizzo di Facade in Romeo . . . . .	57
41	Utilizzo di Strategy in Romeo . . . . .	58
42	Diagramma del design pattern MVC . . . . .	72
43	Diagramma del design pattern Factory . . . . .	74
44	Diagramma del design pattern Singleton . . . . .	75
45	Diagramma del design pattern Adapter . . . . .	76
46	Diagramma del design pattern DAO . . . . .	77
47	Diagramma del design pattern Facade . . . . .	78
48	Diagramma del design pattern Strategy . . . . .	79

49	Romeo: Mock-up della pagina di benvenuto . . . . .	80
50	Mock-up della pagina di creazione di un nuovo Subject . . . . .	81
51	Mock-up della pagina di creazione di un gruppo di Subject . . . . .	81
52	Mock-up della pagina di creazione di un Protocol . . . . .	82
53	Mock-up della pagina di creazione di un Dataset . . . . .	82
54	Mock-up della pagina di visualizzazione dei Subject . . . . .	83
55	Mock-up della pagina di visualizzazione dei gruppi di Subject . . . . .	83
56	Mock-up della pagina di visualizzazione dei gruppi dei Protocol . . . . .	84
57	Mock-up della pagina di visualizzazione dei Dataset . . . . .	84
58	Mock-up della pagina di avvio analisi . . . . .	85
59	Mock-up della finestra di analisi . . . . .	85
60	Mock-up della finestra dei risultati . . . . .	86
61	Mock-up della finestra di dettaglio dei risultati per Protocol . . . . .	86
62	Mock-up della finestra di dettaglio dei risultati per Subject . . . . .	87



# 1 Introduzione

## 1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello dell'applicativo Romeo. Verrà presentata l'architettura generale secondo la quale saranno organizzate le componenti software e saranno descritti i design pattern<sub>G</sub> utilizzati.

## 1.2 Scopo del prodotto

Il prodotto che si intende realizzare, denominato Romeo, si propone di fornire un sistema software per applicare la cluster analysis<sub>G</sub> ad immagini biomediche. Lo scopo principale è quello di offrire alla comunità scientifica internazionale uno strumento semplice, ma allo stesso tempo completo e flessibile per applicare gli algoritmi della cluster analysis<sub>G</sub>.

## 1.3 Glossario

Al fine di evitare ogni ambiguità e per permettere al lettore una migliore comprensione dei termini e acronimi utilizzati nei vari documenti formali, essi sono riportati nel *Glossario v1.0.0* che contiene una descrizione approfondita di tali termini e acronimi.

Ogni volta che compare un termine presente nel *Glossario*, esso è marcato con una “G” in pedice.

## 1.4 Riferimenti

### 1.4.1 Normativi

- **Analisi dei Requisiti:** *Analisi dei Requisiti v1.0.0*
- **Norme di Progetto:** *Norme di Progetto v1.0.0*

### 1.4.2 Informativi

- **Documentazione Qt<sub>G</sub>:** <http://qt-project.org/doc/qt-5/classes.html>;
- **Documentazione ITK<sub>G</sub>:** <http://www.itk.org/Doxygen45/html/index.html>;
- **Documentazione VTK<sub>G</sub>:** <http://www.vtk.org/doc/release/5.10/html/>;
- **Documentazione SQLite:** <http://www.sqlite.org/docs.html>;
- **Informativo Factory:** <http://www.oodeesign.com/factory-pattern.html>;
- **Design Pattern:** Elementi per il riuso di software a oggetti - E. Gamma, R. Helm, R. Johnson, J. Vlissides - 1<sup>a</sup> Edizione (2002).

## 2 Tecnologie utilizzate

In questa sezione verranno presentate le tecnologie su cui si basa lo sviluppo del progetto. In particolare, si presterà attenzione alle motivazioni per cui si è deciso di adottarle.

### 2.1 C++

Si è deciso di sviluppare Romeo con il linguaggio di programmazione C++. Questa scelta è stata dettata prevalentemente dai seguenti motivi:

- **Librerie consigliate:** le librerie a cui appoggiarsi per semplificare alcune attività, indicate dai proponenti, sono scritte in questo linguaggio. Il campo di applicazione di tali librerie verrà esposto in seguito;
- **Conoscenza del linguaggio:** la totalità dei componenti del gruppo, ha già avuto una buona dose di esperienza con tale linguaggio. Questo aspetto, sommato al precedente, è stato considerato sufficiente per giustificare questa scelta.

### 2.2 Qt

Si è deciso di utilizzare il framework<sub>G</sub> Qt<sub>G</sub> per lo sviluppo delle classi dell'architettura. Questa scelta è stata fatta perchè Qt<sub>G</sub> offre i seguenti vantaggi:

- **Framework C++:** Qt<sub>G</sub> è un framework<sub>G</sub> basato principalmente sul linguaggio C++<sub>G</sub>;
- **Forte componente grafica:** tale framework<sub>G</sub> è notoriamente orientato ad una particolare cura verso l'aspetto front-end delle applicazioni. Data l'importanza della GUI<sub>G</sub> nel progetto Romeo, Qt<sub>G</sub> è stato considerato adeguato per lo sviluppo.
- **Qt Designer:** applicativo che permette di disegnare interfacce grafiche senza agire direttamente sul codice sorgente. È possibile creare e personalizzare le finestre in modalità what-you-see-is-what-you-get. Inoltre, tutti gli oggetti creati (Widgets, forms, ecc...) si integrano con il codice sorgente usando il meccanismo di *Signals & Slots*, per cui diventa facile assegnare i comportamenti agli elementi grafici;
- **Qt Creator:** IDE<sub>G</sub> multiplatforma per lo sviluppo di applicazioni Qt<sub>G</sub>. Esso integra il controllo di versione appoggiandosi anche a Git<sub>G</sub>;
- **Esperienza del gruppo:** tutti i componenti hanno già avuto contatto con il framework<sub>G</sub>. Questa scelta mira anche a minimizzare il tempo di studio individuale delle tecnologie.

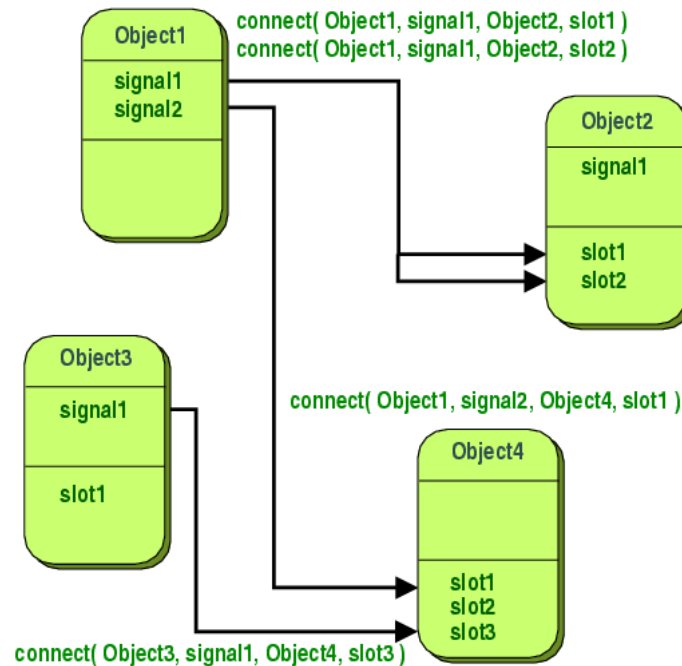
#### 2.2.1 Signals & Slots

Il meccanismo di *Signals & Slots*, è una caratteristica fondamentale del framework<sub>G</sub> Qt<sub>G</sub> che si occupa di far comunicare tra di loro gli oggetti.

Questo aspetto è determinante nel momento in cui si vuole sviluppare una GUI<sub>G</sub>, dato che ad un'azione che l'utente compie sull'interfaccia grafica, molto probabilmente ne consegue un'operazione nella parte logica dell'applicativo e viceversa. Più generalmente, si vogliono far comunicare due o più oggetti di qualsivoglia tipo. Tutte le classi derivate da **QObject** (la classe base di tutto il framework<sub>G</sub>), possono contenere *Signals* e *Slots*.

In Qt<sub>G</sub>, viene emesso un *Signal* nel momento in cui avviene un particolare evento. Le classi derivate da **QWidget** (è la classe base da cui derivano tutti gli oggetti grafici) hanno *Signals* predefiniti, ma è sempre possibile creare delle sottoclassi che ereditano dalle precedenti ed implementare i *Signals* che si desiderano. Gli *Slots* sono delle funzioni che vengono invocate in risposta ad un particolare *Signal*. Anche in questo caso, alcune classi hanno degli *Slots* predefiniti, ma è possibile crearne di personalizzati ereditando le classi opportune. Per collegare il *Signal* di un oggetto con lo *Slot* di un altro, è necessario utilizzare la primitiva (vedi fig. 1):

```
connect(Object_x, signal_1, Object_y, slot_2)
```



**Figura 1:** Schema interazione oggetti tramite *Signals & Slots*

Questo meccanismo è *type safe*: la segnatura del *Signal* deve combaciare con la segnatura dello *Slot* ad esso associato. Inoltre queste due entità sono estremamente disaccoppiate, nel senso che l'oggetto emettente il *Signal*, non si preoccupa del fatto che verrà raccolto o meno e di chi lo raccoglierà. La responsabilità è completamente spostata sullo sviluppatore.

### 2.2.2 The Meta-Object System

Il *Meta-Object System* fornisce il meccanismo di *Signals & Slots* di Qt<sub>G</sub> ed inoltre, offre supporto per l'RTTI<sub>G</sub> e per altre operazioni eseguite dinamicamente. Il sistema si basa su tre componenti fondamentali:

1. **QObject:** è una classe che funge da classe-base per oggetti che possono sfruttare il sistema;
2. **Q\_OBJECT:** è una macro inserita all'interno della sezione privata della dichiarazione di una classe. È usata per abilitare le funzioni dei Meta-Oggetti, quali RTTI<sub>G</sub>, *Signals* e *Slots*
3. **MOC (The Meta-Object Compiler):** è un pre-processor che fornisce ad ogni sottoclasse di **QObject**, il codice sorgente necessario ad implementare le caratteristiche dei Meta-Oggetti.

Il MOC legge i sorgenti C++. Se trova uno o più dichiarazioni di classe che contengono la macro **Q\_OBJECT**, produce un sorgente C++ contenente il codice dei Meta-Oggetti, per ognuna di queste classi. Questi file generati dal MOC, possono essere inclusi nei sorgenti da cui derivano (tramite le direttive di **#include** inserite automaticamente) oppure, più usualmente, vengono compilati e linkati con le implementazioni delle classi.

### 2.2.3 Licenza

Il framework<sub>G</sub> Qt<sub>G</sub> viene distribuito sotto tre diverse tipologie di licenze, in maniera da andare incontro alle esigenze dei vari team di sviluppo.

- Una versione commerciale, di proprietà del **Qt Digia**, adatta per sviluppare software proprietario anche a fini commerciali, in cui non è previsto il rilascio del codice sorgente;

- La licenza **LGPL v2.1** (*GNU Lesser General Public License*)<sup>1</sup>. È una licenza di software libero, che permette alle classi della libreria di essere linkate da codice non libero;
- La licenza **GPL v3.0** (*GNU General Public License*)<sup>2</sup>. È una licenza di software libero che garantisce all'utente la libertà di utilizzo, copia, modifica e distribuzione del prodotto. Permette inoltre di integrare il progetto con le librerie dotate di licenza ad essa compatibile. Dato che gli altri framework<sub>G</sub> con cui si svilupperà Romeo, sono licenziati compatibilmente con la GPL, si è deciso di adottare Qt<sub>G</sub> con questa licenza.

## 2.3 ITK

Si è deciso di utilizzare la libreria ITK<sub>G</sub>, per implementare alcune operazioni fondamentali che il software deve svolgere. In particolare, fornisce delle classi che permettono di importare svariate tipologie di formati di immagini nel sistema e conseguentemente di esportarle. Inoltre, sono disponibili delle classi che consentono di memorizzare degli algoritmi che operano sulle immagini (chiamate filtri). Quest'ultima caratteristica risulta fondamentale per implementare le feature extractors<sub>G</sub> e gli algoritmi di clustering<sub>G</sub>.

ITK<sub>G</sub> è stato scelto principalmente per i seguenti motivi:

- Indicazione dei proponenti;
- Libreria interamente scritta in C++ e quindi facilmente integrabile con Qt<sub>G</sub>;
- Libreria ideata per operare su dati biomedici; integra quindi la possibilità di manipolare immagini e dati provenienti da fMRI<sub>G</sub> ecc...;
- Fornita con licenza Apache v2.0, una licenza di software libero compatibile con la GPL v3.0;
- Libreria multiplatforma.

## 2.4 VTK

Si è deciso di utilizzare la libreria VTK<sub>G</sub>, per implementare le funzioni di visualizzazione delle immagini biomediche che il software dovrà supportare. In particolare, fornisce delle classi che permettono di manipolare immagini con formato Analyze 7.5<sub>G</sub> e NIfTI<sub>G</sub> e di poterle quindi importare e visualizzare.

VTK<sub>G</sub> è stato scelto principalmente per i seguenti motivi:

- Indicazione dei proponenti;
- Libreria interamente scritta in C++ e compatibile con Qt<sub>G</sub>;
- Libreria multiplatforma;
- Ideata per operare su dati biomedici.

---

<sup>1</sup><https://qt-project.org/doc/qt-5.0/qtdoc/lgpl.html>

<sup>2</sup><https://qt-project.org/doc/qt-5.0/qtdoc/gpl.html>

## 2.5 SQLite

Per implementare il database su cui il software andrà ad operare, si è deciso di utilizzare SQLite. Quest'ultima è una libreria software scritta in linguaggio C, che definisce un DBMS SQL incorporabile all'interno di applicazioni.

Le principali motivazioni che hanno portato a questa scelta sono:

- SQLite supporta la specifica standard SQL 92, di cui ogni componente del gruppo ha già avuto esperienza;
- L'installazione è compatta e leggera, infatti occupa solo 256KB di memoria;
- È autosufficiente, nel senso che non necessita di un server;
- È una libreria di dominio pubblico;
- L'intero database è immagazzinato in un unico file. Per questo, SQLite non diffonde files all'interno del calcolatore.

## 3 Descrizione architettura

### 3.1 Premesse

Si procederà alla descrizione dell'architettura realizzata, utilizzando un approccio di tipo top-down, ovvero iniziando da una panoramica delle componenti macroscopiche, per arrivare poi a considerare le componenti più specifiche. Di conseguenza, verranno descritti i `packageG` ed i componenti macroscopici per entrare successivamente nel dettaglio delle singole classi, specificando per ognuna: una breve descrizione, il contesto di utilizzo, le dipendenze entranti e/o uscenti e le eventuali classi da cui eredita. Verranno successivamente messi in rilievo gli utilizzi dei Design Pattern<sub>G</sub> all'interno dell'architettura del sistema, dando una spiegazione più dettagliata degli stessi, in appendice A.

Per i diagrammi dei `packageG`, delle classi e di attività, è stato utilizzato lo standard UML<sub>G</sub> 2.0. Nei diagrammi dei `packageG` si è fatto uso, ove ritenuto necessario, di colori diversi per migliorare la leggibilità dei diversi componenti.

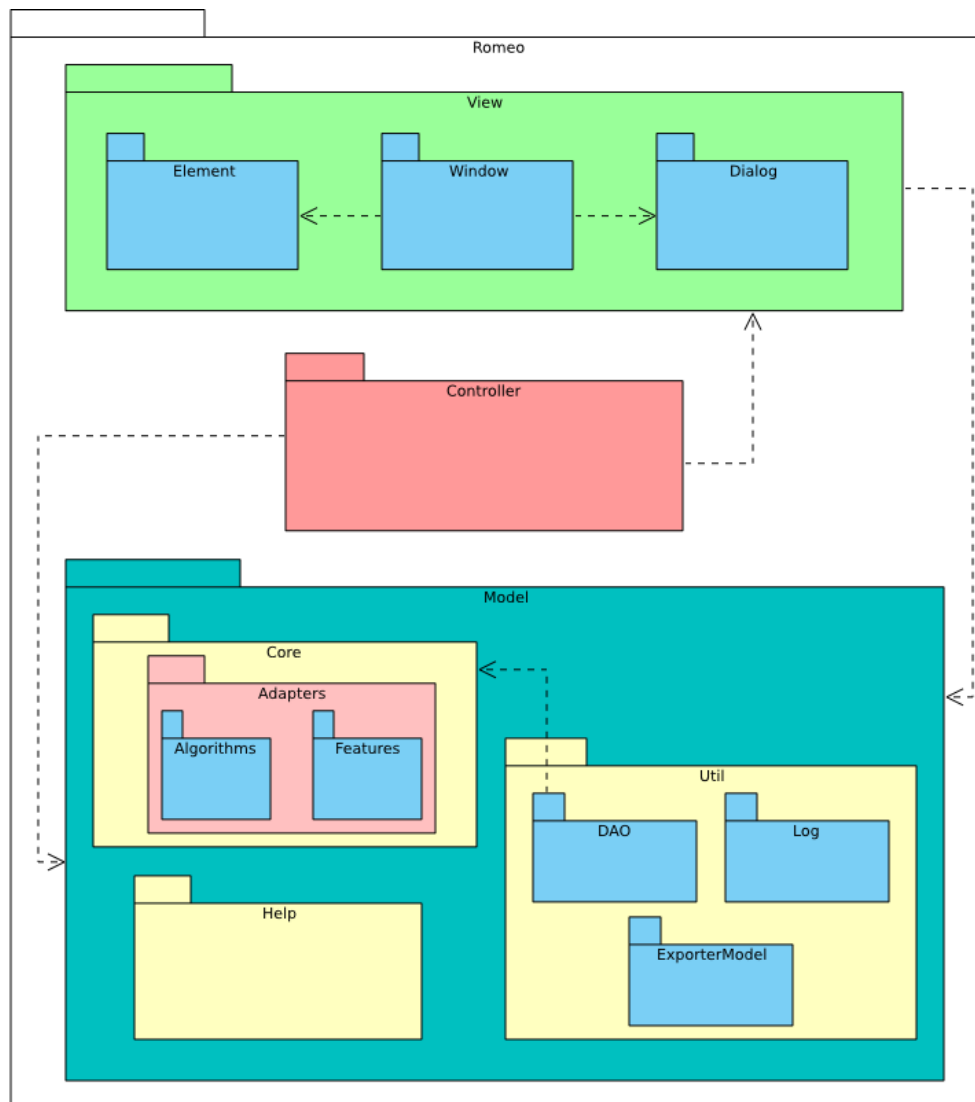
Si è deciso di non implementare l'architettura Model/View fornita da Qt<sub>G</sub>, in quanto è emersa la volontà di separare maggiormente la rappresentazione grafica della View dalla propria gestione degli eventi.

### 3.2 Architettura generale

L'architettura del software segue quanto stabilito dal design pattern  $MVC_G$ , ed è quindi suddivisa nelle seguenti parti:

- **Model:** rappresenta la logica di *business*;
- **View:** visualizza i dati all'utente;
- **Controller:** rappresenta la logica *applicativa*.

Nella seguente figura viene presentata l'architettura ad alto livello dell'applicazione, indicando i vari package  $G$  e le varie relazioni tra di essi.

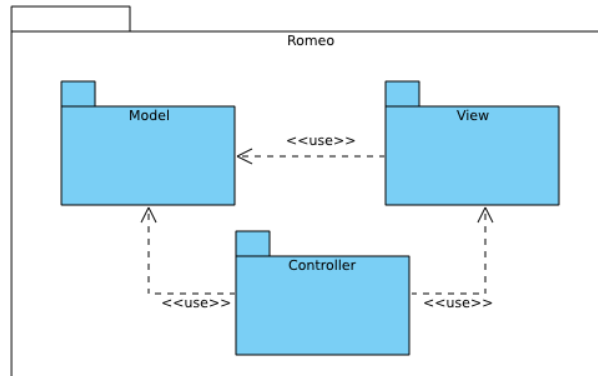


**Figura 2:** Vista package dell'architettura di Romeo

## 4 Componenti e classi

### 4.1 Romeo

#### 4.1.1 Informazioni sul package



**Figura 3:** Componente Romeo

#### 4.1.2 Descrizione

Il package<sub>G</sub> Romeo rappresenta il package<sub>G</sub> globale del progetto.

Le relazioni tra i package<sub>G</sub> Model, View e Controller rappresentano le relazioni tipiche del design pattern<sub>G</sub> MVC<sub>G</sub>.

#### 4.1.3 Package contenuti

- Romeo::Model;
- Romeo::View;
- Romeo::Controller.

#### 4.1.4 Relazioni d'uso tra i componenti

I package<sub>G</sub> contenuti nel package<sub>G</sub> Romeo, rispettano il design pattern<sub>G</sub> MVC. In particolare, il Model viene utilizzato dal Controller e dalla View; il primo per modificare i dati a seguito di un'interazione con l'utente, il secondo per visualizzarli. Inoltre, il Controller si relaziona anche con la View per aggiornare i dati.



## 4.2 Romeo::Model

### 4.2.1 Informazioni sul package

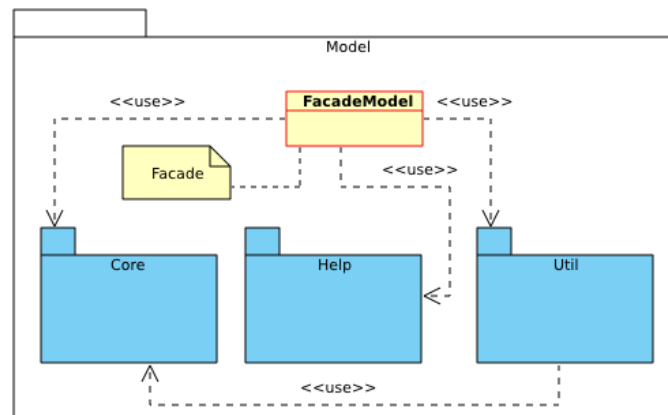


Figura 4: Componente Romeo::Model

### 4.2.2 Descrizione

Package<sub>G</sub> che rappresenta la componente model nell'architettura MVC<sub>G</sub>.

### 4.2.3 Package contenuti

- Romeo::Model::Core;
- Romeo::Model::Util;
- Romeo::Model::Help.

### 4.2.4 Relazioni d'uso tra i componenti

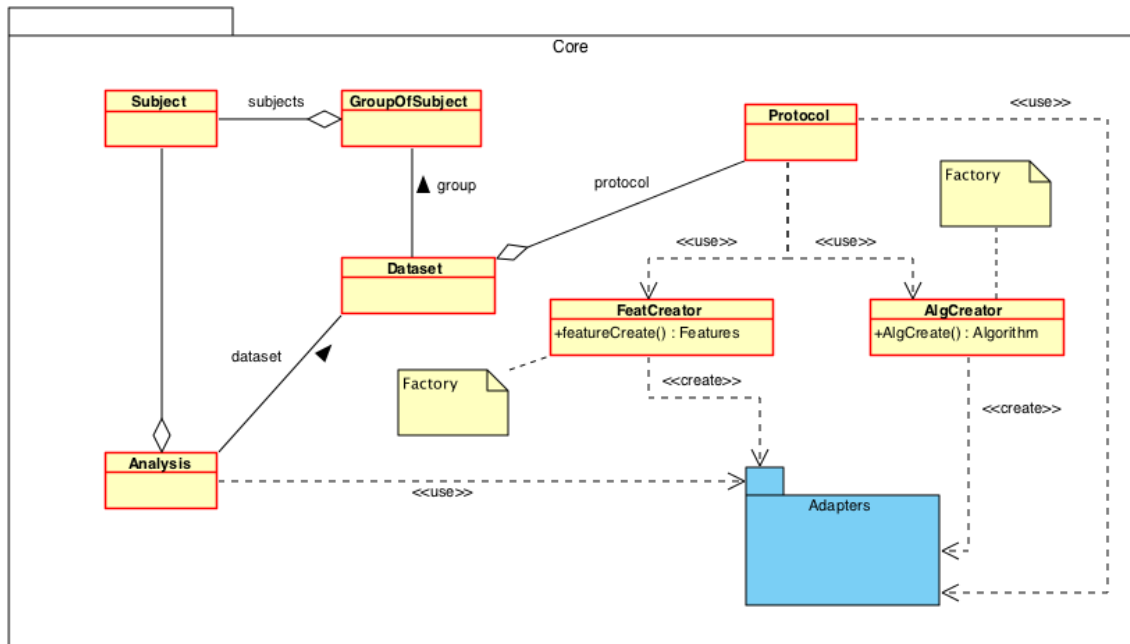
La classe *FacadeModel* si occupa di gestire tutte le richieste entranti o uscenti dal package<sub>G</sub>. In particolare, smista le richieste d'uso per i componenti dei package<sub>G</sub> Core, Help ed Util. Inoltre, il package<sub>G</sub> Util ha bisogno di relazionarsi con il package<sub>G</sub> Core per avere un riferimento dei tipi interni a quest'ultimo.

### 4.2.5 Classi contenute

#### FacadeModel

**Descrizione:** classe che rappresenta il punto di accesso per il componente Model. È la componente Facade del model ed è implementata tramite il design pattern<sub>G</sub> Singleton.

**Contesto d'utilizzo:** viene utilizzata per accedere Model in modo protetto, nascondendo l'implementazione delle varie operazioni e parte della struttura del model.



**Descrizione:** classe che rappresenta un  $\text{Subject}_G$  con le relative proprietà: *Nome*, *Nome dell'immagine/video* ed eventualmente una *Maschera*.

**Contesto di utilizzo:** viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessitano di riferirsi ad uno o più  $\text{Subject}_G$ .

### GroupOfSubjects

**Descrizione:** classe che rappresenta un gruppo di  $\text{Subject}_G$  con le relative proprietà: *Nome del Gruppo*, *Tipo del Gruppo* e la lista dei  $\text{Subject}_G$ .

**Contesto di utilizzo:** viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessitano di riferirsi ad uno o più gruppi di  $\text{Subject}_G$ .

### Protocol

**Descrizione:** classe che rappresenta un  $\text{Protocol}_G$  con le relative proprietà: *Nome*, *Tipo*, *Lista di feature $_G$*  e *Algoritmo di cluster $_G$* .

**Contesto di utilizzo:** viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessitano di riferirsi ad uno o più  $\text{Protocol}_G$ .

### Dataset

**Descrizione:** classe che rappresenta un  $\text{Dataset}_G$  con le relative proprietà: *Nome*, *Gruppo di  $\text{Subject}_G$*  e *Protocol $_G$* .

**Contesto di utilizzo:** viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessitano di riferirsi ad uno o più  $\text{Dataset}_G$ .

### Analysis

**Descrizione:** classe che rappresenta un'Analisi $_G$  con le relative proprietà: *Subject $_G$*  da analizzare, *Feature $_G$*  di cui salvare i risultati e *Feature $_G$*  di cui visualizzare i risultati.

**Contesto di utilizzo:** viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessitano di utilizzare un oggetto Analisi.

## 4.4 Romeo::Model::Core::Adapters

### 4.4.1 Informazioni sul package

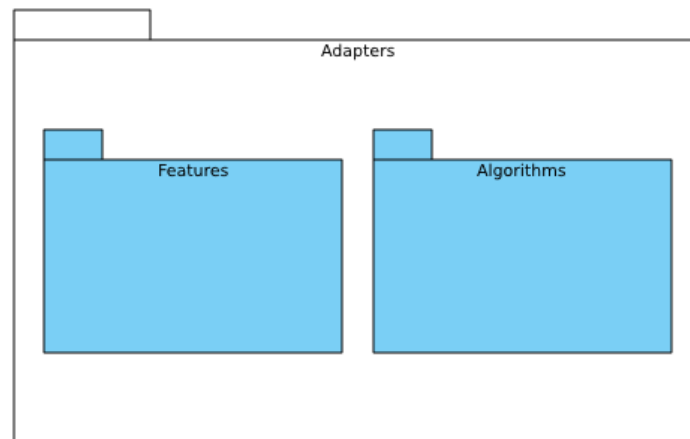


Figura 6: Componente Romeo::Model::Core::Adapters

### 4.4.2 Descrizione

Package<sub>G</sub> contenente due sottopackage relativi agli algoritmi di clustering<sub>G</sub> e alle feature<sub>G</sub> utilizzati dal sistema per l'analisi.

### 4.4.3 Package contenuti

- Romeo::Model::Core::Adapters::Features
- Romeo::Model::Core::Adapters::Algorithms

## 4.5 Romeo::Model::Core::Adapters::Features

### 4.5.1 Informazioni sul package

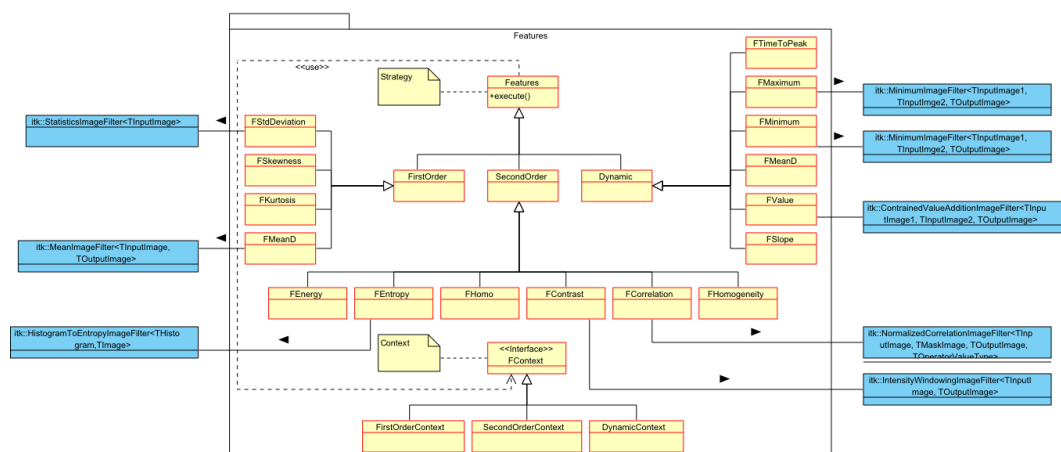


Figura 7: Componente Romeo::Model::Core::Adapters::Features

### 4.5.2 Descrizione

Package<sub>G</sub> contenente le classi che permettono al model di utilizzare gli algoritmi previsti nei requisiti. In questo package<sub>G</sub> verrà implementato il design pattern<sub>G</sub> Adapter che permette di utilizzare funzionalità di librerie esterne.

### 4.5.3 Relazioni d'uso tra i componenti

- La classe Feature avrà un riferimento di tipo FContext: ogni classe che concretizzerà una delle classi astratte della gerarchia di feature<sub>G</sub>, ridefinirà il riferimento, facendolo puntare al relativo context;
- FStdDeviation avrà un riferimento a ITK::StaticsImageFilter <TInputImage>;
- FMeanD avrà un riferimento a ITK::MeanImageFiter <TInputImage,TOutputImage>;
- FEntropy avrà un riferimento a ITK::HistogramToEntropyImageFilter < THistogram,TImage>;
- FContrast avrà un riferimento a ITK::IntensityWindowingImageFilter <TInputImage,TOutputImage>;
- FCorrelation avrà un riferimento a ITK::NormalizedCorrelationImageFilter <TInputImage,TMaskImage,TOutputImage,TOperatorValueType>;
- FValue avrà un riferimento a ITK::ContrainedValueAdditionImageFilter <TInputImage1,TInputImage2,TOutputImage>;
- FMiniumum avrà un riferimento a ITK::MinimumImageFilter <TInputImage1, TInputImage2, TOutputImage>;
- FMaximum avrà un riferimento a ITK::MaximumImageFilter <TInputImage1, TInputImage2,TOutputImage>.

### 4.5.4 Classi contenute

**Features** classe astratta che fornisce un contratto per l'applicazione di una feature<sub>G</sub>. Viene ereditata dalle classi astratte che rappresentano logicamente un raggruppamento delle feature<sub>G</sub> aventi determinate caratteristiche in comune. Viene ereditata da:

- Romeo::Model::Core::Adapters::Features::FirstOrder
- Romeo::Model::Core::Adapters::Features::SecondOrder
- Romeo::Model::Core::Adapters::Features::Dynamic

**FirstOrder:** classe astratta che rappresenta le feature<sub>G</sub> aventi un unico parametro *window size* e che vengono applicate a immagini di tipo statico. Le classi che ereditano da essa sono:

- Romeo::Model::Core::Adapters::Features::FStdDeviation;
- Romeo::Model::Core::Adapters::Features::FSkewness;
- Romeo::Model::Core::Adapters::Features::FKurtosis;
- Romeo::Model::Core::Adapters::Features::FMeanD.

**SecondOrder:** classe astratta che rappresenta le feature<sub>G</sub> aventi due parametri: *window size* e *GLCM*. Queste feature<sub>G</sub> vengono applicate a immagini di tipo statico; viene ereditata dalle seguenti classi:

- Romeo::Model::Core::Adapters::Features::FEnergy;
- Romeo::Model::Core::Adapters::Features::FEntropy;
- Romeo::Model::Core::Adapters::Features::FHomo;
- Romeo::Model::Core::Adapters::Features::FContrast;
- Romeo::Model::Core::Adapters::Features::FCorrelation;
- Romeo::Model::Core::Adapters::Features::FHomogeneity.

**Dynamic:** classe astratta che rappresenta le feature<sub>G</sub> aventi due parametri: *frame di inizio* e *frame di fine*. Queste feature<sub>G</sub> vengono applicate a immagini di tipo time dependent; viene ereditata dalla seguenti classi:

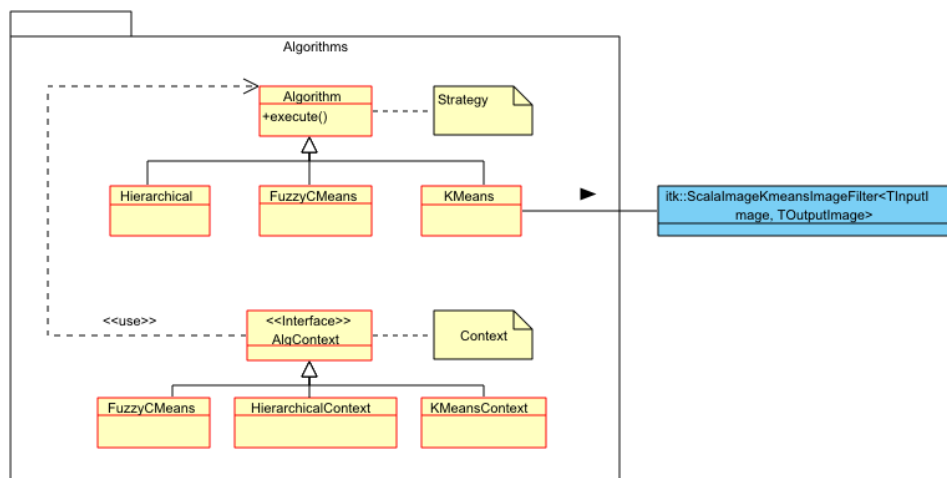
- Romeo::Model::Core::Adapters::Features::FTimeToPeak;
- Romeo::Model::Core::Adapters::Features::FMaximum;
- Romeo::Model::Core::Adapters::Features::FMinimum;
- Romeo::Model::Core::Adapters::Features::FMeanD;
- Romeo::Model::Core::Adapters::Features::FValue;
- Romeo::Model::Core::Adapters::Features::FSlope.

**FContext** interfaccia che rappresenta il componente Context del design pattern<sub>G</sub> Strategy, dalla quale derivano le classi Context specifiche per ogni gerarchia di feature<sub>G</sub>. Da essa derivano quindi:

- Romeo::Model::Core::Adapters::Features::FirstOrderContext;
- Romeo::Model::Core::Adapters::Features::SecondOrderContext;
- Romeo::Model::Core::Adapters::Features::DynamicContext;

## 4.6 Romeo::Model::Core::Adapters::Algorithms

### 4.6.1 Informazioni sul package



**Figura 8:** Componente Romeo::Model::Core::Adapters::Algorithms

### 4.6.2 Descrizione

Package<sub>G</sub> contenente le classi che permettono al model di utilizzare gli algoritmi di clustering<sub>G</sub> previsti nei requisiti. In questo package<sub>G</sub> verrà implementato il design pattern<sub>G</sub> Adapter che permette di utilizzare funzionalità di librerie esterne.

### 4.6.3 Relazioni d'uso tra i componenti

- La classe Algorithms avrà un riferimento di tipo AlgContext: ogni classe che concretizzerà una delle classi astratte della gerarchia di algoritmi<sub>G</sub>, ridefinirà il riferimento, facendolo puntare al relativo context;
- KMeans avrà un riferimento a ITK::ScalaImageKmeansImageFilter <TInputImage,TOutputImage>.

### 4.6.4 Classi contenute

**Algorithms** classe astratta che fornisce un contratto per l'applicazione di una algoritmo di clustering<sub>G</sub>. Viene ereditata dalle seguenti classi, che rappresentano uno specifico algoritmo di clustering<sub>G</sub>:

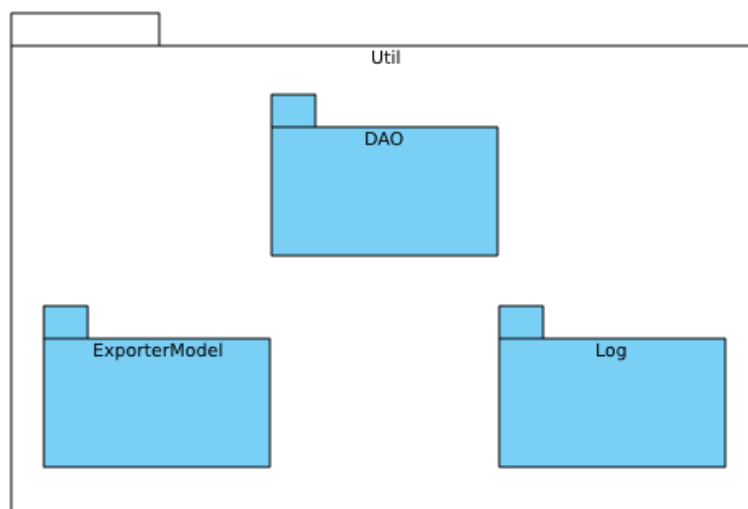
- Romeo::Model::Core::Adapters::Algorithms::KMeans
- Romeo::Model::Core::Adapters::Algorithms::FuzzyCMeans
- Romeo::Model::Core::Adapters::Algorithms::Hierarchical

**AlgContext** interfaccia che rappresenta il componente Context del design pattern<sub>G</sub> Strategy, dalla quale derivano le classi Context specifiche per ogni algoritmo di clustering<sub>G</sub>. Da essa derivano quindi:

- Romeo::Model::Core::Adapters::Algorithms::KMeansContext
- Romeo::Model::Core::Adapters::Algorithms::FuzzyCMeansContext
- Romeo::Model::Core::Adapters::Algorithms::HierarchicalContext

## 4.7 Romeo::Model::Util

### 4.7.1 Informazioni sul package



**Figura 9:** Componente Romeo::Model::Util

### 4.7.2 Descrizione

L'obiettivo di questo package<sub>G</sub> è quello di fornire una serie di classi di utilità per alcuni compiti.

### 4.7.3 Package contenuti

- Romeo::Model::Util::DAO;
- Romeo::Model::Util::ExporterModel;
- Romeo::Model::Util::Log.

## 4.8 Romeo::Model::Util::DAO

### 4.8.1 Informazioni sul package

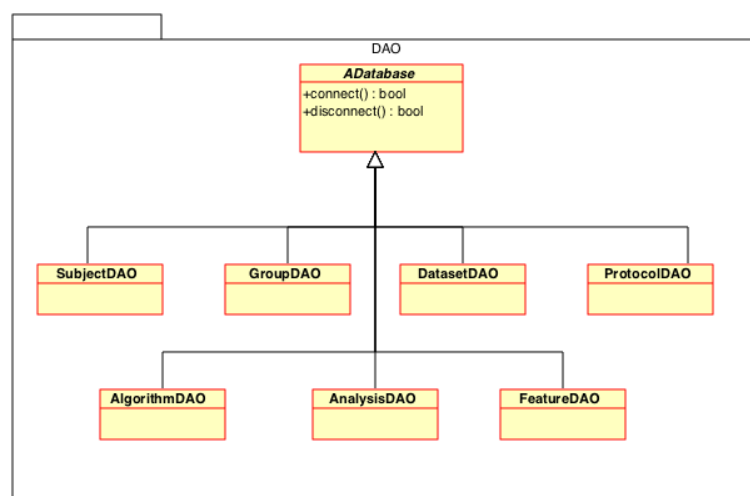


Figura 10: Componente Romeo::Model::Util::DAO

### 4.8.2 Descrizione

Package<sub>G</sub> che gestisce l'interfacciamento con la base dati del sistema<sup>3</sup>. Il database verrà utilizzato e gestito dal componente Romeo::Model::FacadeModel.

Per ogni tabella della base di dati è stata creata una classe che estenda la classe astratta ADatabase.

### 4.8.3 Classi contenute

**ADatabase** classe astratta che fornisce i metodi di connessione e disconnessione al database. Tale classe viene solamente estesa dalle classi che opereranno sulle tabelle del database. Viene ereditata dalle seguenti classi:

- Romeo::Model::Util::DAO::SubjectDAO
- Romeo::Model::Util::DAO::GroupDAO
- Romeo::Model::Util::DAO::DatasetDAO
- Romeo::Model::Util::DAO::ProtocolDAO

<sup>3</sup>Per maggiori informazioni vedere la sezione 5



- `Romeo::Model::Util::DAO::AlgorithmDAO`
- `Romeo::Model::Util::DAO::AnalysisDAO`
- `Romeo::Model::Util::DAO::FeatureDAO`

**SubjectDAO** classe che si occupa di effettuare query di interrogazione e di aggiunta riguardanti la tabella `SubjectG` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più `SubjectG` presenti nel database, oppure per l'aggiunta di un `SubjectG`. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

**GroupDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta, di eliminazione e di modifica, riguardanti la tabella `GroupG` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più `GroupG` presenti nel database, oppure per l'aggiunta o l'eliminazione di un `GroupG`. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

**DatasetDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta e di eliminazione, riguardanti la tabella `DatasetG` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più `DatasetG` presenti nel database, oppure per l'aggiunta o l'eliminazione di un `DatasetG`. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

**ProtocolDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta e di eliminazione, riguardanti la tabella `ProtocolG` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più `ProtocolG` presenti nel database, oppure per l'aggiunta o l'eliminazione di un `ProtocolG`. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

**AlgorithmDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta, di eliminazione e di modifica, riguardanti la tabella `cluster algorithm` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più algoritmi di `clusteringG` presenti nel database e usati in qualche `Protocol`, oppure per l'aggiunta o l'eliminazione di un record in tale tabella. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

**AnalysisDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta, di eliminazione e di modifica, riguardanti la tabella `Analysis` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più analisi presenti nel database, oppure per l'aggiunta di una nuova analisi. Eredita da:

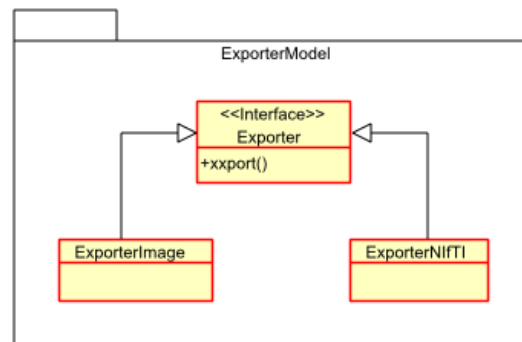
- `Romeo::Model::Util::DAO::ADatabase`

**FeatureDAO** classe che si occupa di effettuare query di interrogazione, di aggiunta, di eliminazione e di modifica, riguardanti la tabella `FeatureG` del database. Viene utilizzata ogni qualvolta il sistema richiede un'informazione riguardante uno o più `featureG` presenti nel database ed utilizzata in un `Protocol`, oppure per l'aggiunta o l'eliminazione di un record in tale tabella. Eredita da:

- `Romeo::Model::Util::DAO::ADatabase`

## 4.9 Romeo::Model::Util::ExporterModel

### 4.9.1 Informazioni sul package



**Figura 11:** Componente Romeo::Model::Util::ExporterModel

### 4.9.2 Descrizione

Package contenente le classi che si occupano di trasformare le immagini dal formato usato per l'analisi, al formato desiderato dall'utente, tra quelli previsti dai requisiti.

### 4.9.3 Classi contenute

**Export** interfaccia che fornisce il contratto un metodo per esportare i risultati delle analisi. Viene implementata dalle classi che specializzano l'esportazione per tipologia di file. Ereditata da:

- Romeo::Model::Util::ExporterModel::ExporterImage
- Romeo::Model::Util::ExporterModel::ExporterNifTi

**ExporterImage** classe che si occupa di esportare un immagine di tipo 2D, nel formato desiderato dall'utente tra quelli previsti dai requisiti e nel percorso indicato dall'utente. La classe viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessita di esportare un'immagine di tipo 2D. Eredita da:

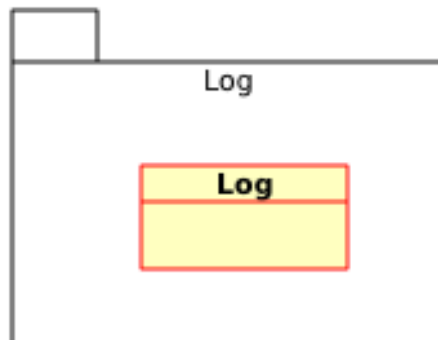
- Romeo::Model::Util::ExporterModel::Export

**ExporterNifTi** classe che si occupa di esportare un immagine di tipo NifTi, nel percorso indicato dall'utente. La classe viene utilizzata in seguito alla ricezione di un signal da parte dei controller che necessita di esportare un'immagine di tipo 3D. Eredita da:

- Romeo::Model::Util::ExporterModel::Export

## 4.10 Romeo::Model::Util::Log

### 4.10.1 Informazioni sul package



**Figura 12:** Componente Romeo::Model::Util::Log

### 4.10.2 Descrizione

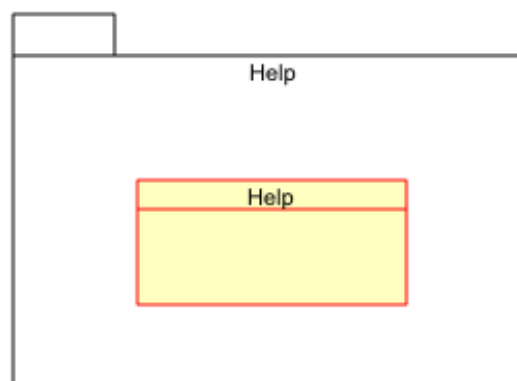
Package contenente la classe che si occupa di gestire e produrre dei file di log, nei quali saranno contenute informazioni utili agli sviluppatori.

### 4.10.3 Classi contenute

**Log** classe che si occupa di produrre e scrivere il file di log, con alcune operazioni che Romeo effettuerà. Nel file saranno quindi memorizzate informazioni riguardanti errori derivati dall'uso del programma, azioni dell'utente, esecuzione di analisi e interazione del sistema con il database. Vi sarà sempre attiva un'istanza della classe log, la quale sarà utilizzata dal package `Romeo::Model::Util::DAO`, dal package `Romeo::Model::Core` e dal package `Romeo::Controller`.

## 4.11 Romeo::Model::Help

### 4.11.1 Informazioni sul package



**Figura 13:** Componente Romeo::Model::Help

#### 4.11.2 Descrizione

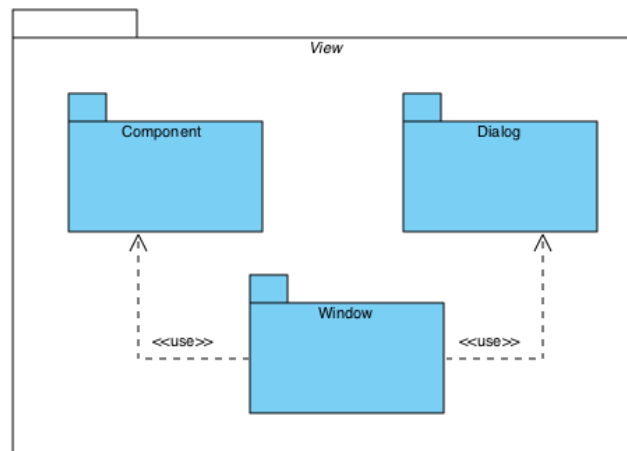
Package contenente la classe dedicata a caricare il contenuto dei file riguardanti la guida utente.

#### 4.11.3 Classi contenute

**Help** classe che viene utilizzata per caricare il contenuto dei file riguardanti la guida, qualora venisse richiesta dall'utente, e permette la ricerca di un termine all'interno della guida. La classe verrà creata nel momento in cui l'utente vorrà visualizzare la guida, attraverso la pressione del pulsante apposito. Tale oggetto verrà eliminato soltanto quando l'utente chiuderà la finestra in cui è visualizzata la guida.

## 4.12 Romeo::View

### 4.12.1 Informazioni sul package



**Figura 14:** Componente Romeo::View

### 4.12.2 Descrizione

Package<sub>G</sub> che rappresenta la componente View dell'architettura MVC<sub>G</sub>.

### 4.12.3 Package contenuti

- Romeo::View::Window;
- Romeo::View::Dialog;
- Romeo::View::Component.

### 4.12.4 Relazioni tra i componenti

Il package<sub>G</sub> Window utilizzerà vari componenti del package<sub>G</sub> Component e del package<sub>G</sub> Dialog per generare le varie finestre e dialoghi.

## 4.13 Romeo::View::Window

### 4.13.1 Informazioni sul package

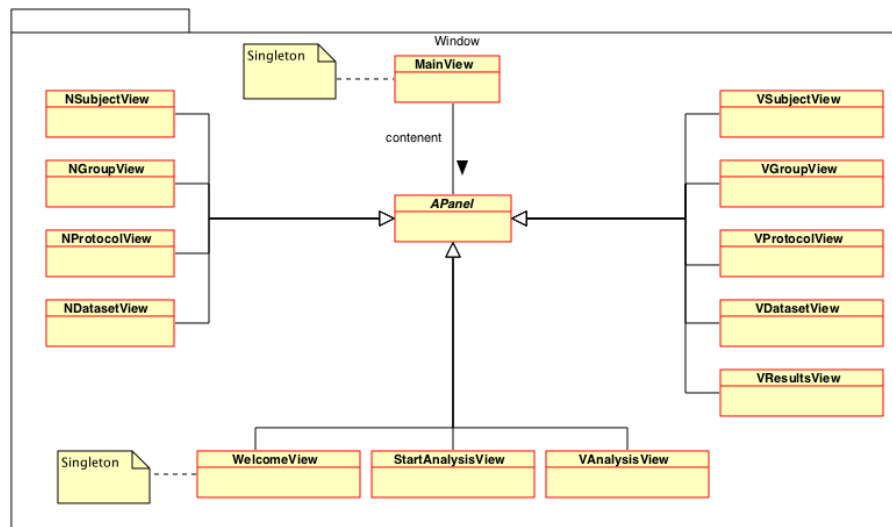


Figura 15: Componente Romeo::View::Window

### 4.13.2 Descrizione

Package<sub>G</sub> che contiene l'insieme delle “finestre” con le quali l'utente può interagire durante l'esecuzione di Romeo.

### 4.13.3 Relazioni tra i componenti

La classe MainView contiene un riferimento alla classe APanel, una classe astratta che rappresenta una generica finestra del programma.

### 4.13.4 Classi contenute

**MainView** classe che rappresenta la finestra principale dell'applicativo Romeo con la quale l'utente interagisce.

È implementata tramite l'utilizzo del design pattern<sub>G</sub> Singleton. Viene creata *unicamente* al primo avvio del programma e rimane attiva fino alla chiusura del programma.

**APanel** classe astratta che rappresenta un generico “widget” utilizzato dalla MainView come contenuto centrale. In un dato istante la MainView avrà sempre un *unico* widget. Essendo una classe astratta non verrà mai utilizzata direttamente, ma verrà estesa dalle classi che verranno utilizzate nella MainWindow.

#### Ereditato da:

- Romeo::View::Window::NSubjectView;
- Romeo::View::Window::NGroupView;
- Romeo::View::Window::NProtocolView;
- Romeo::View::Window::NDatasetView;
- Romeo::View::Window::WelcomeView;
- Romeo::View::Window::StartAnalysisView;

- `Romeo::View::Window::VAnalysisView;`
- `Romeo::View::Window::VSubjectView;`
- `Romeo::View::Window::VResultView;`
- `Romeo::View::Window::VProtocolView;`
- `Romeo::View::Window::VGroupView;`
- `Romeo::View::Window::VDatasetView.`

**WelcomeView** classe che rappresenta la view principale del sistema. Permette all'utente di scegliere una tra le varie funzionalità del programma. Implementa il design pattern `Singleton`, vista la frequenza con la quale l'utente interagirà con questa finestra e per evitare continue distruzioni e creazioni della classe. Viene creata all'avvio del programma e permette all'utente di selezionare una delle funzionalità presenti. Emette un signal in seguito alla scelta effettuata. Eredita da:

- `Romeo::View::Window::APanel.`

**NSubjectView** permette la creazione di un nuovo `SubjectG`. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, la funzionalità di creazione di un nuovo `SubjectG`. Emette un signal in seguito alla conferma, da parte dell'utente, della creazione di un nuovo `SubjectG`. Eredita da:

- `Romeo::View::Window::APanel.`

**NGroupView** permette la creazione di un nuovo gruppo di `SubjectG`. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, la funzionalità di creazione di un nuovo gruppo di `SubjectG`. Per essere creata necessita che almeno un `SubjectG` sia presente nel sistema. Emette un signal in seguito alla conferma, da parte dell'utente, della creazione di un nuovo gruppo di `SubjectG`. Eredita da:

- `Romeo::View::Window::APanel.`

**NProtocolView** permette la creazione di un nuovo `ProtocolG`. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, la funzionalità di creazione di un nuovo `ProtocolG`. Emette un signal in seguito alla conferma, da parte dell'utente, della creazione di un nuovo `ProtocolG`. Eredita da:

- `Romeo::View::Window::APanel.`

**NDatasetView** permette la creazione di un nuovo `DatasetG`. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, la funzionalità di creazione di un nuovo `DatasetG`. Per essere creata necessita che almeno un gruppo di `SubjectG` e un `ProtocolG`, siano presenti nel sistema. Emette un signal in seguito alla conferma, da parte dell'utente, della creazione di un nuovo gruppo di `DatasetG`. Eredita da:

- `Romeo::View::Window::APanel.`

**VSubjectView** permette la visualizzazione dei `SubjectG` memorizzati nel sistema. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, l'opzione di visualizzazione della lista dei `SubjectG`. Essa comunica con il Controller per acquisire la lista dei `SubjectG`. Eredita da:

- `Romeo::View::Window::APanel.`

**VGroupView** classe che rappresenta la view per la visualizzazione, l'eliminazione e la modifica dei vari gruppi di `SubjectG` presenti nel sistema. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, l'opzione di visualizzazione dei lista dei gruppi di `SubjectG`. Comunica con il relativo controller per ottenere la lista dei gruppi di `SubjectG` e per eliminare eventuali gruppi selezionati. Eredita da:

- `Romeo::View::Window::APanel`.

**VProtocolView** permette la visualizzazione e l'eliminazione dei `ProtocolG` presenti nel sistema. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, l'opzione di visualizzazione dei lista dei `ProtocolG`. Comunica con il relativo controller per visualizzare i dettagli di un `ProtocolG` selezionato e per eliminare eventuali `ProtocolG` selezionati. Eredita da:

- `Romeo::View::Window::APanel`.

**VDatasetView** classe che rappresenta la view per la visualizzazione e l'eliminazione dei vari `DatasetG` esistenti. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, l'opzione di visualizzazione dei lista dei `DatasetG`. Comunica con il relativo controller per visualizzare i dettagli di un `DatasetG` e per eliminare eventuali `DatasetG` selezionati. Eredita da:

- `Romeo::View::Window::APanel`.

**StartAnalysisView** permette l'avvio di una nuova analisi. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, la funzionalità di avvio di una nuova analisi. Per essere creata necessita che almeno un `DatasetG` sia presente nel sistema. Emette un signal in seguito all'interazione con l'utente, per l'avvio dell'analisi. Eredita da:

- `Romeo::View::Window::APanel`.

**VAnalysisView** permette la visualizzazione dei risultati delle analisi precedentemente effettuate. Viene creata ogni qualvolta l'utente seleziona, dalla `WelcomeView`, l'opzione di visualizzazione dei risultati. Comunica con il relativo controller per visualizzare la lista dei risultati. Eredita da:

- `Romeo::View::Window::APanel`.

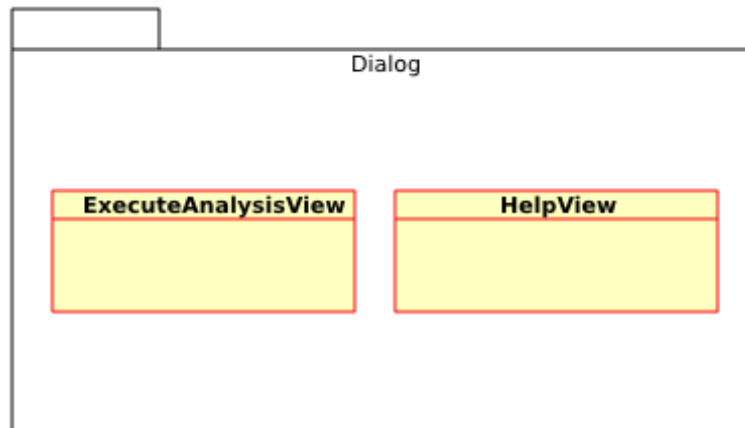
**VResultView** permette la visualizzazione del risultato di una specifica analisi selezionata. Viene creata conseguentemente alla selezione di una specifica analisi, dalla lista delle analisi effettuate, presente nella `VAnalysisView`. Comunica inoltre con il relativo controller per acquisire la lista dei `SubjectG` e dei `ProtocolG` coinvolti nell'analisi e per avviare l'esportazione dei risultati. Eredita da:

- `Romeo::View::Window::APanel`.



## 4.14 Romeo::View::Dialog

### 4.14.1 Informazioni sul Package



**Figura 16:** Componente Romeo::View::Dialog

### 4.14.2 Descrizione

Package<sub>G</sub> che contiene l'insieme dei “dialog” con i quali l'utente può interagire durante l'esecuzione di Romeo.

### 4.14.3 Classi contenute

**HelpView** questa classe rappresenta la finestra nella quale verrà visualizzata la guida interattiva richiesta dall'utente e nella quale egli potrà cercare la sezione di aiuto desiderata. Viene creata ogni qualvolta l'utente seleziona il relativo componente, all'interno di una view. Comunica con il relativo controller attraverso il meccanismo Signal e Slot delle Qt<sub>G</sub>.

**ExecuteAnalysisView** questa classe rappresenta la finestra nella quale verranno mostrati i risultati intermedi di un'analisi durante la sua esecuzione. Permette inoltre di interrompere l'analisi oppure di fermare la visualizzazione dei risultati intermedi. Contiene una barra di avanzamento che si aggiorna dinamicamente all'avanzare dell'analisi, informando l'utente sullo stato della stessa. Viene creata ogni qualvolta l'utente seleziona la funzione di avvio analisi dalla finestra VAnalysisView. Emette signal verso il relativo controller, riguardo alle opzioni di visualizzazione dei risultati, oltre che per l'annullamento dell'analisi.

## 4.15 Romeo::View::Component

### 4.15.1 Informazioni sul package

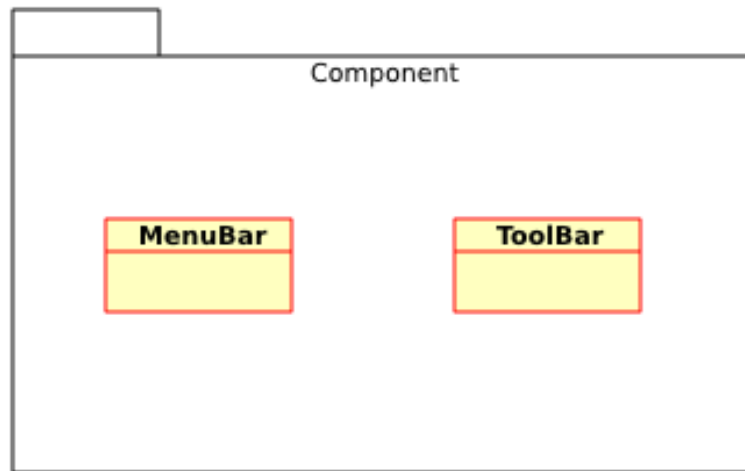


Figura 17: Componente Romeo::View::Window

### 4.15.2 Descrizione

Package `G` contenente le classi comuni a tutte le “view”.

### 4.15.3 Classi contenute

**MenuBar** classe che rappresenta la classica *menu bar*, con la quale l’utente può interagire in qualsiasi momento. Viene creata dalla `MainWindow`, all’avvio dell’applicazione e contiene dei componenti che permettono all’utente di interagire con le funzionalità del programma.

**ToolBar** classe che rappresenta una *toolbar* che aggiunge delle funzionalità per la visualizzazione dei dati, nelle finestre di visualizzazione/modifica degli elementi. Viene visualizzata nelle seguenti classi: `VSubjectView`, `VGroupView`, `VProtocolView`, `VDatasetView`, `VAnalysisView` e `VResultView`.

## 4.16 Romeo::Controller

### 4.16.1 Informazioni sul package

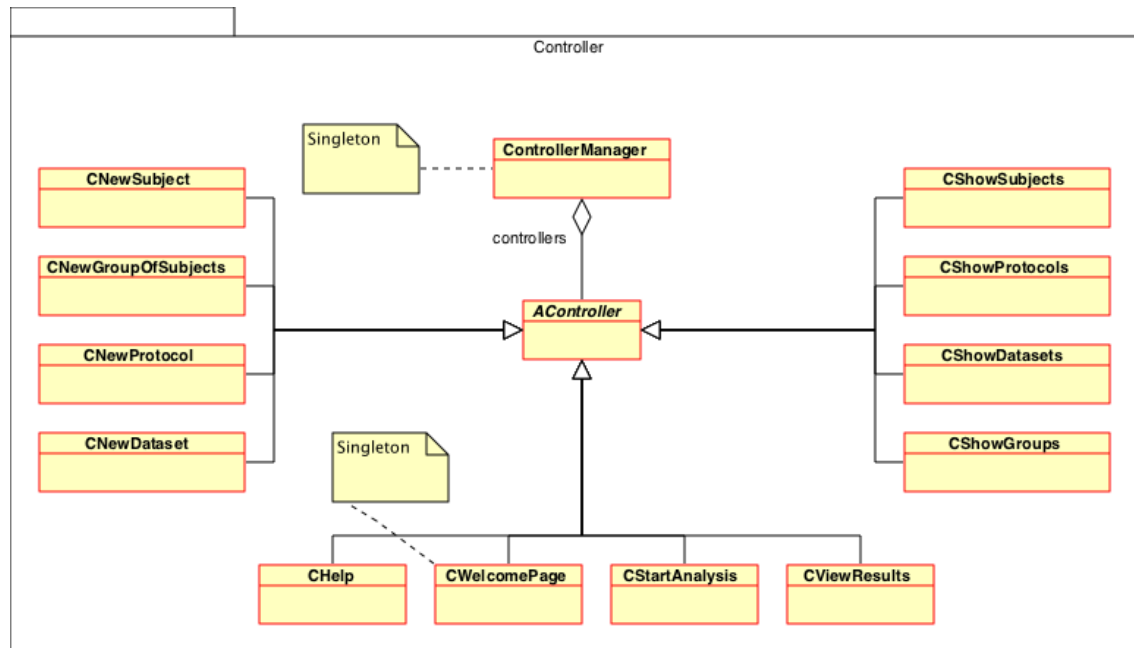


Figura 18: Componente Romeo::Controller

### 4.16.2 Descrizione

Package<sub>G</sub> che rappresenta la componente Controller dell'architettura MVC<sub>G</sub>.

### 4.16.3 Relazioni tra i componenti

La classe ControllerManager contiene dei riferimenti ai vari controller attivi.

### 4.16.4 Classi contenute

**ControllerManager** classe che implementa il design pattern Singleton e che contiene la lista dei controller attivi in un determinato momento. Viene utilizzata per gestire l'eliminazione di un determinato controller, attraverso il meccanismo Signal e Slot delle Qt<sub>G</sub>.

**AController** classe *astratta* che rappresenta un generico controller dell'applicazione Romeo.

Ereditato da:

- Romeo::Controller::CNewSubject;
- Romeo::Controller::CNewGroupOfSubjects;
- Romeo::Controller::CNewProtocol;
- Romeo::Controller::CNewDataset;
- Romeo::Controller::CShowSubjects;
- Romeo::Controller::CShowProtocols;

- **Romeo::Controller::CShowDatasets;**
- **Romeo::Controller::CShowGroups;**
- **Romeo::Controller::CHelp;**
- **Romeo::Controller::CWelcomePage;**
- **Romeo::Controller::CStartAnalysis;**
- **Romeo::Controller::CViewResults.**

**CNewSubject** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo Subject<sub>G</sub>. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un Subject<sub>G</sub> e per invocare i relativi metodi del Model.

**Eredita da:**

- **Romeo::Controller::AController**

**CNewGroupOfSubject** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo gruppo di Subject<sub>G</sub>. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un gruppo di Subject<sub>G</sub> e per invocare i relativi metodi del Model.

**Eredita da:**

- **Romeo::Controller::AController**

**CNewProtocol** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo Protocol<sub>G</sub>. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un Protocol<sub>G</sub> e per invocare i relativi metodi del Model.

**Eredita da:**

- **Romeo::Controller::AController**

**CNewDataset** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo Dataset<sub>G</sub>. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un nuovo Dataset<sub>G</sub> e per invocare i relativi metodi del Model.

**Eredita da:**

- **Romeo::Controller::AController**

**CShowSubjects** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei Subject<sub>G</sub> presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione dei dettagli del Subject<sub>G</sub> selezionato, invocando i relativi metodi del Model.

**Eredita da:**

- **Romeo::Controller::AController**

**CShowGroups** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei gruppi di `SubjectG` presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione e la gestione dei `SubjectG` facenti parte del gruppo selezionato e per invocare i relativi metodi del Model. Inoltre gestirà i signal relativi alla modifica ed eliminazione dei gruppi di `SubjectG`.

**Classi ereditate:**

- **Romeo::Controller::AController**

**CShowProtocols** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei `ProtocolG` presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione ed eliminazione dei `ProtocolG` facenti parte del sistema. Inoltre gestisce i signal per la visualizzazione delle informazioni sulle `FeatureG` ed algoritmi di clustering<sub>G</sub> che formano il `ProtocolG` selezionato.

**Eredita da:**

- **Romeo::Controller::AController**

**CShowDatasets** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei `DatasetG` presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione ed eliminazione dei `DatasetG` facenti parte del sistema. Gestisce inoltre i signal per la visualizzazione dei protocol, dei subject e informazioni aggiuntive sul dataset.

**Eredita da:**

- **Romeo::Controller::AController**

**CHelp** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione della guida interattiva. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione della guida.

**Eredita da:**

- **Romeo::Controller::AController**

**CWelcomePage** classe che implementa il design pattern<sub>G</sub> Singleton; essa rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione pagina iniziale. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione della pagina selezionata, tramite dei pulsanti.

**Eredita da:**

- **Romeo::Controller::AController**

**CStartAnalysis** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante l'avvio e l'esecuzione di un'analisi<sub>G</sub>. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti l'avvio dell'analisi e l'esecuzione dell'analisi.

**Eredita da:**

- **Romeo::Controller::AController**

**CViewResults** classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione della lista delle analisi effettuate e dei dettagli dei risultati di ogni analisi. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione dei dettagli di un'analisi, l'esportazione completa dei risultati e un'eventuale ripetizione dell'analisi. Inoltre gestisce i Signal emessi dalla View dei risultati di una particolare analisi.

**Eredita da:**

- **Romeo::Controller::AController**

## 5 Database Romeo

Viene riportato in questa sezione il database dell'applicativo Romeo con le classi e le relazioni tra esse.

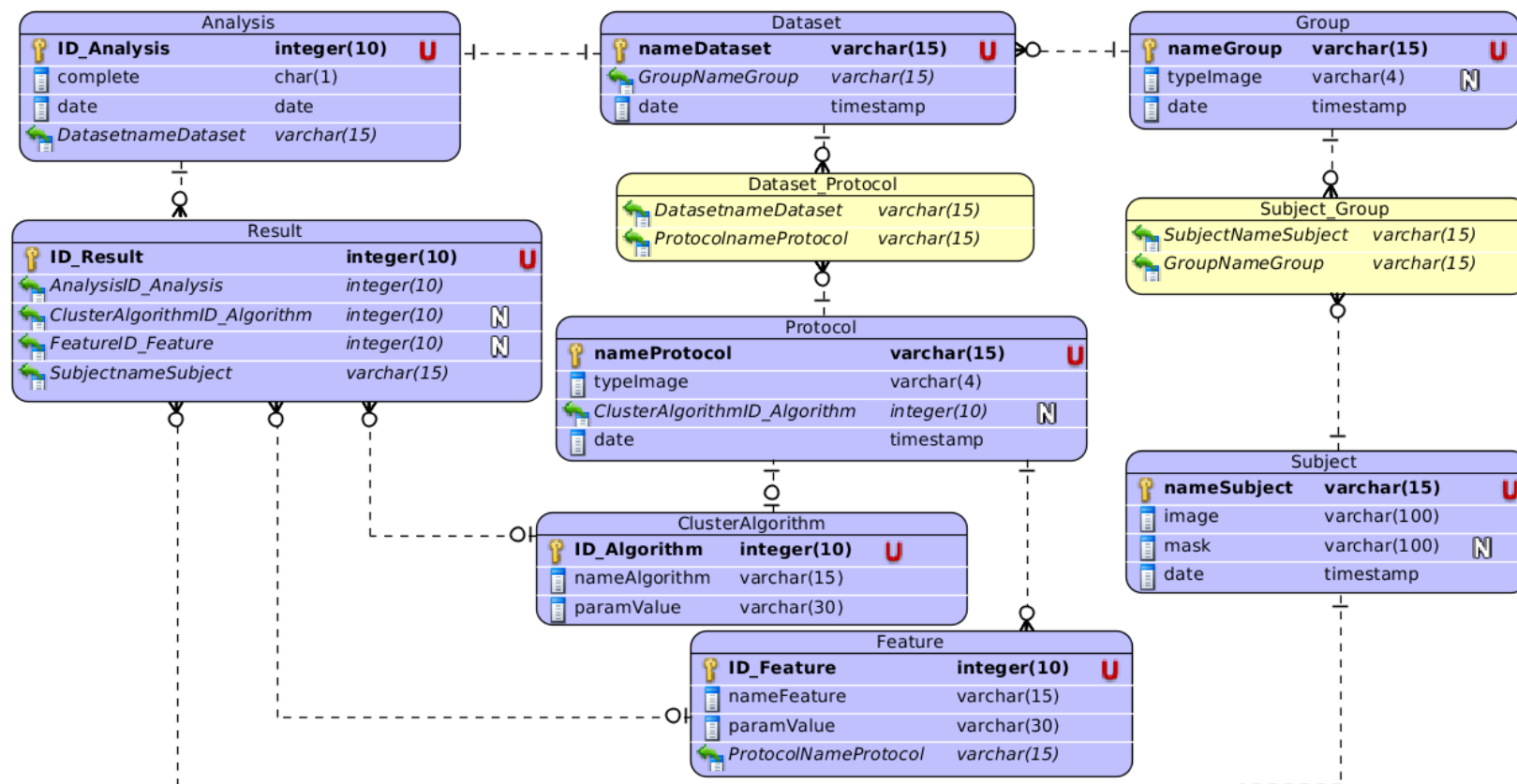


Figura 19: Struttura del database di Romeo

Il database ha il compito di mantenere lo storico delle operazioni effettuate dall'utente sul software, ovvero deve rendere disponibile tutto ciò che l'utente ha creato e modificato, ma non eliminato, dal momento dell'installazione di Romeo.

## 5.1 Descrizione testuale delle classi

### 5.1.1 Subject

La classe *Subject* raccoglie tutte le informazioni necessarie per identificare un *Subject<sub>G</sub>*.

#### Attributi:

- **nameSubject:** *varchar(15)* «PK» nome univoco, identificativo di un *Subject<sub>G</sub>*;
- **image:** *varchar(100)* memorizza il percorso del filesystem dove è presente l'immagine associata al *Subject<sub>G</sub>*;
- **mask:** *varchar(100)* memorizza il percorso del filesystem dove è presente la maschera associata al *Subject<sub>G</sub>*;
- **date:** *timestamp* memorizza la data e l'ora di creazione del *Subject<sub>G</sub>* all'interno del programma.

### 5.1.2 Group

La classe *Group* raccoglie tutte le informazioni necessarie per un gruppo di *Subject<sub>G</sub>*.

#### Attributi:

- **nameGroup:** *varchar(15)* «PK» nome univoco, identificativo di un gruppo di *Subject<sub>G</sub>*;
- **typeImage:** *varchar(4)* rappresenta il tipo di immagine che dovranno avere i *Subject<sub>G</sub>* contenuti all'interno del gruppo;
- **date:** *timestamp* memorizza la data e l'ora di creazione del gruppo di *Subject<sub>G</sub>* all'interno del software.

### 5.1.3 Dataset

La classe *Dataset* raccoglie tutte le informazioni necessarie per un *Dataset<sub>G</sub>*.

#### Attributi:

- **nameDataset:** *varchar(15)* «PK» nome univoco, identificativo di un *Dataset<sub>G</sub>*;
- **date:** *timestamp* memorizza la data e l'ora di creazione del *Dataset<sub>G</sub>* all'interno del software.

### 5.1.4 Protocol

La classe *Protocol* raccoglie tutte le informazioni necessarie per un *Protocol<sub>G</sub>*.

#### Attributi:

- **nameProtocol:** *varchar(15)* «PK» nome univoco, identificativo di *Protocol<sub>G</sub>*;
- **typeImage:** *varchar(4)* rappresenta il tipo di immagine a cui il *Protocol<sub>G</sub>* verrà applicato;
- **date:** *timestamp* memorizza la data e l'ora di creazione del *Protocol<sub>G</sub>* all'interno del programma.



### 5.1.5 ClusterAlgorithm

La classe *ClusterAlgorithm* contiene tutte le istanze degli algoritmi di clustering<sub>G</sub> che l'utente ha creato.

#### Attributi:

- **ID\_Algorithm:** *Integer(10)* «PK» codice univoco, identificativo di una particolare istanza di algoritmo di clustering<sub>G</sub>;
- **nameAlgorithm:** *varchar(15)* nome dell'algoritmo di clustering<sub>G</sub>;
- **paramValue:** *varchar(30)* elenco dei valori dei parametri separati da punto.

Si noti che, per istanza di algoritmi di clustering<sub>G</sub>, si intende un algoritmo di clustering<sub>G</sub> istanziato con determinati valori dei parametri, definiti dall'utente.

### 5.1.6 Feature

La classe *Feature* contiene tutte le istanze delle feature<sub>G</sub> che l'utente ha creato.

#### Attributi:

- **ID\_Feature:** *Integer(10)* «PK» codice univoco, identificativo di una particolare istanza di feature<sub>G</sub>;
- **nameAlgorithm:** *varchar(15)* nome della feature<sub>G</sub>;
- **paramValue:** *varchar(30)* elenco dei valori dei parametri separati da punto.

Si noti che, per istanza di feature<sub>G</sub>, si intende una feature<sub>G</sub> istanziata con determinati valori dei parametri, definiti dall'utente.

### 5.1.7 Analysis

La classe *Analysis* contiene tutte le informazioni relative alle analisi fatte.

#### Attributi:

- **ID\_Analysis:** *Integer(10)* «PK» codice univoco, identificativo di un'analisi;
- **complete:** *char(1)* se vale "y", l'analisi è stata portata a termine, altrimenti vale "n" e indica che l'analisi è stata interrotta;
- **date:** *timestamp* memorizza la data e l'ora di dell'ultima analisi eseguita dal programma.

### 5.1.8 Result

La classe *Result* contiene tutte le istanze dei risultati ottenuti dalle analisi effettuate.

#### Attributi:

- **ID\_Result:** *Integer(10)* «PK» codice univoco, identificativo di un risultato.

## 5.2 Descrizione delle associazioni

- **“contain”**: associazione tra *Group* e *Subject*. Un gruppo può contenere uno o più *Subject<sub>G</sub>* ed un *Subject<sub>G</sub>* può appartenere o meno a più gruppi;
- **“include”**: associazione tra *Dataset* e *Group*. Un *Dataset<sub>G</sub>* include un unico gruppo di *Subject<sub>G</sub>* ed un gruppo può essere incluso o meno in più *Dataset<sub>G</sub>*;
- **“hasA”**: associazione tra *Dataset* e *Protocol*. Un *Dataset<sub>G</sub>* include uno o più *Protocol<sub>G</sub>* ed un *Protocol<sub>G</sub>* può essere contenuto o meno in più *Dataset<sub>G</sub>*;
- **“prevedere”**: associazione tra *Protocol* e *ClusterAlgorithm*. Un *Protocol<sub>G</sub>* contiene al più un algoritmo di clustering<sub>G</sub> ed una particolare istanza di un algoritmo di clustering<sub>G</sub>, può essere presente in uno o più *Protocol<sub>G</sub>*;
- **“calculate”**: associazione tra *Protocol* e *Feature*. Un *Protocol<sub>G</sub>* può calcolare più feature<sub>G</sub> oppure nessuna ed una particolare istanza di una feature<sub>G</sub>, può essere presente in un solo *Protocol<sub>G</sub>*;
- **“isDoing”**: associazione tra *Analysis* e *Dataset*. Un’analisi viene fatta su un particolare *Dataset<sub>G</sub>* ed un *Dataset<sub>G</sub>* può essere presente o meno su più analisi;
- **“generate”**: associazione tra *Analysis* e *Result*. Un’analisi genera uno o più risultati ed un particolare risultato è generato da una sola analisi;
- **“associated”**: associazione tra *Result* e *Subject*. Un particolare risultato è associato ad un solo *Subject<sub>G</sub>* ed un *Subject<sub>G</sub>* può avere associato o meno a più risultati;
- **“resultof”**: associazione tra *Result* e *ClusterAlgorithm*. Un risultato può derivare dall’applicazione di un algoritmo di clustering<sub>G</sub> e quest’ultimo può produrre più risultati;
- **“apply”**: associazione tra *Result* e *Feature*. Un risultato può derivare dall’applicazione di una feature<sub>G</sub> e quest’ultima può avere più risultati nel momento in cui viene associata a *Subject<sub>G</sub>* diversi.

## 5.3 Progettazione logica

A fronte di quanto descritto precedentemente (sezione 5.2), emerge la necessità di integrare lo schema del database con alcune classi. Questo è dovuto al fatto che sono presenti associazioni molti a molti. In figura 19 queste classi sono rappresentate in colore giallo.

- **Subject\_Group**: rappresenta l’associazione “contain” e ha come attributi le «PK» di *Group* e *Subject*;
- **Dataset\_Protocol**: rappresenta l’associazione “hasA” e ha come attributi le «PK» di *Dataset* e *Protocol*.

La traduzione dello schema concettuale in schema logico, ha portato all’aggiunta, in alcune classi, di alcuni attributi come «FK» che riferiscono ad’altre classi.

- **GroupNameGroup**: «FK» in *Dataset* che rappresenta il gruppo di *Subject<sub>G</sub>* contenuto nel *Dataset<sub>G</sub>*;
- **ClusterAlgorithmID\_Algorithm**: «FK» in *Protocol* che rappresenta l’eventuale algoritmo di clustering<sub>G</sub> associato al *Protocol<sub>G</sub>*;
- **ProtocolNameProtocol**: «FK» in *Feature* che rappresenta il *Protocol<sub>G</sub>* a cui è associata l’istanza della feature<sub>G</sub>;
- **DatasetNameDataset**: «FK» in *Analysis* che rappresenta il *Dataset<sub>G</sub>* su cui viene fatta l’analisi;
- **AnalysisID\_Analysis**: «FK» in *Result* che rappresenta l’analisi che ha prodotto quel risultato;

- **ClusterAlgorithmID\_Algorithm:** «FK» in *Result* che rappresenta l'algoritmo di clustering<sub>G</sub> applicato per ottenere quel risultato, qualora il risultato derivasse dall'algoritmo di clustering<sub>G</sub>;
- **FeatureID\_Feature:** «FK» in *Result* che rappresenta la feature<sub>G</sub> ottenuta, qualora il risultato sia ottenuto applicando la feature extractor<sub>G</sub>;
- **SubjectnameSubject:** «FK» in *Result* che indica a quale Subject<sub>G</sub> fa riferimento il risultato.

## 6 Diagrammi delle attività

Vengono di seguito illustrati i diagrammi di attività che descrivono l'interazione dell'utente, con l'applicativo Romeo. Il diagramma d'uso principale (fig. 20) è stato suddiviso in sotto-diagrammi per ovvi motivi di spazio. I riquadri con sfondo bianco quindi, sono da considerarsi singole azioni, mentre quelli con sfondo azzurro sono attività ad alto livello.

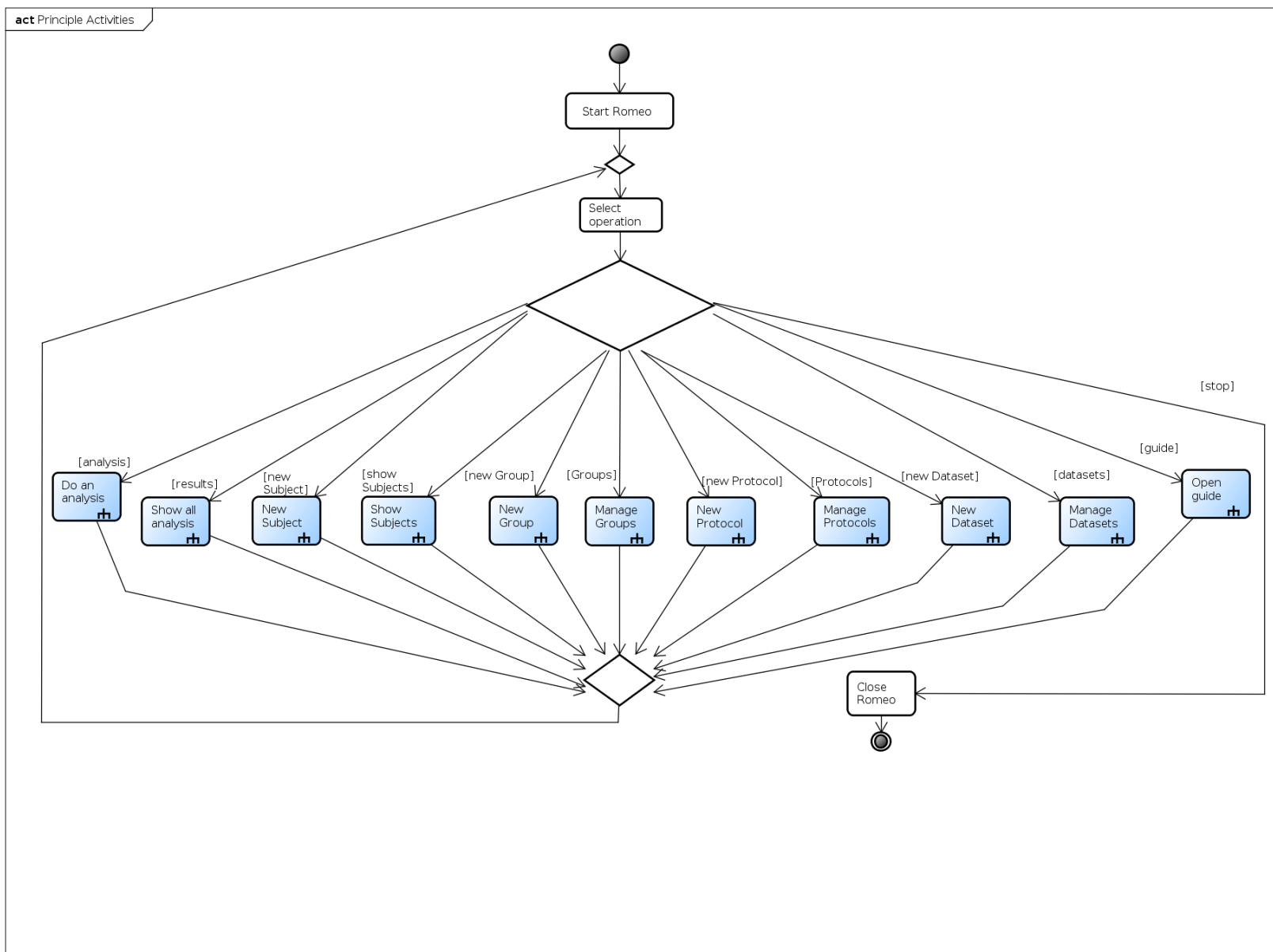
### 6.1 Attività principali

Una volta avviato il programma, l'utente può:

- **Avviare un'analisi;**
- **Creare:** nuovi Subject<sub>G</sub>, nuovi gruppi di Subject<sub>G</sub>, nuovi Protocol<sub>G</sub> e nuovi Dataset<sub>G</sub>;
- **Visualizzare:** i Subject<sub>G</sub>, i gruppi di Subject<sub>G</sub>, i Protocol<sub>G</sub> e i Dataset<sub>G</sub> presenti nel sistema, oltre ai risultati della analisi finora effettuate;
- **Modificare:** i gruppi di Subject<sub>G</sub> (aggiungendo o togliendo uno o più Subject<sub>G</sub>);
- **Eliminare:** Protocol<sub>G</sub> e Dataset<sub>G</sub>;
- **Esportare:** i risultati delle analisi effettuate;
- **Aprire:** la guida contestuale.

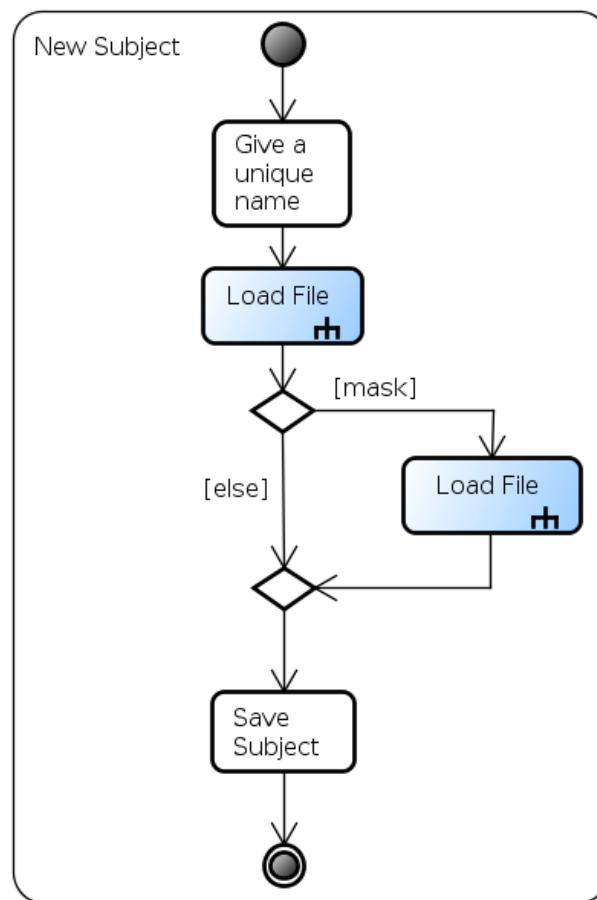
Le funzionalità sopra descritte, potranno essere sfruttate dall'utente in mutua esclusione. Una volta terminata l'azione che l'utente ha deciso di intraprendere, sarà per lui possibile sceglierne un'altra tra quelle proposte oppure chiudere l'applicativo.

Si evidenzia inoltre che, per mantenere una rappresentazione chiara, pulita e fluida delle attività, si è omesso il fatto che l'utente in ogni momento potrà chiudere il programma, accedere ad una voce del menù o ancora, annullare i passi fatti fino a quel momento ritornando alla pagina iniziale.



**Figura 20:** Diagramma Attività - Attività principali dell'applicativo Romeo

## 6.2 Creazione nuovo Subject

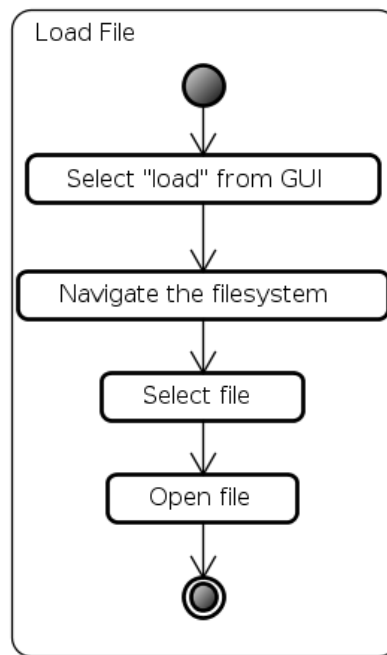


**Figura 21:** Diagramma Attività - Creazione nuovo Subject

### Descrizione

L'attività di creazione di un nuovo Subject<sub>G</sub> (fig. 21), prevede innanzitutto l'assegnazione di un nome univoco al Subject<sub>G</sub> in creazione. Successivamente è necessario caricare il file, che può essere un'immagine o un video, ed eventualmente caricare una sua maschera. Infine, si procede con il salvataggio del Subject<sub>G</sub>.

### 6.2.1 Caricare un File

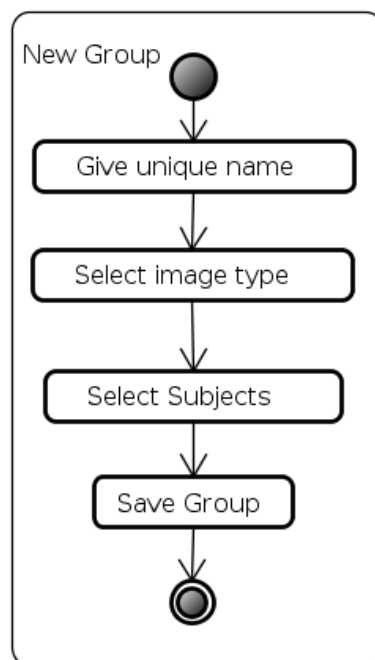


**Figura 22:** Diagramma Attività - Caricamento di un file

#### Descrizione

L'attività di caricamento di un file (fig. 22), prevede la navigazione all'interno del filesystem e la selezione del file che si desidera caricare. Infine, dopo la conferma dell'utente, si procede con l'apertura dello stesso.

### 6.3 Creare nuovo gruppo di Subjects



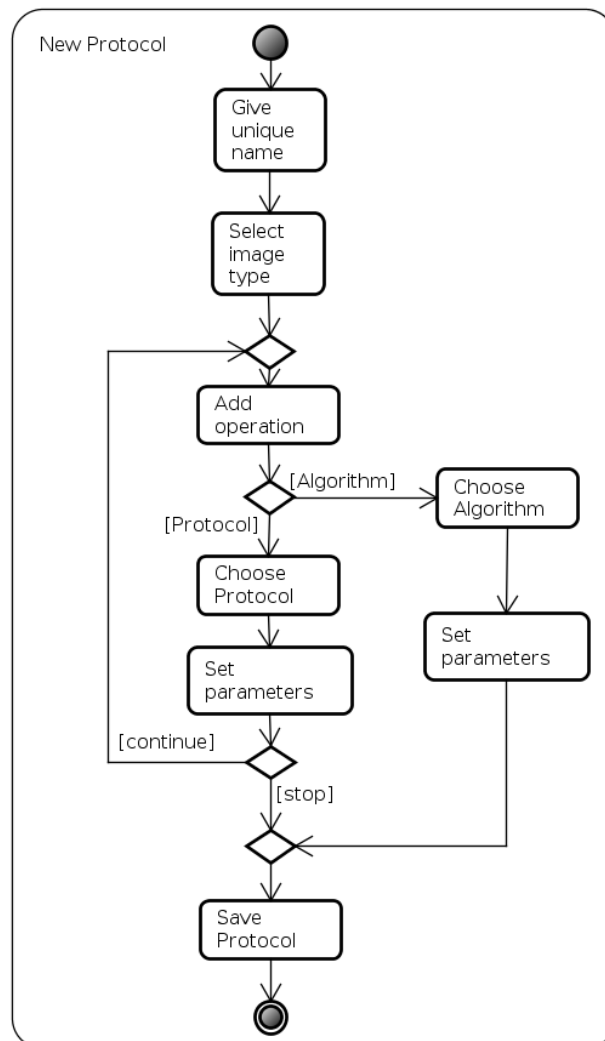
**Figura 23:** Diagramma Attività - Creazione nuovo gruppo di Subject

#### Descrizione

L'attività di creazione di un nuovo gruppo di Subject<sub>G</sub> (fig. 23), prevede in primo luogo l'assegnazione di un nome univoco al gruppo e la scelta del tipo d'immagine (2D, 2D-t, 3D o 3D-t) che si vuole utilizzare. Successivamente è necessario selezionare i Subject<sub>G</sub> da inserire nel gruppo, scegliendo tra quelli che hanno un'immagine associata del tipo precedentemente scelto. Infine si procede con il salvataggio del gruppo.



## 6.4 Creare nuovo Protocol

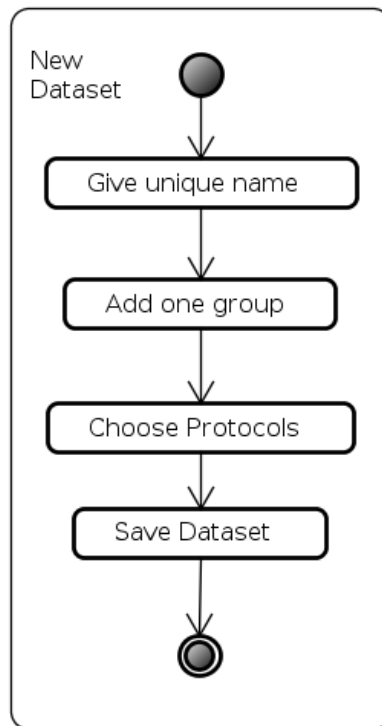


**Figura 24:** Diagramma Attività - Creazione di un nuovo Protocol

### Descrizione

L'attività di creazione di un nuovo Protocol<sub>G</sub> (fig. 24), prevede in primo luogo l'assegnazione di un nome univoco al Protocol<sub>G</sub> e la scelta del tipo di immagine a cui dovrà essere applicato. È possibile poi selezionare le feature extractors<sub>G</sub> che si vogliono utilizzare, dando dei valori ai parametri richiesti, e/o selezionare l'algoritmo di clustering<sub>G</sub> dando anche per esso, dei valori ai parametri richiesti. Qualora non vengano assegnati dei valori, verranno presi quelli di default previsti dal sistema. Una volta terminata la selezione, il Protocol<sub>G</sub> è pronto per essere salvato.

## 6.5 Creare nuovo Dataset

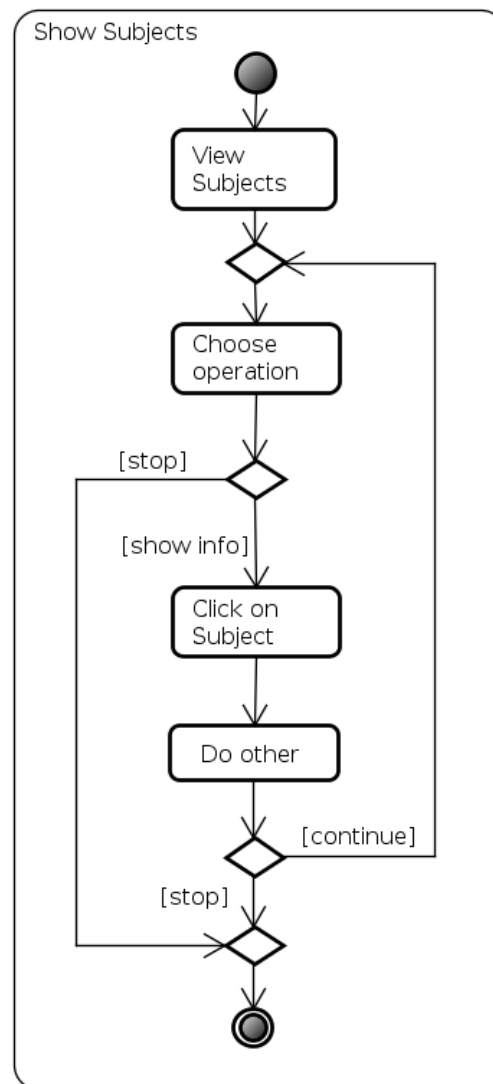


**Figura 25:** Diagramma Attività - Creazione di un nuovo Dataset

### Descrizione

L'attività di creazione di un nuovo Dataset<sub>G</sub> (fig. 25), prevede in primo luogo l'assegnazione di un nome univoco al Dataset<sub>G</sub> in creazione e l'inserimento, in quest'ultimo, di un unico gruppo di Subject<sub>G</sub>. Successivamente, è necessario scegliere uno o più protocol<sub>G</sub> da applicare al gruppo di Subject<sub>G</sub>. I Protocol<sub>G</sub> che potranno essere associati, saranno solo quelli compatibili in base al tipo di immagine del gruppo. Creata quest'associazione, il Dataset<sub>G</sub> è pronto per essere salvato.

## 6.6 Visualizzare i Subject

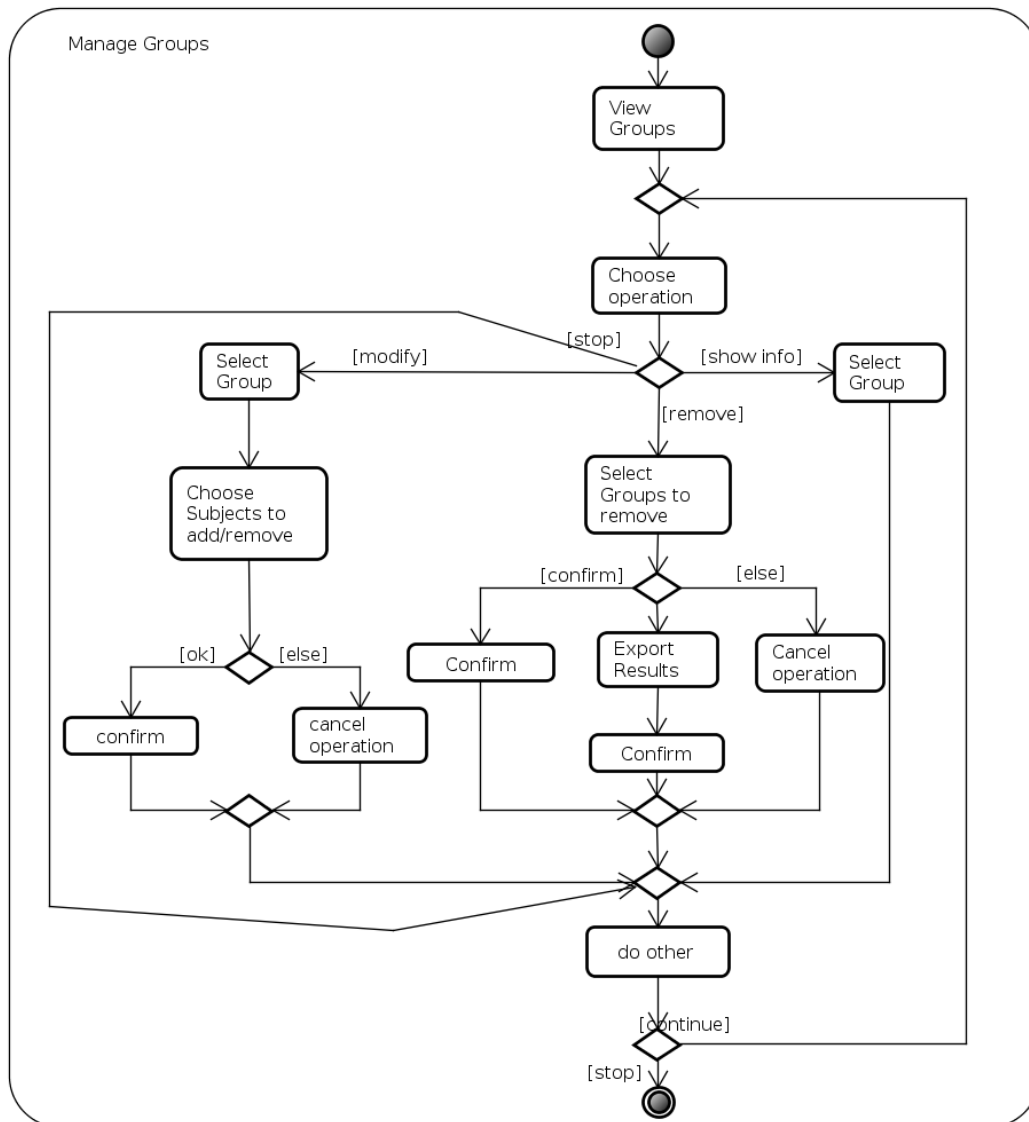


**Figura 26:** Diagramma Attività - Visualizzazione dei Subject

### Descrizione

L'utente avrà a disposizione l'elenco di tutti i `SubjectG` creati fino a quel momento. Selezionandone uno, potrà avere un'anteprima dell'immagine associata, assieme ad alcuni valori di interesse, come per esempio il tipo di immagine e la data di creazione.

## 6.7 Gestire i gruppi di Subject



**Figura 27:** Diagramma Attività - Gestione dei gruppi di Subject

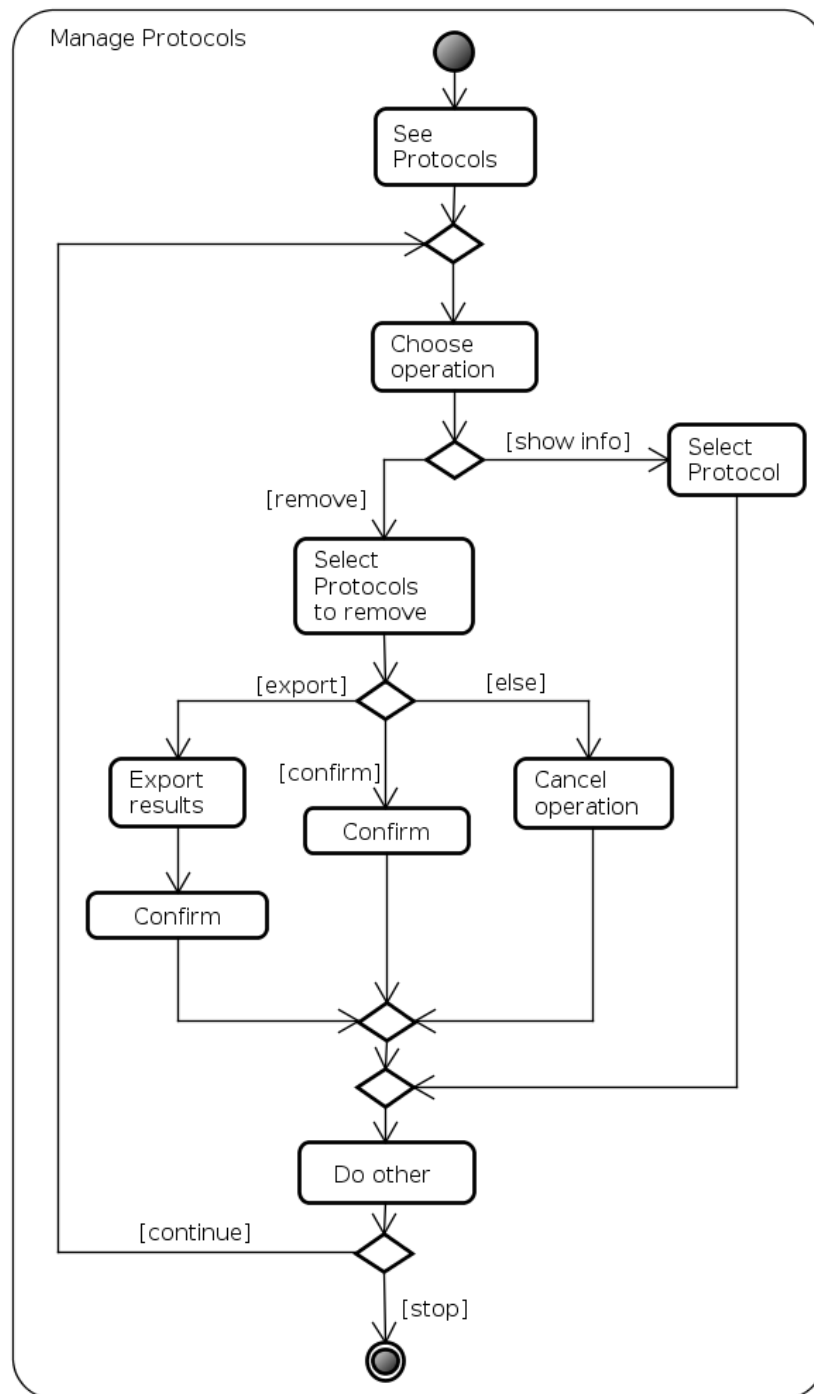
### Descrizione

L'attività di gestione dei gruppi di Subject<sub>G</sub> (fig. 27), dà innanzitutto la possibilità all'utente di visualizzare i gruppi salvati in quel momento nel sistema. Per ogni entità, l'utente può effettuare alcune operazioni citate di seguito.

È possibile rimuovere uno o più gruppi, visualizzarne le informazioni (Subject<sub>G</sub> appartenenti, tipo di immagine, ecc...) e modificarlo. La modifica consiste nel selezionare il gruppo e scegliere quali Subject<sub>G</sub> eliminare o inserire.

Per ogni singola operazione, è necessaria la conferma da parte dell'utente, che può inoltre annullarla in qualsiasi momento.

## 6.8 Gestire i Protocol

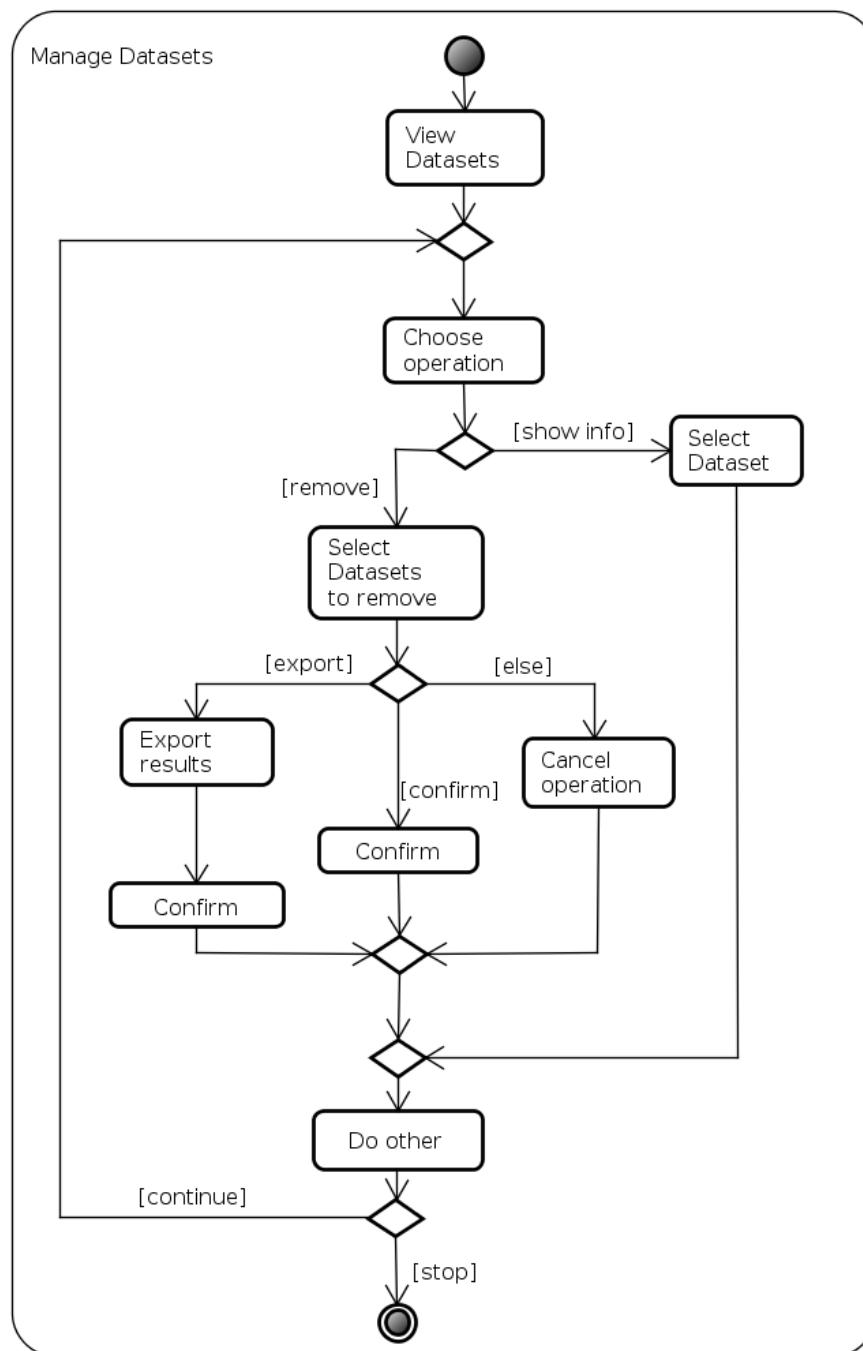


**Figura 28:** Diagramma Attività - Gestione dei Protocol

### Descrizione

L'attività di gestione dei Protocol<sub>G</sub> (fig. 28), dà la possibilità all'utente di visualizzare i Protocol<sub>G</sub> presenti in quel momento nel sistema. Per ogni Protocol<sub>G</sub>, l'utente può decidere se visualizzarne le informazioni d'interesse o se rimuoverlo. La cancellazione di un Protocol<sub>G</sub> necessita della conferma dell'utente.

## 6.9 Gestire i Dataset

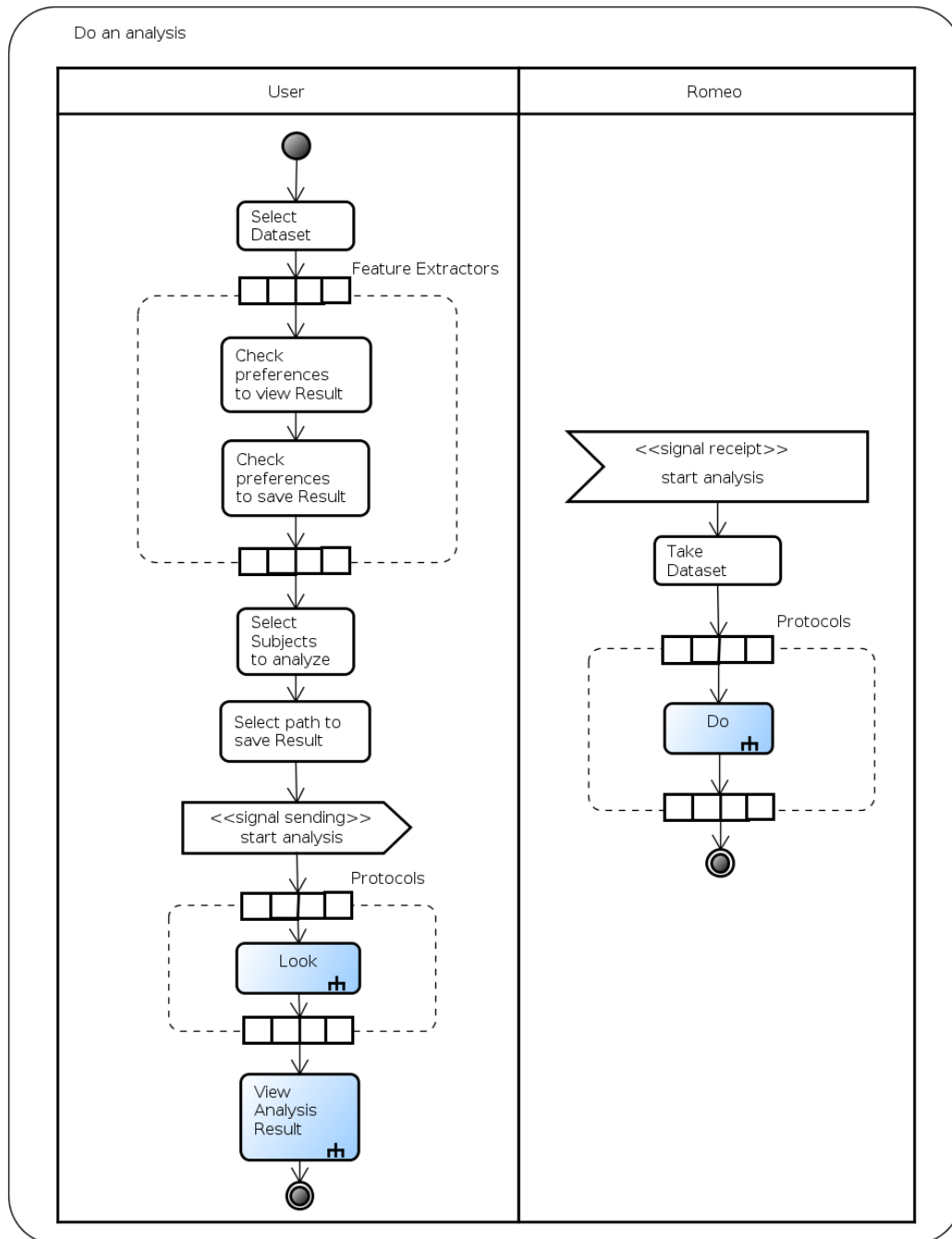


**Figura 29:** Diagramma Attività - Gestione dei Dataset

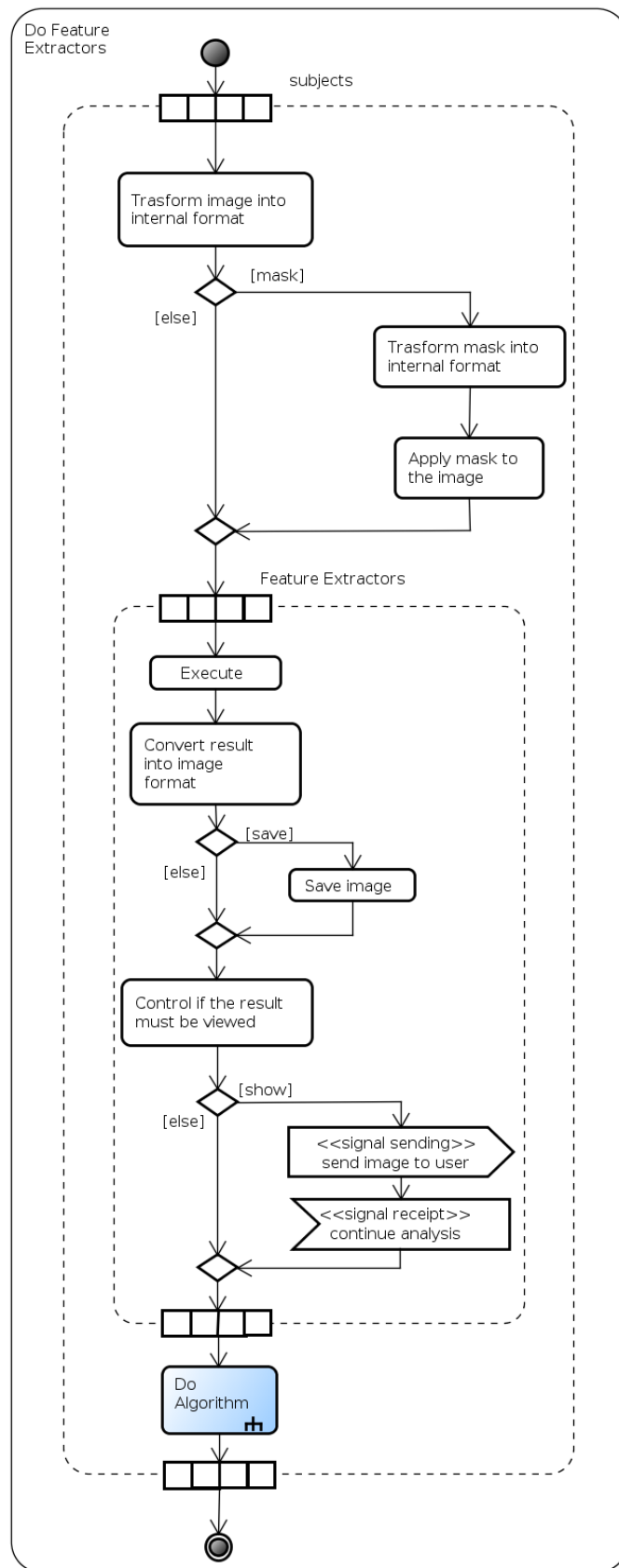
### Descrizione

L'attività di gestione dei Dataset<sub>G</sub> (fig. 29), dà la possibilità all'utente di visualizzare i Dataset<sub>G</sub> presenti in quel momento nel sistema. Per ogni Dataset<sub>G</sub>, è possibile decidere se visualizzarne le informazioni d'interesse o se rimuoverlo. La cancellazione di un Dataset<sub>G</sub> necessita della conferma da parte dell'utente.

## 6.10 Avviare un'analisi

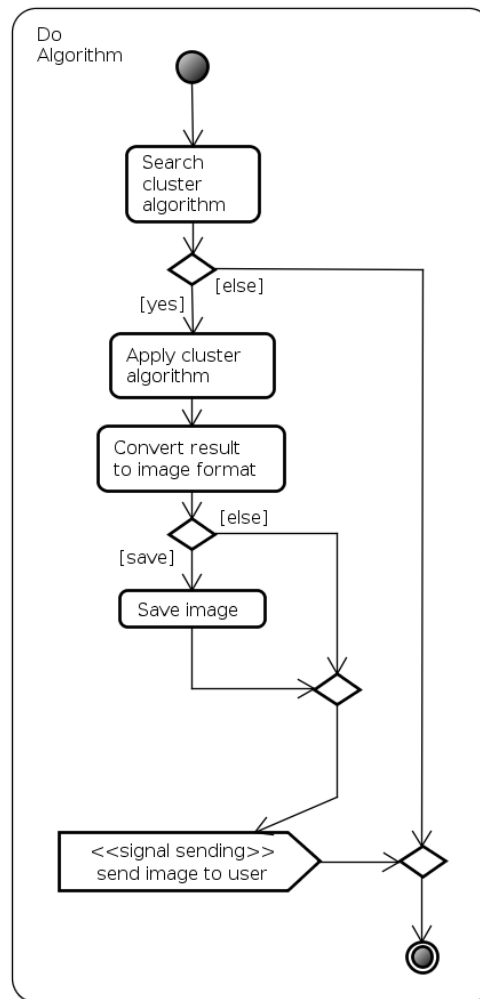


**Figura 30:** Diagramma Attività - Avvio di un'analisi



**Figura 31:** Diagramma Attività - Esecuzione analisi per ogni Protocol





**Figura 32:** Diagramma Attività - Esecuzione analisi per ogni Protocol

### Descrizione

L'attività di esecuzione di un'analisi (fig. 30), è il processo principale del software Romeo. Per questo motivo si è scelto di rappresentare, non solo i passi che l'utente potrà fare, ma anche l'interazione vera e propria con il sistema.

Innanzitutto, l'utente dovrà selezionare il Dataset<sub>G</sub> su cui vuole eseguire l'analisi. Successivamente, per ogni feature extractor<sub>G</sub> presente nei vari Protocol<sub>G</sub> del Dataset<sub>G</sub>, dovrà decidere se visualizzare e se salvare il risultato dell'estrazione, subito dopo il suo calcolo. L'utente dovrà poi selezionare i Subject<sub>G</sub> su cui vorrà effettuare l'analisi, e il path in cui vorrà salvarne i risultati.

Nel momento in cui l'utente fa partire l'esecuzione dell'analisi, il software eseguirà i seguenti passi per ogni Protocol<sub>G</sub> presente nel Dataset<sub>G</sub>:

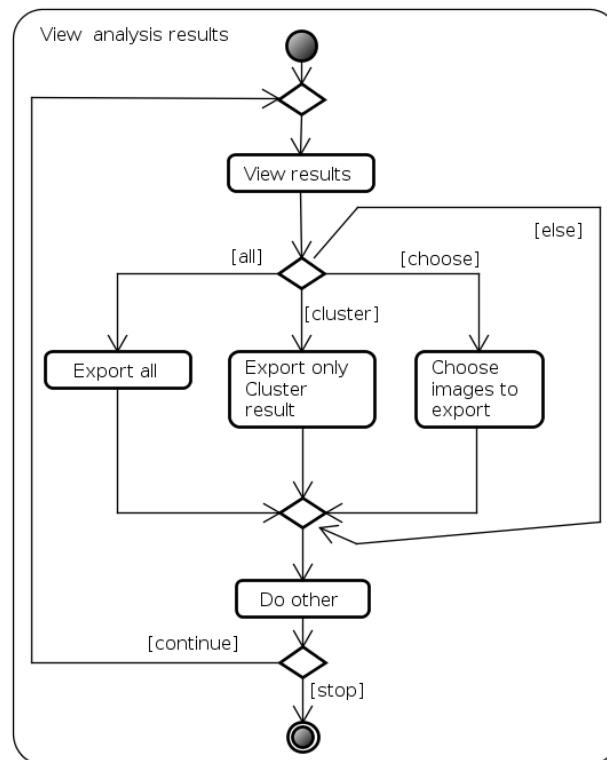
- Trasforma l'immagine e l'eventuale maschera di un Subject<sub>G</sub> nel formato interno al software;
- Applica la maschera all'immagine (se non è presente, questo passo verrà saltato);
- Per ogni feature extractor<sub>G</sub> presente nel Protocol<sub>G</sub>:
  - Esegue l'estrazione della feature<sub>G</sub> dell'immagine;
  - Converte il risultato dell'estrazione in immagine;
  - Salva l'immagine (questo passo viene eseguito se precedentemente scelto dall'utente);

- Mostra l'immagine a video (questo passo viene eseguito se precedentemente scelto dall'utente)(fig. 31);
- Lascia decidere all'utente se continuare a visualizzare i risultati delle features extractors<sub>G</sub> oppure continuare l'estrazione senza più mostrarli;
- Viene eseguito l'algoritmo di clustering<sub>G</sub> eventualmente presente nel Protocol<sub>G</sub>;
- Il risultato verrà riconvertito in immagine;
- L'immagine verrà salvata (condizione precedentemente decisa dall'utente);

I passi sopra elencati, verranno eseguiti per ogni Subject<sub>G</sub> presente nel gruppo di Subject<sub>G</sub> del Dataset<sub>G</sub>.

Una volta terminata l'analisi, l'utente potrà visualizzare tutti risultati dell'analisi (vedi 6.10.1).

### 6.10.1 Visualizzare i risultati dell'analisi

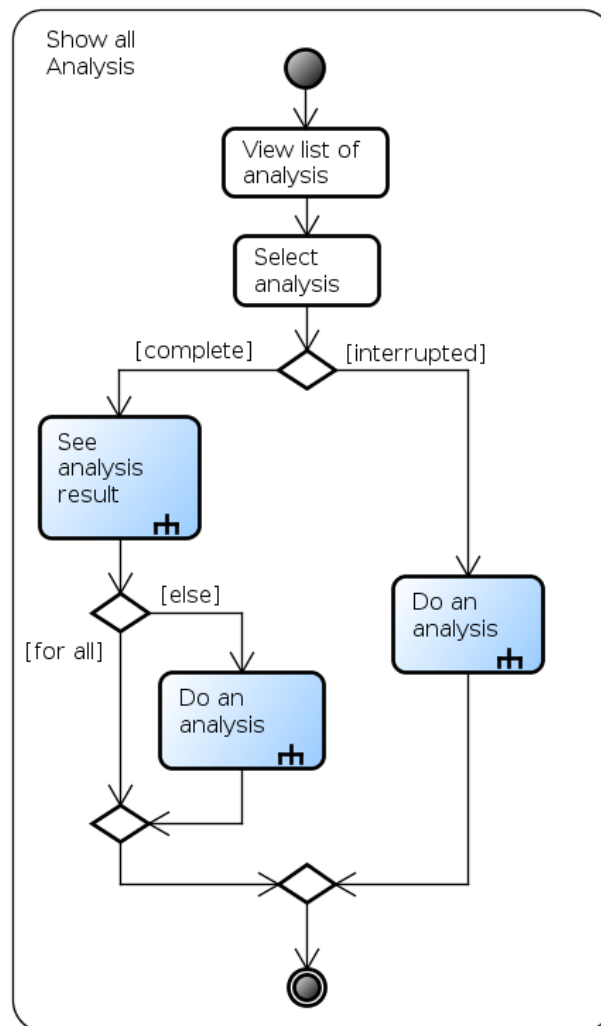


**Figura 33:** Diagramma Attività - Visualizzazione dei risultati dell'analisi effettuata

#### Descrizione

Quest'attività, conseguente alla fine dell'analisi del Dataset<sub>G</sub>, permette all'utente di visualizzarne i risultati. A questo punto, l'utente può decidere se esportarli o meno. Nel caso in cui li voglia esportare, può decidere se esportarli tutti, se esportare solo il risultato ultimo della cluster analysis<sub>G</sub> oppure decidere manualmente quali immagini esportare.

### 6.11 Visualizzare le analisi effettuate



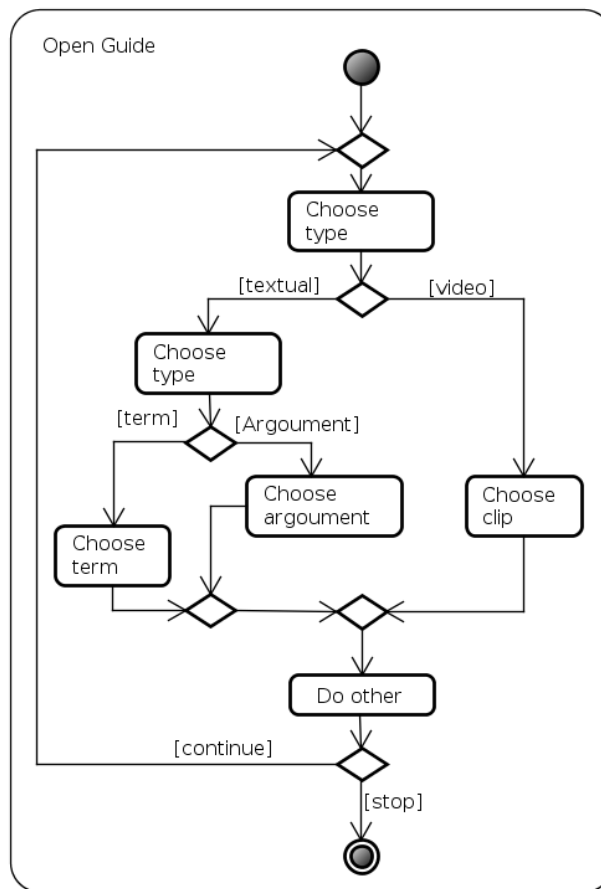
**Figura 34:** Diagramma Attività - Visualizzazione di tutte le analisi effettuate

#### Descrizione

Quest'attività permette all'utente di visualizzare lo storico delle analisi effettuate sui Dataset<sub>G</sub> presenti nel software. Si possono presentare tre casi:

- L'analisi è stata completata su tutti gli elementi del gruppo di Subject<sub>G</sub>. Sarà quindi possibile visualizzarne i risultati;
- L'analisi è stata completata solo per un sottoinsieme di Subject<sub>G</sub> del gruppo. In questo caso, sarà possibile visualizzarne i risultati ed eventualmente decidere di completare l'analisi sul resto degli elementi;
- L'analisi è stata interrotta. In questo caso si ha solo la possibilità di rieseguire l'analisi (fig. 30).

## 6.12 Aprire la guida



**Figura 35:** Diagramma Attività - Apertura della guida

### Descrizione

L'attività di apertura della guida (fig. 6.12), fornisce all'utente la possibilità di essere guidato nell'utilizzo del software Romeo. L'utente può scegliere se aprire la guida testuale oppure la guida video. Per quanto riguarda la guida testuale, ha a disposizione la ricerca per argomento oppure per termine, mentre per quanto riguarda la guida video, potrà scegliere il clip di interesse. Una volta terminata la ricerca, può continuare con una nuova ricerca oppure chiuderla.

## 7 Design pattern

### 7.1 Design pattern architetturali

#### 7.1.1 MVC

**Scopo dell'utilizzo:** viene utilizzato il design pattern MVC per mantenere separati i compiti dei diversi componenti software che interpretano i tre ruoli principali: Model View e Controller.

**Contesto:** il pattern MVC viene utilizzato per l'architettura generale dell'applicazione. Viene utilizzato il sistema Signal e Slot di Qt descritto nella sezione 2.2.1 per far comunicare i componenti tra loro. Ogni modifica fatta sulla View da parte dell'utente, viene inviata al Controller che da un comando al Model. Il Model notifica alla View ogni cambiamento e la View recupera i dati aggiornati del Model.

### 7.2 Design pattern creazionali

#### 7.2.1 Factory

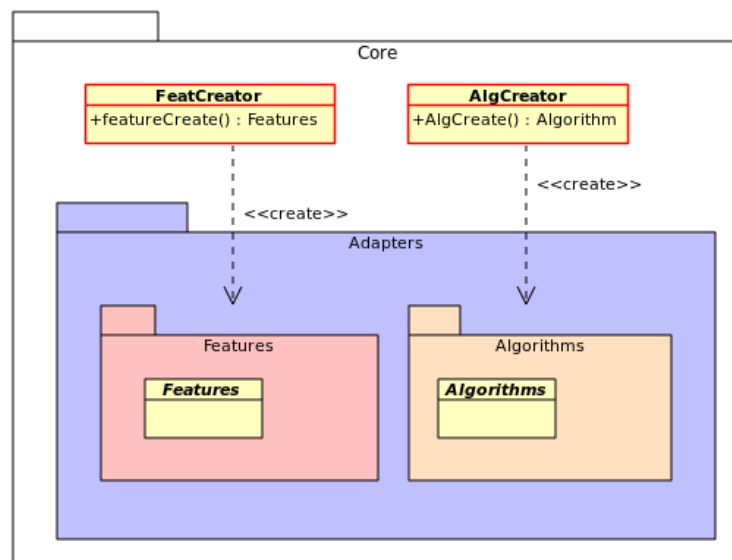


Figura 36: Utilizzo di Factory in Romeo

**Scopo dell'utilizzo:** viene utilizzato il design pattern Factory per delegare alle sotto-classi dell'interfaccia la decisione di quale oggetto creare.

Si noti che per non appesantire troppo la rappresentazione, le Concrete Product sono state omesse, ma sono comunque presenti nella parte che descrive i componenti e le classi.

**Contesto:** le classi che utilizzano tale pattern sono:

- `Romeo::Model::Core::FeatCreator` è factory per tutte le classi (Concrete Product) che ereditano da `Romeo::Model::Core::Adapters::Features` (Product);
- `Romeo::Model::Core::AlgCreator` è factory per tutte le classi (Concrete Product) che ereditano da `Romeo::Model::Core::Adapters::Algorithms` (Product).

### 7.2.2 Singleton

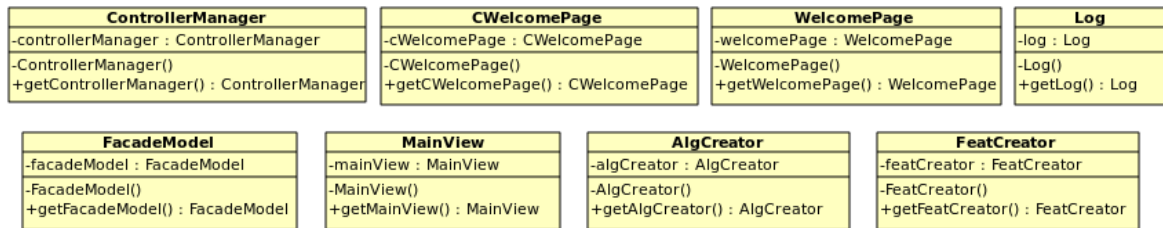


Figura 37: Utilizzo di Singleton in Romeo

**Scopo dell'utilizzo:** viene utilizzato il design pattern Singleton per garantire che durante l'esecuzione dell'applicazione esista una sola istanza della classe in questione.

**Contesto:** le classi che utilizzano tale pattern sono:

- Romeo::Controller::ControllerManager;
- Romeo::Controller::CWelcomePage;
- Romeo::View::WelcomePage;
- Romeo::View::Window::MainView;
- Romeo::Model::FacadeModel;
- Romeo::Model::Core::AlgCreator;
- Romeo::Model::Core::FeatCreator;
- Romeo::Model::Util::Log.

## 7.3 Design pattern strutturali

### 7.3.1 Adapter

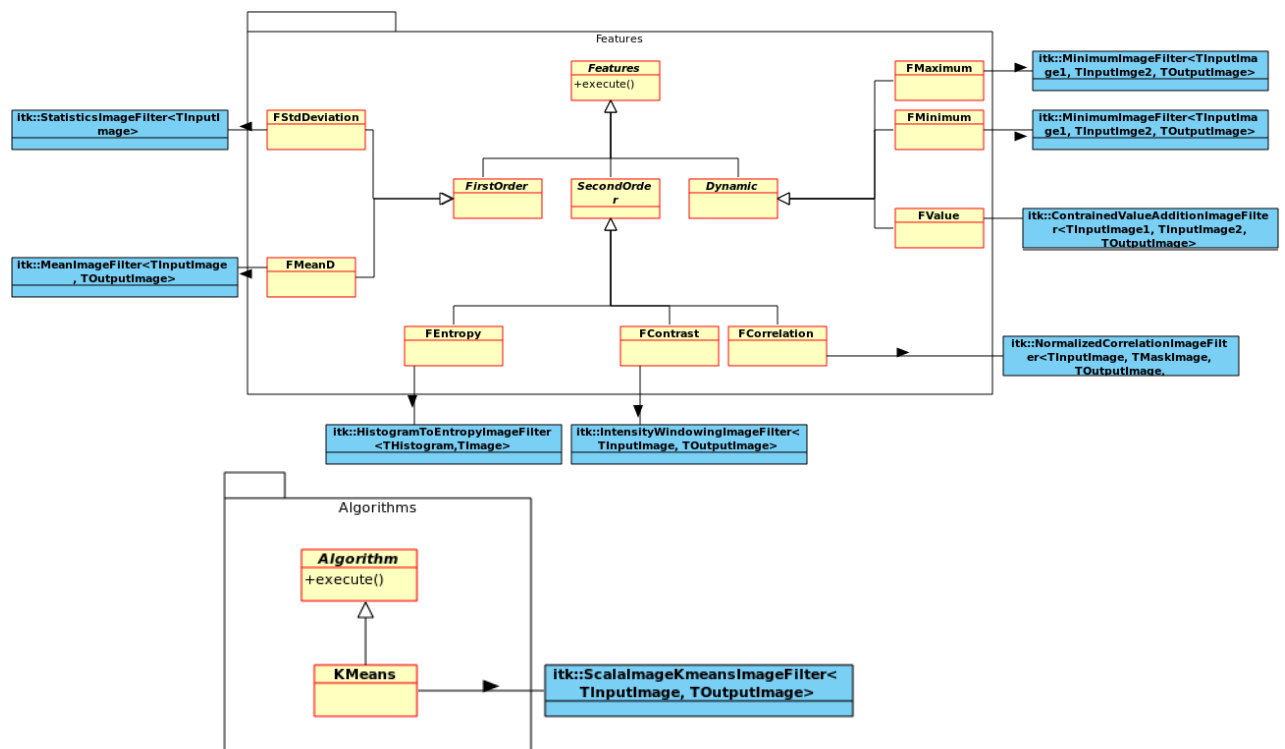


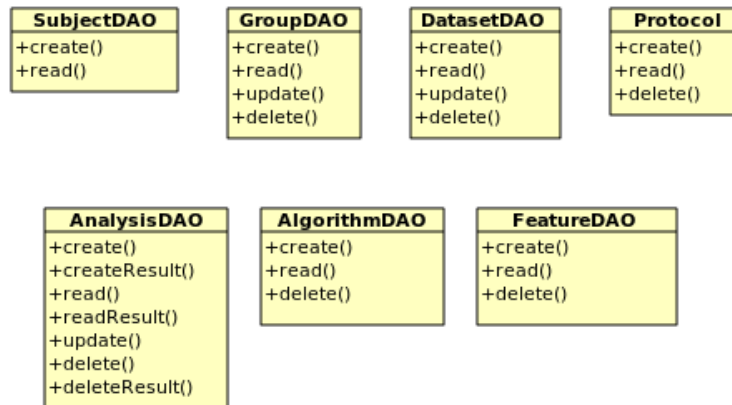
Figura 38: Utilizzo di Adapter in Romeo

**Scopo dell'utilizzo:** viene utilizzato il design pattern Adapter per adattare le interfacce di alcune classi esterne (contenute in ITK<sub>G</sub>) alle nostre classi.

**Contesto:** le classi che utilizzano tale pattern sono:

- `Romeo::Model::Core::Adapters::Features::FStdDeviation` adatta `ITK::StatisticsImageFilter <TInputImage>;`
- `Romeo::Model::Core::Adapters::Features::FMeanD` adatta `ITK::MeanImageFilter<TInputImage, TOutputImage>;`
- `Romeo::Model::Core::Adapters::Features::FEntropy` adatta `ITK::HistogramToEntropyImageFilter<THistogram, TImage>;`
- `Romeo::Model::Core::Adapters::Features::FContrast` adatta `ITK::IntensityWindowingImageFilter<TInputImage, TOutputImage>;`
- `Romeo::Model::Core::Adapters::Features::FCorrelation` adatta `ITK::StatisticsImageFilter<TInputImage, TMaskImage, TOutputImage>;`
- `Romeo::Model::Core::Adapters::Features::FValue` adatta `ITK::ConstrainedValueAdditionImageFilter<TInputImage1, TInputImage2, TOutputImage>;`
- `Romeo::Model::Core::Adapters::Features::FMinimum` adatta `ITK::MinimumImageFilter<TInputImage1, TInputImage2, TOutputImage>;`
- `Romeo::Model::Core::Adapters::Algorithms::KMeans` adatta `ITK::ScalaImageKmeansImageFilter<TInputImage, TOutputImage>;`

### 7.3.2 DAO (Data Access Object)



**Figura 39:** Utilizzo di DAO in Romeo

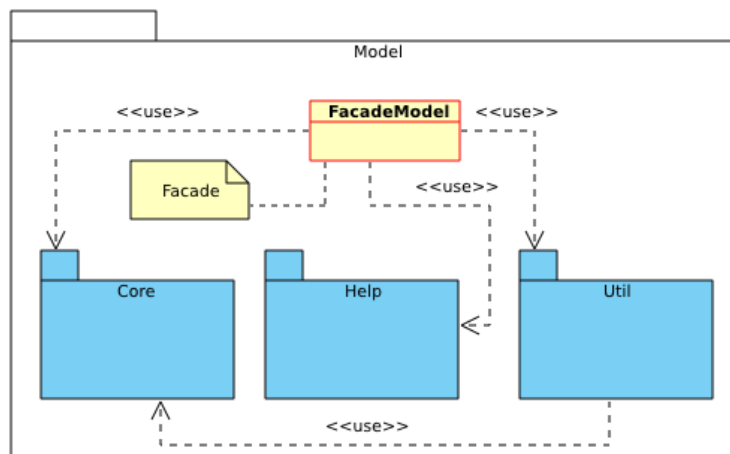
**Scopo dell'utilizzo:** viene utilizzato il design pattern `DAO` per permettere l'interfac-  
ciamento al database.

**Contesto:** le classi che utilizzano tale pattern sono:

- `Romeo::Model::Util::DAO::SubjectDAO;`
- `Romeo::Model::Util::DAO::GroupDAO;`
- `Romeo::Model::Util::DAO::DatasetDAO;`
- `Romeo::Model::Util::DAO::ProtocolDAO;`
- `Romeo::Model::Util::DAO::AlgorithmDAO;`
- `Romeo::Model::Util::DAO::AnalysisDAO;`
- `Romeo::Model::Util::DAO::FeatureDAO.`



### 7.3.3 Facade



**Figura 40:** Utilizzo di Facade in Romeo

**Scopo dell'utilizzo:** viene utilizzato il design pattern **Facade** per permettere, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.

**Contesto:** le classi che utilizzano tale pattern sono:

- `Romeo::Model::FacadeModel`.

## 7.4 Design pattern comportamentali

### 7.4.1 Strategy

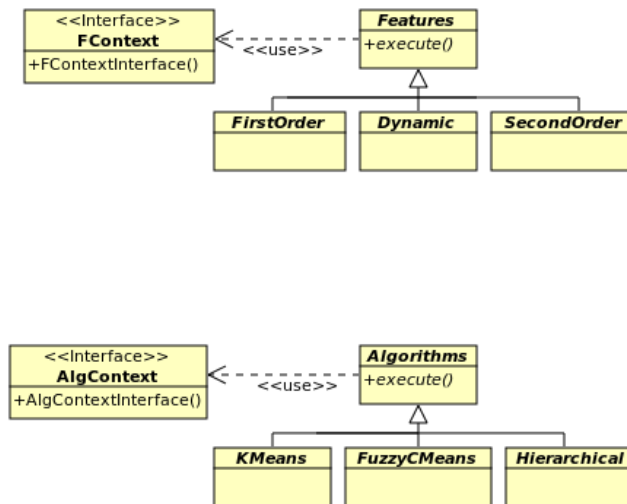


Figura 41: Utilizzo di Strategy in Romeo

**Scopo dell'utilizzo:** viene utilizzato il design pattern<sub>G</sub> Strategy per poter definire raggruppare algoritmi di clustering<sub>G</sub> e feature in base alla tipologia e renderli intercambiabili tra di loro. Si è deciso di utilizzare anzichè, il pattern Strategy puro, una sua variante che consiste nel far sì che Strategy (nel nostro caso, Features e Algorithms) mantenga un riferimento a Context (rispettivamente FContext e AlgContext) anche se si avrebbe un accoppiamento più stretto tra Strategy e Context.

**Contesto:** le classi che utilizzano tale pattern sono:

- `Romeo::Model::Core::Adapters::Features;`
- `Romeo::Model::Core::Adapters::Algorithms.`

## 8 Stime di fattibilità e risorse necessarie

L'architettura definita nella sezione 3 ha un livello di dettaglio sufficiente per poter dare una stima sulla fattibilità e sul bisogno di risorse per poterla realizzare.

- Analizzando l'architettura fino a questo momento progettata, si può confermare che le tecnologie che si adotteranno, risultano essere adeguate per la realizzazione del prodotto e riescono a ricoprire le esigenze progettuali del prodotto;
- Le componenti individuate possono essere assegnate a diversi progettisti, per far sì che vengano definite in modo completo ed esaustivo durante la Definizione del Prodotto. Ci sarà un momento in cui i vari progettisti dovranno essere collaborativi e comunicare tra di loro, ossia quando si dovranno definire le interfacce che permettono alle componenti di interfacciarsi tra loro;
- Gli strumenti necessari sono in gran parte a disposizione da parte dei componenti del gruppo; quelli mancanti sono comunque facilmente reperibili e scaricabili. L'esperienza tecnologica da parte del team *Seven Monkeys* non presenta pesanti lacune, mitigabili, qualora ve ne fossero, con un approfondimento dell'argomento;

Da quando descritto precedentemente si evince che risorse materiali e temporali a disposizione siano sufficienti per poter realizzare, nei tempi preventivati, il prodotto.

## 9 Tracciamento

Seguono le tabelle di tracciamento tra componenti e requisiti. Il componente Controller, essendo un semplice gestore di flusso di controllo tra View e Model, risulta non tracciato da alcun requisito. I componenti Model e Util hanno il solo scopo di contenere altri pacchetti, perciò non sono soggetti a tracciamento.

### 9.1 Tracciamento componenti-requisiti

Componente	Requisito
Romeo	
Romeo::Controller	
Romeo::Model	
Romeo::Model::Core	R0F1 R0F1.2 R0F1.2.1 R0F1.2.1.1 R0F1.2.1.2 R0F1.2.1.3 R0F1.2.1.4 R0F1.2.1.5 R0F1.2.1.6 R0F1.3 R0F1.3.1 R0F1.3.2 R0F1.3.3 R0F1.3.4 R0F1.3.5 R0F1.4 R0F10 R0F10.1 R0F10.1.1 R0F10.2 R0F10.2.1 R0F10.3 R0F12.3 R0F26 R0F26.1 R0F26.2 R0F27 R0F27.1 R0F27.2 R0F3 R0F3.1 R0F4 R0F5 R0F5.1 R0F5.2 R0F5.3 R0F5.4 R0F6 R0F8 R0F8.2

	R0F8.3 R2F3.2
Romeo::Model::Core::Adapters	
Romeo::Model::Core::Adapters::Algorithms	R0F5.4.1 R0F5.4.1.1 R0F5.4.1.1.1 R0F5.4.1.2 R0F5.4.1.2.1 R0F5.4.1.3 R0F5.4.1.3.1 R0F5.4.1.4 R0F5.4.1.4.1 R0F5.4.2 R0F5.4.2.1 R0F5.4.2.1.1 R0F5.4.2.2 R0F5.4.2.2.1 R0F5.4.2.3 R0F5.4.2.3.1 R0F5.4.2.4 R0F5.4.2.4.1 R0F5.4.3 R0F5.4.3.1 R0F5.4.3.1.1 R0F5.4.3.2 R0F5.4.3.2.1
Romeo::Model::Core::Adapters::Features	R0F5.2.1 R0F5.2.1.1 R0F5.2.1.1.1 R0F5.2.1.1.2 R0F5.2.10 R0F5.2.10.1 R0F5.2.10.1.1 R0F5.2.10.2 R0F5.2.10.2.1 R0F5.2.11 R0F5.2.11.1 R0F5.2.11.1.1 R0F5.2.11.2 R0F5.2.11.2.1 R0F5.2.12 R0F5.2.12.1 R0F5.2.12.1.1 R0F5.2.12.2 R0F5.2.12.2.1 R0F5.2.13 R0F5.2.13.1 R0F5.2.13.1.1 R0F5.2.13.2 R0F5.2.13.2.1 R0F5.2.14 R0F5.2.14.1 R0F5.2.14.1.1

	R0F5.2.14.2
	R0F5.2.14.2.1
	R0F5.2.15
	R0F5.2.15.1
	R0F5.2.15.1.1
	R0F5.2.15.2
	R0F5.2.15.2.1
	R0F5.2.2
	R0F5.2.2.1
	R0F5.2.2.1.1
	R0F5.2.2.1.2
	R0F5.2.3
	R0F5.2.3.1
	R0F5.2.3.1.1
	R0F5.2.3.1.2
	R0F5.2.4
	R0F5.2.4.1
	R0F5.2.4.1.1
	R0F5.2.4.1.2
	R0F5.2.5
	R0F5.2.5.1
	R0F5.2.5.1.1
	R0F5.2.5.1.2
	R0F5.2.5.2
	R0F5.2.5.2.1
	R0F5.2.6
	R0F5.2.6.1
	R0F5.2.6.1.1
	R0F5.2.6.1.2
	R0F5.2.6.2
	R0F5.2.6.2.1
	R0F5.2.7
	R0F5.2.7.1
	R0F5.2.7.1.1
	R0F5.2.7.1.2
	R0F5.2.7.2
	R0F5.2.7.2.1
	R0F5.2.8
	R0F5.2.8.1
	R0F5.2.8.1.1
	R0F5.2.8.1.2
	R0F5.2.8.2
	R0F5.2.8.2.1
	R0F5.2.9
	R0F5.2.9.1
	R0F5.2.9.1.1
	R0F5.2.9.1.2
	R0F5.2.9.2
	R0F5.2.9.2.1
	R0V7

Romeo::Model::Help	R0F14 R0F14.1 R2F14.1.1 R2F14.1.2 R2F14.2 R2F14.2.1
Romeo::Model::Util	
Romeo::Model::Util::DAO	R0F1 R0F1.1 R0F11 R0F13 R0F26 R0F26.1 R0F26.2 R0F27 R0F27.1 R0F27.2 R0F3 R0F3.1 R0F4 R0F4.2 R0F4.3 R0F5 R0F5.1 R0F5.2 R0F5.3 R0F5.4 R0F6 R0F8 R0F8.1 R0F8.2 R0F8.3
Romeo::Model::Util::ExporterModel	R0F10.2 R0F10.2.1 R0F12 R0F12.1 R0F12.2 R0F12.3 R0F12.4 R0F13.1 R0F13.2 R0F4.1
Romeo::Model::Util::Log	
Romeo::View	R0F1 R0F9
Romeo::View::Component	
Romeo::View::Dialog	R0F10 R0F10.2 R0F10.3 R0F12.4

Romeo::View::Window	R0F1.1 R0F1.2 R0F1.3 R0F10.4 R0F10.5 R0F13 R0F13.1 R0F13.2 R0F26 R0F26.1 R0F26.2 R0F27 R0F27.1 R0F27.2 R0F3 R0F3.1 R0F4 R0F4.2 R0F4.3 R0F5 R0F5.1 R0F6 R0F6.1 R0F8 R0F8.1 R2F5.5
---------------------	---

**Tabella 3:** Tracciamento componenti-requisiti

## 9.2 Tracciamento requisiti-componenti

Requisito	Descrizione	Componente
R0F1	L'utente può creare un Subject	Core DAO View
R0F1.1	L'utente deve poter dare un nome univoco al Subject	DAO Window
R0F1.2	L'utente deve poter caricare un immagine 2D, 3D o video per ogni Subject	Core Window
R0F1.2.1	L'utente deve poter caricare come immagine di ogni Subject file di formato diverso	Core
R0F1.2.1.1	Il sistema deve accettare in input file di formato PNG <sub>G</sub>	Core
R0F1.2.1.2	Il sistema deve accettare in input file di formato JPG <sub>G</sub>	Core
R0F1.2.1.3	Il sistema deve accettare in input file di formato BMP <sub>G</sub>	Core
R0F1.2.1.4	Il sistema deve accettare in input file di formato AVI <sub>G</sub>	Core
R0F1.2.1.5	Il sistema deve accettare in input file di formato NIfTI <sub>G</sub>	Core



R0F1.2.1.6	Il sistema deve accettare in input file di formato Analyze7.5 <sub>G</sub>	Core
R0F1.3	L'utente deve poter caricare un'immagine maschera per ogni Subject	Core Window
R0F1.3.1	L'utente può caricare un file di formato PNG <sub>G</sub> come maschera di un immagine 2D o 2D time dependent	Core
R0F1.3.2	L'utente può caricare un file di formato JPG come maschera di un immagine 2D o 2D time dependent	Core
R0F1.3.3	L'utente può caricare un file di formato BMP come maschera di un immagine 2D o 2D time dependent	Core
R0F1.3.4	L'utente può caricare un file di formato NIfTI <sub>G</sub> come maschera di un immagine 3D o 3D time dependent	Core
R0F1.3.5	L'utente può caricare un file di formato Analyze <sub>G</sub> come maschera di un immagine 3D o 3D time dependent	Core
R0F1.4	Il software deve bloccare e notificare un tentativo di caricamento di file con formato non consentito	Core
R0F10	Il software deve analizzare le immagini ricevute in input	Core Dialog
R0F10.1	Il software deve terminare l'analisi relativa ad un Subject prima di iniziarne una relativa ad un altro	Core
R0F10.1.1	Il software, per ogni Subject, deve prima calcolare tutte le feature ed eventualmente poi applicare l'algoritmo di clustering	Core
R0F10.2	Il software deve poter interrompere l'analisi per permettere all'utente di visionare i risultati delle immagini appena processate	Core Dialog ExporterModel
R0F10.2.1	Il software deve mostrare il risultato appena pronto	Core ExporterModel
R0F10.3	Il software deve permettere all'utente di interrompere l'analisi in corso	Core Dialog
R0F10.4	Il software deve dare la possibilità all'utente di visualizzare i risultati al termine dell'analisi	Window
R0F10.5	Il software deve fornire una barra di avanzamento che rispecchi il progresso dell'analisi in corso	Window
R0F11	L'utente deve poter salvare i Protocol creati	DAO
R0F12	L'utente deve poter esportare i risultati delle analisi effettuate	ExporterModel
R0F12.1	L'utente deve poter esportare anche i risultati di ogni singola feature	ExporterModel
R0F12.2	L'utente deve poter esportare i risultati con lo stesso formato dei file di input	ExporterModel
R0F12.3	Il software deve salvare i risultati dell'analisi ogni qualvolta termini l'analisi di un singolo Subject	Core ExporterModel

R0F12.4	L'utente deve poter indicare dove salvare i risultati delle analisi di ogni gruppo di Subject	Dialog ExporterModel
R0F13	L'utente deve poter visualizzare i risultati delle analisi effettuate	DAO Window
R0F13.1	Il software deve permettere la visualizzazione di immagini 2D	ExporterModel Window
R0F13.2	Il software deve permettere la visualizzazione di immagini 3D	ExporterModel Window
R0F14	Il software deve fornire una guida	Help
R0F14.1	La guida all'interno del software deve essere in formato testuale	Help
R0F26	L'utente deve poter modificare i gruppi di Subject	Core DAO Window
R0F26.1	L'utente deve poter aggiungere Subject ad un gruppo già esistente	Core DAO Window
R0F26.2	L'utente deve poter rimuovere dei Subject da un gruppo già esistente	Core DAO Window
R0F27	L'utente deve poter eliminare i Dataset	Core DAO Window
R0F27.1	L'utente deve poter eliminare un singolo Dataset	Core DAO Window
R0F27.2	L'utente deve poter eliminare più di un Dataset alla volta	Core DAO Window
R0F3	L'utente può creare gruppi di Subject	Core DAO Window
R0F3.1	L'utente deve dare ai gruppi di Subject un nome univoco	Core DAO Window
R0F4	L'utente può eliminare gruppi di Subject	Core DAO Window
R0F4.1	L'utente deve poter scegliere di esportare i risultati prima dell'eliminazione del gruppo di Subject	ExporterModel
R0F4.2	L'utente deve poter eliminare un solo gruppo di Subject	DAO Window

R0F4.3	L'utente deve poter eliminare più gruppi di Subject alla volta	DAO Window
R0F5	Il software deve permettere la creazione di Protocol	Core DAO Window
R0F5.1	L'utente deve poter dare un nome univoco al Protocol	Core DAO Window
R0F5.2	I Protocol possono contenere una o più feature extractors	Core DAO
R0F5.2.1	Il software deve saper calcolare la feature <u>Mean</u>	Features
R0F5.2.1.1	L'utente deve poter inserire la window size per <u>Mean</u>	Features
R0F5.2.1.1.1	Il valore di default di window size della feature <u>Mean</u> per immagini 2D è 3x3	Features
R0F5.2.1.1.2	Il valore di default di window size della feature <u>Mean</u> per immagini 3D è 3x3x3	Features
R0F5.2.10	Il software deve saper calcolare la feature <u>G Time to Peak</u>	Features
R0F5.2.10.1	L'utente deve poter inserire il frame d'inizio per <u>Time to Peak</u>	Features
R0F5.2.10.1.1	Il valore di default del frame d'inizio per <u>Time to Peak</u> è 1	Features
R0F5.2.10.2	L'utente deve poter inserire il frame di fine per <u>Time to Peak</u>	Features
R0F5.2.10.2.1	Il valore di default del frame di fine per <u>Time to Peak</u> è l'ultimo frame del video inserito	Features
R0F5.2.11	Il software deve saper calcolare la feature <u>G Maximum</u>	Features
R0F5.2.11.1	L'utente deve poter inserire il frame d'inizio per <u>Maximum</u>	Features
R0F5.2.11.1.1	Il valore di default del frame d'inizio per <u>Maximum</u> è 1	Features
R0F5.2.11.2	L'utente deve poter inserire il frame di fine per <u>Maximum</u>	Features
R0F5.2.11.2.1	Il valore di default del frame di fine per <u>Maximum</u> è l'ultimo frame del video inserito	Features
R0F5.2.12	Il software deve saper calcolare la feature <u>G Minimum</u>	Features
R0F5.2.12.1	L'utente deve poter inserire il frame d'inizio per <u>Minimum</u>	Features
R0F5.2.12.1.1	Il valore di default del frame d'inizio per <u>Minimum</u> è 1	Features
R0F5.2.12.2	L'utente deve poter inserire il frame di fine per <u>Minimum</u>	Features
R0F5.2.12.2.1	Il valore di default del frame di fine per <u>Minimum</u> è l'ultimo frame del video inserito	Features
R0F5.2.13	Il software deve saper calcolare la feature <u>G Slope</u>	Features
R0F5.2.13.1	L'utente deve poter inserire il frame d'inizio per <u>Slope</u>	Features
R0F5.2.13.1.1	Il valore di default del frame d'inizio per <u>Slope</u> è 1	Features
R0F5.2.13.2	L'utente deve poter inserire il frame di fine per <u>Slope</u>	Features
R0F5.2.13.2.1	Il valore di default del frame di fine per <u>Slope</u> è l'ultimo frame del video inserito	Features

R0F5.2.14	Il software deve saper calcolare la feature $\mathbf{G}$ Mean	Features
R0F5.2.14.1	L'utente deve poter inserire il frame d'inizio per Mean	Features
R0F5.2.14.1.1	Il valore di default del frame d'inizio per Mean è 1	Features
R0F5.2.14.2	L'utente deve poter inserire il frame di fine per Mean	Features
R0F5.2.14.2.1	Il valore di default del frame di fine per Mean è l'ultimo frame del video inserito	Features
R0F5.2.15	Il software deve saper calcolare la feature $\mathbf{G}$ Value	Features
R0F5.2.15.1	L'utente deve poter inserire il frame d'inizio per Value	Features
R0F5.2.15.1.1	Il valore di default del frame d'inizio per Value è 1	Features
R0F5.2.15.2	L'utente deve poter inserire il frame di fine per Value	Features
R0F5.2.15.2.1	Il valore di default del frame di fine per Value è l'ultimo frame del video inserito	Features
R0F5.2.2	Il software deve saper calcolare la feature $\mathbf{G}$ Standard deviation	Features
R0F5.2.2.1	L'utente deve poter inserire la window size per Standard deviation	Features
R0F5.2.2.1.1	Il valore di default di window size della feature Standard deviation per immagini 2D è 3x3	Features
R0F5.2.2.1.2	Il valore di default di window size della feature Standard deviation per immagini 3D è 3x3x3	Features
R0F5.2.3	Il software deve saper calcolare la feature Skewness	Features
R0F5.2.3.1	L'utente deve poter inserire la window size per Skewness	Features
R0F5.2.3.1.1	Il valore di default di window size della feature Skewness per immagini 2D è 3x3	Features
R0F5.2.3.1.2	Il valore di default di window size della feature Skewness per immagini 3D è 3x3x3	Features
R0F5.2.4	Il software deve saper calcolare la feature Kurtosis	Features
R0F5.2.4.1	L'utente deve poter inserire la window size per Kurtosis	Features
R0F5.2.4.1.1	Il valore di default di window size della feature Kurtosis per immagini 2D è 3x3	Features
R0F5.2.4.1.2	Il valore di default di window size della feature Kurtosis per immagini 3D è 3x3x3	Features
R0F5.2.5	Il software deve saper calcolare la feature Contrast	Features
R0F5.2.5.1	L'utente deve poter inserire la window size per Contrast	Features
R0F5.2.5.1.1	Il valore di default di window size della feature Contrast per immagini 2D è 3x3	Features
R0F5.2.5.1.2	Il valore di default di window size della feature Contrast per immagini 3D è 3x3x3	Features
R0F5.2.5.2	L'utente deve poter inserire la distanza della GLCM per Contrast	Features
R0F5.2.5.2.1	Il valore di default per la distanza della GLCM per Contrast è 1	Features
R0F5.2.6	Il software deve saper calcolare la feature Homogeneity	Features
R0F5.2.6.1	L'utente deve poter inserire la window size per Homogeneity	Features

R0F5.2.6.1.1	Il valore di default di window size della feature Homogeneity per immagini 2D è 3x3	Features
R0F5.2.6.1.2	Il valore di default di window size della feature Homogeneity per immagini 3D è 3x3x3	Features
R0F5.2.6.2	L'utente deve poter inserire la distanza della GLCM per Homogeneity	Features
R0F5.2.6.2.1	Il valore di default per la distanza della GLCM per Homogeneity è 1	Features
R0F5.2.7	Il software deve saper calcolare la feature Entropy	Features
R0F5.2.7.1	L'utente deve poter inserire la window size per Entropy	Features
R0F5.2.7.1.1	Il valore di default di window size della feature Entropy per immagini 2D è 3x3	Features
R0F5.2.7.1.2	Il valore di default di window size della feature Entropy per immagini 3D è 3x3x3	Features
R0F5.2.7.2	L'utente deve poter inserire la distanza della GLCM per Entropy	Features
R0F5.2.7.2.1	Il valore di default per la distanza della GLCM per Entropy è 1	Features
R0F5.2.8	Il software deve saper calcolare la feature Energy	Features
R0F5.2.8.1	L'utente deve poter inserire la window size per Energy	Features
R0F5.2.8.1.1	Il valore di default di window size della feature Energy per immagini 2D è 3x3	Features
R0F5.2.8.1.2	Il valore di default di window size della feature Energy per immagini 3D è 3x3x3	Features
R0F5.2.8.2	L'utente deve poter inserire la distanza della GLCM per Energy	Features
R0F5.2.8.2.1	Il valore di default per la distanza della GLCM per Energy è 1	Features
R0F5.2.9	Il software deve saper calcolare la feature Correlation	Features
R0F5.2.9.1	L'utente deve poter inserire la window size per Correlation	Features
R0F5.2.9.1.1	Il valore di default di window size della feature Correlation per immagini 2D è 3x3	Features
R0F5.2.9.1.2	Il valore di default di window size della feature Correlation per immagini 3D è 3x3x3	Features
R0F5.2.9.2	L'utente deve poter inserire la distanza della GLCM per Correlation	Features
R0F5.2.9.2.1	Il valore di default per la distanza della GLCM per Correlation è 1	Features
R0F5.3	Un Protocol può contenere due istanze di una stessa feature extractor ma con parametri diversi	Core DAO
R0F5.4	Ogni Protocol deve contenere al massimo un algoritmo di clustering	Core DAO
R0F5.4.1	Il software deve saper applicare l'algoritmo di clustering K-means	Algorithms
R0F5.4.1.1	L'utente deve poter inserire il numero di cluster per K-means	Algorithms
R0F5.4.1.1.1	Il valore di default per il numero di clusters di K-means è 10	Algorithms
R0F5.4.1.2	L'utente deve poter inserire il numero di repliche per K-means	Algorithms

R0F5.4.1.2.1	Il valore di default per il numero di repliche di K-means <sub>G</sub> è 5	Algorithms
R0F5.4.1.3	L'utente deve poter inserire il massimo numero di iterazioni per K-means	Algorithms
R0F5.4.1.3.1	Il valore di default per il massimo numero di iterazioni di K-means <sub>G</sub> è 200	Algorithms
R0F5.4.1.4	L'utente deve poter inserire il tipo di distanza per K-means	Algorithms
R0F5.4.1.4.1	Il valore di default per il tipo di distanza di K-means <sub>G</sub> è euclidea	Algorithms
R0F5.4.2	Il software deve saper applicare l'algoritmo di clustering Fuzzy C	Algorithms
R0F5.4.2.1	L'utente deve poter inserire il numero di cluster per Fuzzy C	Algorithms
R0F5.4.2.1.1	Il valore di default per il numero di clusters di Fuzzy C <sub>G</sub> è 10	Algorithms
R0F5.4.2.2	L'utente deve poter inserire il massimo numero di iterazioni per Fuzzy C	Algorithms
R0F5.4.2.2.1	Il valore di default per il massimo numero di iterazioni di Fuzzy C è 200	Algorithms
R0F5.4.2.3	L'utente deve poter inserire il Fuzzy index	Algorithms
R0F5.4.2.3.1	Il valore di default per il fuzzy index di Fuzzy C è 2.0	Algorithms
R0F5.4.2.4	L'utente deve poter inserire la soglia di probabilità per Fuzzy C <sub>G</sub>	Algorithms
R0F5.4.2.4.1	Il valore di default per la soglia di probabilità di Fuzzy C <sub>G</sub> è 1e-3	Algorithms
R0F5.4.3	Il software deve saper applicare l'algoritmo di clustering Hierarchical	Algorithms
R0F5.4.3.1	L'utente deve poter inserire il criterio di collegamento per Hierarchical	Algorithms
R0F5.4.3.1.1	Il valore di default per il criterio di collegamento di Hierarchical <sub>G</sub> è single linkage	Algorithms
R0F5.4.3.2	L'utente deve poter inserire il tipo di distanza per Hierarchical	Algorithms
R0F5.4.3.2.1	Il valore di default per il tipo di distanza di Hierarchical è euclidea	Algorithms
R0F6	L'utente può eliminare un Protocol	Core DAO Window
R0F6.1	L'utente può eliminare più di un Protocol alla volta	Window
R0F8	L'utente può creare un Dataset	Core DAO Window
R0F8.1	L'utente deve poter dare un nome univoco al Dataset	DAO Window
R0F8.2	L'utente può inserire uno o più Protocol nel Dataset	Core DAO
R0F8.3	L'utente può inserire un gruppo di Subject nel Dataset	Core DAO
R0F9	Il software deve avere una GUI	View

R0V7	L'architettura del software deve permettere,in futuro,l'aggiunta di nuove feature a livello di codice	Features
R2F14.1.1	La guida deve essere dotata di un piccolo motore di ricerca che permetta all'utente di cercare alcuni termini all'interno della guida stessa	Help
R2F14.1.2	La guida deve contenere le informazioni suddivise per argomenti per facilitarne la consultazione	Help
R2F14.2	Il software deve fornire una video-guida	Help
R2F14.2.1	La video-guida deve mostrare in maniera veloce ma completa,come utilizzare il software	Help
R2F3.2	Per ogni gruppo di Subject,l'utente può inserire più Subject aventi immagini di formato diverso ma dello stesso tipo(2D,2D-t,3D,3D-t)	Core
R2F5.5	L'utente deve poter inserire una descrizione opzionale del Protocol creato	Window

**Tabella 4:** Tracciamento requisiti-componenti

## A Descrizione dei design pattern

### A.1 Design pattern architetturali

#### A.1.1 MVC

##### Contesto

L'applicazione deve fornire una interfaccia grafica (GUI<sub>G</sub>) costituita da più schermate, che mostrano vari dati all'utente. Inoltre, le informazioni che vengono visualizzate, devono essere sempre quelle aggiornate.

##### Problema

L'applicazione deve avere una natura modulare e basata sulle responsabilità, al fine di ottenere una vera e propria applicazione component-based. Questo è conveniente per poter più facilmente gestire la manutenzione dell'applicazione.

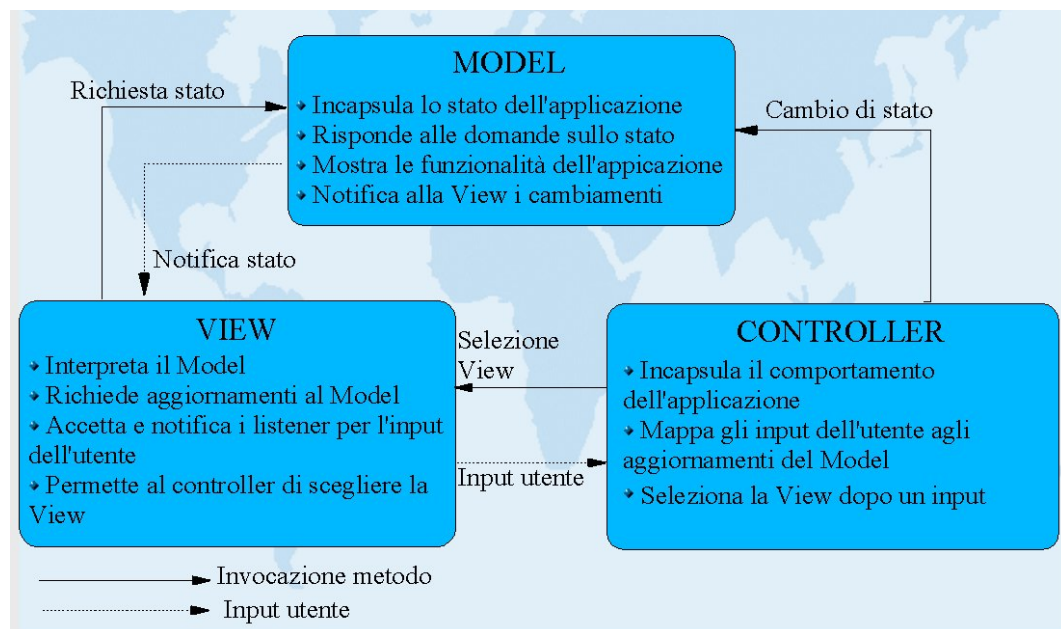
Appare quindi chiaro il bisogno di un'architettura che permetta la separazione netta tra i componenti software che gestiscono il modo di presentare i dati, e i componenti che gestiscono i dati stessi.

##### Soluzione e struttura

L'applicazione deve separare i componenti software che implementano il modello delle funzionalità di business, dai componenti che implementano la logica di presentazione e di controllo che utilizzano tali funzionalità. Vengono quindi definiti tre tipologie di componenti che soddisfano tali requisiti:

- **Model:** implementa le funzionalità di business;
- **View:** implementa la logica di presentazione;
- **Controller:** implementa la logica di controllo.

La figura seguente ha lo scopo di offrire una rappresentazione della struttura del design pattern MVC.



**Figura 42:** Diagramma del design pattern MVC

##### Scopo

Disaccoppiare le tre componenti.



## Partecipanti e responsabilità

- **Model:** Analizzando la figura 42, si evince che il core dell'applicazione viene implementato dal Model, che, incapsulando lo stato dell'applicazione, definisce i dati e le operazioni che possono essere eseguite su questi. Definisce quindi le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento.

Per lo sviluppo del Model, è vivamente consigliato utilizzare le tipiche tecniche di progettazione object oriented, al fine di ottenere un componente software che astragga al meglio i concetti importati dal mondo reale. Il Model può inoltre avere la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare agli occhi degli utenti dati sempre aggiornati;

- **View:** La logica di presentazione dei dati viene gestita solo e solamente dalla View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (GUI<sub>G</sub>), che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema. Ogni GUI<sub>G</sub> può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione. Per far sì che i dati presentati siano sempre aggiornati, è possibile adottare due strategie note come "push model" e "pull model".

Il push model adotta il pattern Observer, registrando le View come osservatori del Model. Le View possono quindi richiedere gli aggiornamenti al Model in tempo reale, grazie alla notifica di quest'ultimo. Benché questa rappresenti la strategia ideale, non è sempre applicabile. In tali casi è possibile utilizzare il "pull Model", dove la View richiede gli aggiornamenti quando "lo ritiene opportuno". Inoltre, la View delega al Controller l'esecuzione dei processi richiesti dall'utente, dopo averne catturato gli input e la scelta delle eventuali schermate da presentare;

- **Controller:** Questo componente ha la responsabilità di trasformare le interazioni dell'utente con la View, in azioni eseguite dal Model. 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.

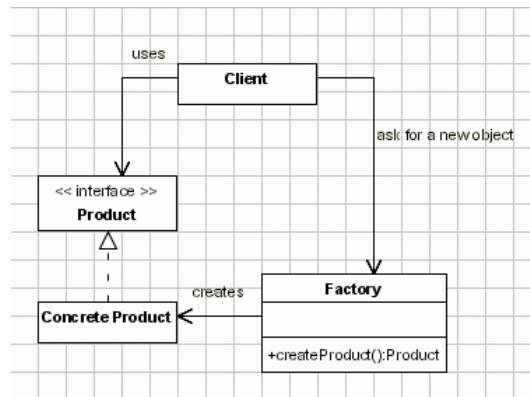
## Applicabilità

Il design pattern MVC<sub>G</sub> 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.

## A.2 Design pattern creazionali

### A.2.1 Factory



**Figura 43:** Diagramma del design pattern Factory

#### Scopo

Definire un'interfaccia per la creazione di un oggetto, lasciando alle sottoclassi la decisione sulla classe che deve essere istanziata. Il design pattern **G** Factory consente di deferire l'istanziamento di una classe alle sottoclassi.

#### Motivazione

I framework **G** utilizzano classi astratte per definire e mantenere le relazioni tra oggetti e spesso sono anche responsabili della creazione di questi oggetti. Il framework **G** deve gestire l'istanziamento di classi, ma ha conoscenza diretta soltanto di classi astratte, che non possono essere istanziate. Il pattern fornisce una soluzione a questo problema; incapsula la conoscenza su quale specifica sottoclasse deve essere creata e sposta questa conoscenza all'esterno del framework **G**. Factory è una variante del design pattern **G** Factory Method.

#### Implementazione

Il client ha bisogno di un Concrete Product, ma invece di crearlo direttamente con l'uso dell'operatore `new` chiede di crearlo all'oggetto Factory dandogli informazioni sul tipo di oggetto da creare. Il Factory istanzia un nuovo concrete Product e ritorna al client il prodotto appena creato (facendo un cast alla classe astratta Product); Il client usa i Concrete Product come Product senza essere conscio della loro concreta implementazione.

### A.2.2 Singleton



**Figura 44:** Diagramma del design pattern Singleton

#### Scopo

Assicurare che una classe abbia una sola istanza e fornire un punto d'accesso globale a tale istanza.

#### Motivazione

È importante poter assicurare che per alcune classi esista una sola istanza. Per raggiungere questo scopo, la classe stessa ha la responsabilità di creare le proprie istanze, assicurare che nessun'altra istanza possa essere creata e fornire un modo semplice per accedere all'istanza.

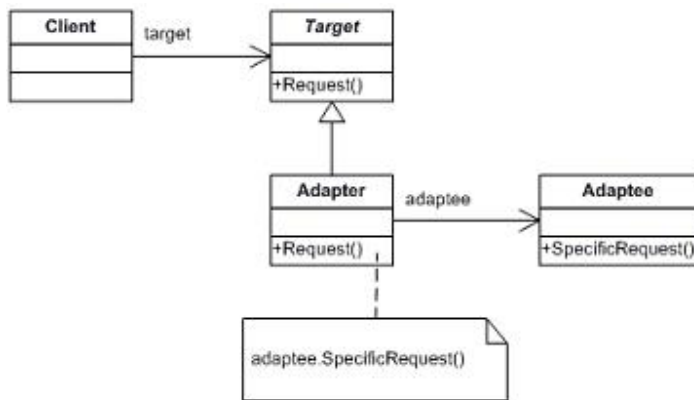
#### Applicabilità

Il design pattern **G** Singleton può essere utilizzato nei seguenti casi:

- Quando deve esistere esattamente un'istanza di una classe e tale istanza deve essere resa accessibile ai client attraverso un punto di accesso noto a tutti gli utilizzatori;
- Quando l'unica istanza deve poter essere estesa attraverso la definizione di sottoclassi e i client devono essere in grado di utilizzare le istanze estese, senza dover modificare il proprio codice.

## A.3 Design pattern strutturali

### A.3.1 Adapter



**Figura 45:** Diagramma del design pattern Adapter

#### Scopo

Convertire l'interfaccia di una classe in un'altra interfaccia richiesta dal client. Esso consente a classi diverse di operare insieme, quando ciò non è altrimenti possibile a causa di interfacce incompatibili.

#### Motivazione

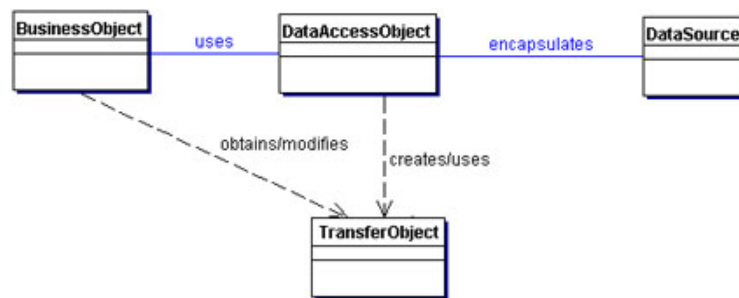
A volte, una classe di supporto che è stata progettata con obiettivi di riuso, non può essere riusata, semplicemente perché la sua interfaccia non è compatibile con l'interfaccia richiesta da un'applicazione.

#### Applicabilità

Il design pattern **Adapter** può essere utilizzato nei seguenti casi:

- Quando si vuole usare una classe esistente, ma la sua interfaccia non è compatibile con quella desiderata;
- Quando si vuole creare una classe riusabile in grado di cooperare con classi non correlate o impreviste, cioè con classi che non necessariamente hanno interfacce compatibili;
- Per gli oggetti adapter quando si devono utilizzare diverse sottoclassi esistenti, ma non è pratico adattare la loro interfaccia creando una sottoclasse per ciascuna di esse.

### A.3.2 DAO (Data Access Object)



**Figura 46:** Diagramma del design pattern DAO

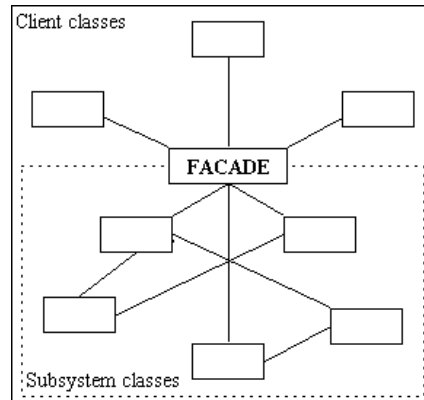
#### Caratteristiche

- Incapsula l'accesso ai dati;
- Permette la sostituzione della tecnologia di memorizzazione senza modifiche al resto del programma;
- Disaccoppia le entità dal relativo codice di persistenza.

#### Giustificazione

L'utilizzo del design pattern DAO permette di ridurre le dipendenze fra logica di business e logica di persistenza, in quanto la comunicazione con il database, viene lasciata agli oggetti propri del design pattern, rendendo il sistema manutenibile.

### A.3.3 Facade



**Figura 47:** Diagramma del design pattern Facade

#### Scopo

Fornire un'interfaccia unificata per un insieme di interfacce presenti in un sottosistema. Definisce un'interfaccia di livello più alto che rende il sottosistema più semplice da utilizzare.

#### Motivazione

Suddividere un sistema in sottosistemi aiuta a ridurre la complessità. Un obiettivo comune di progettazione è la minimizzazione delle comunicazioni e delle dipendenze fra i diversi sottosistemi. Un modo per raggiungere questo obiettivo, è introdurre un oggetto facade, che fornisce un'interfaccia unica e semplificata per accedere alle funzionalità offerte da un sottosistema.

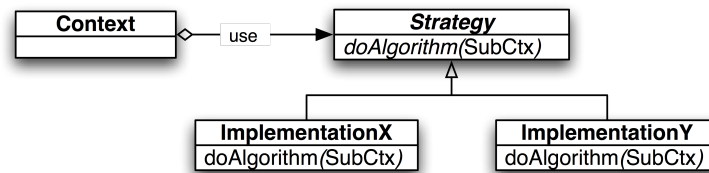
#### Applicabilità

Il design pattern **G** Facade può essere utilizzato nei seguenti casi:

- Quando si vuole fornire un'interfaccia semplice ad un sottosistema complesso, poiché fornisce una vista semplice su un sottosistema che si rivela essere sufficiente per la maggior parte dei client;
- Nei casi in cui ci sono molte dipendenze fra i client e le classi che implementano un'astrazione, in quanto si disaccoppia il sottosistema dai client e dagli altri sistemi, promuovendo portabilità e indipendenza dei sottosistemi;
- Quando si vogliono organizzare i sottosistemi in una struttura a livelli.

## A.4 Design pattern comportamentali

### A.4.1 Strategy



**Figura 48:** Diagramma del design pattern Strategy

#### Scopo

Definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Permette agli algoritmi di variare indipendentemente dai client che ne fanno uso.

#### Motivazione

Esistono molti algoritmi (strategie) che non possono essere inserite direttamente nei client:

- I client rischiano di essere troppo complessi;
- Differenti strategie sono appropriate in casi differenti;
- Difficoltà nell'aggiungere nuovi algoritmi e modificare gli esistenti.

#### Applicabilità

Il design pattern **Strategy** può essere utilizzato nei seguenti casi:

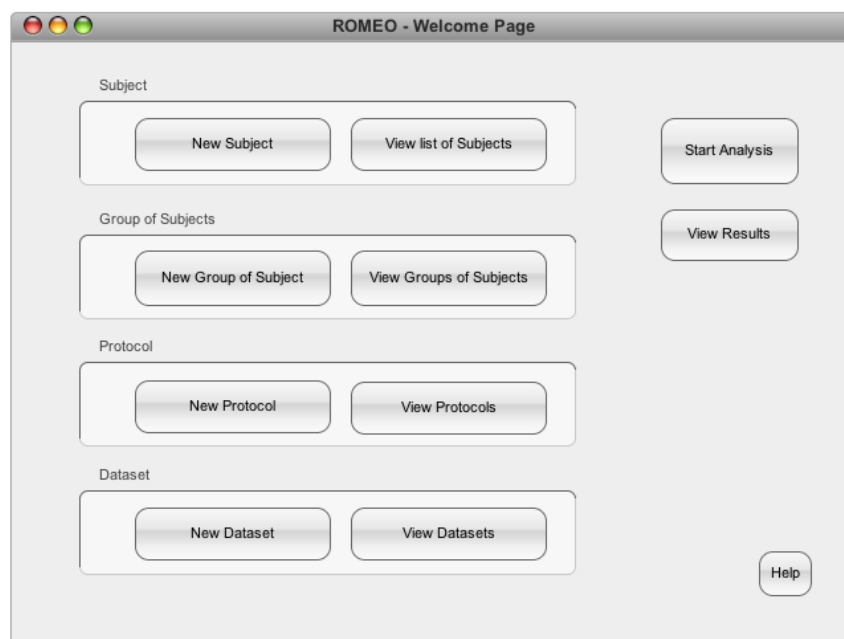
- Implementare le parti invarianti di un algoritmo una volta sola;
- Evitare la duplicazione del codice;
- Controllare le possibili estensioni di una classe;
- Un algoritmo usa una struttura dati che non dovrebbe essere resa nota ai client. Strategy può essere usato per evitare di esporre strutture dati complesse e specifiche dell'algoritmo.

## B Prototipo di UI

Al fine di ottenere il prima possibile un feedback con il proponente riguardo all'interfaccia grafica del prodotto da sviluppare, è stato creato un prototipo di UI<sub>G</sub>.

### B.1 Welcome Page

La pagina iniziale del prodotto contiene una lista delle operazioni che l'utente può eseguire. Essa è pensata in modo tale da semplificare l'utilizzo dell'utente al primo lancio del programma e in contemporanea fornire all'utente esperto l'accesso alle funzionalità principali. A sinistra sono presenti dei pulsanti che consentono la *creazione* e la *modifica/visualizzazione* di elementi. Nella parte destra invece, sono presenti due pulsanti per l'avvio dell'analisi e la visualizzazione dei risultati delle precedenti analisi. Inoltre, in ogni pagina, è presente un pulsante **Help**, che rimanda alla guida per l'uso del prodotto.



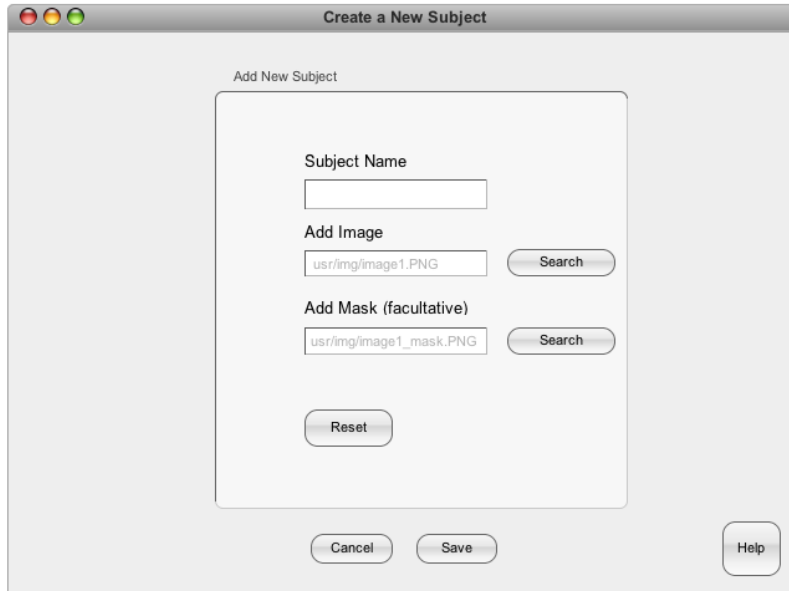
**Figura 49:** Romeo: Mock-up della pagina di benvenuto



## B.2 Pagine di creazione

### B.2.1 Creazione di un nuovo Subject

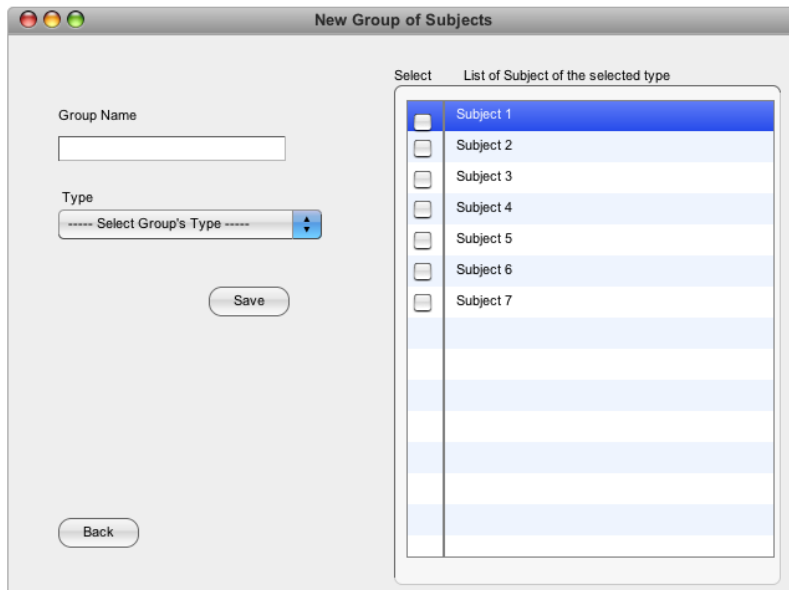
In questa pagina sarà possibile inserire un nuovo subject<sub>G</sub>, specificandone il *nome*, caricando il file dell'immagine o del video e aggiungendo un eventuale maschera<sub>G</sub>.



**Figura 50:** Mock-up della pagina di creazione di un nuovo Subject

### B.2.2 Creazione di un gruppo di Subject

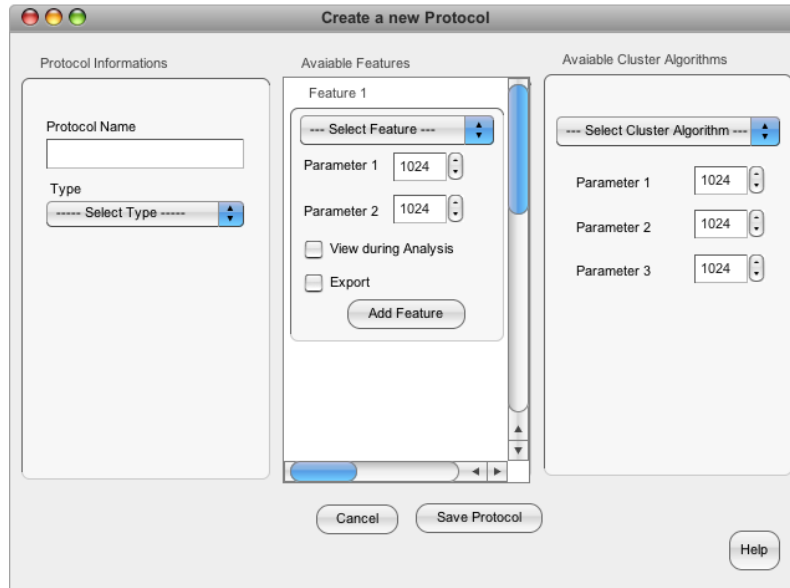
In questa pagina sarà possibile creare un nuovo gruppo di subject<sub>G</sub>, inserendo il *nome* del gruppo, specificandone il *tipo* e selezionando dall'elenco i subject esistenti per il tipo selezionato.



**Figura 51:** Mock-up della pagina di creazione di un gruppo di Subject

### B.2.3 Creazione di un Protocol

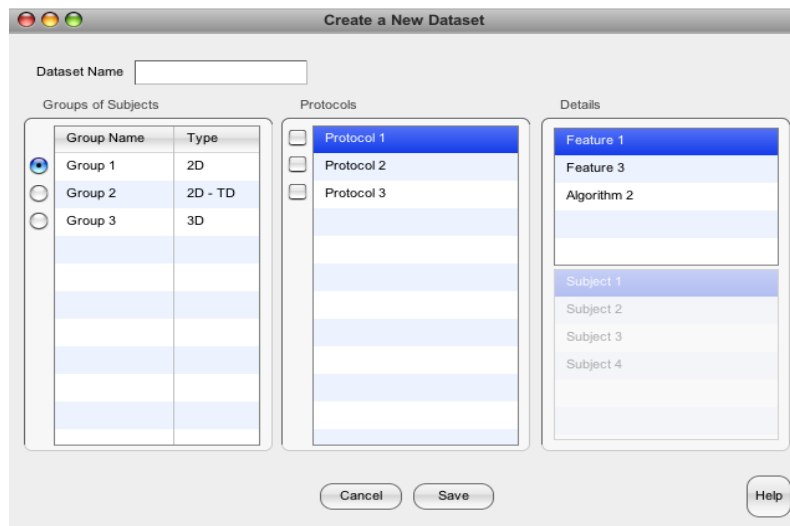
In questa pagina sarà possibile creare un nuovo protocol<sub>G</sub>, inserendo il nome del protocol<sub>G</sub>, il tipo, una o più feature<sub>G</sub> e un algoritmo di cluster<sub>G</sub> da applicare per l'analisi.



**Figura 52:** Mock-up della pagina di creazione di un Protocol

### B.2.4 Creazione di un Dataset

In questa pagina sarà possibile creare un nuovo dataset<sub>G</sub>, inserendo il nome, scegliendo uno dei gruppi di subject<sub>G</sub> esistenti e uno o più protocol<sub>G</sub> da applicare all'analisi.



**Figura 53:** Mock-up della pagina di creazione di un Dataset

### B.3 Pagine di visualizzazione e modifica

In queste pagine sarà possibile visualizzare e/o gestire<sup>4</sup> le varie entità<sup>5</sup> presenti nel sistema. A destra verrà visualizzata una lista degli elementi, mentre a sinistra verranno mostrati dei dettagli per ogni elemento selezionato.

#### B.3.1 Visualizzazione dei Subject

In questa pagina verranno visualizzati tutti i `subjectG` presenti nel sistema. Per ogni `subjectG` verrà visualizzato un pannello contenente i dettagli dell'immagine.

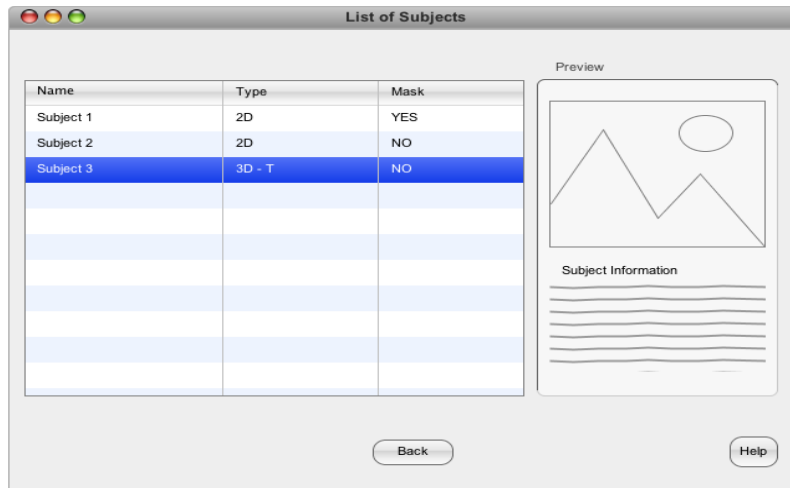


Figura 54: Mock-up della pagina di visualizzazione dei Subject

#### B.3.2 Visualizzazione dei gruppi di Subject

In questa pagina verranno visualizzati tutti i gruppi di `subjectG` presenti nel sistema. Per ogni gruppo selezionato verrà visualizzata la lista dei `subjectG` che lo compongono.

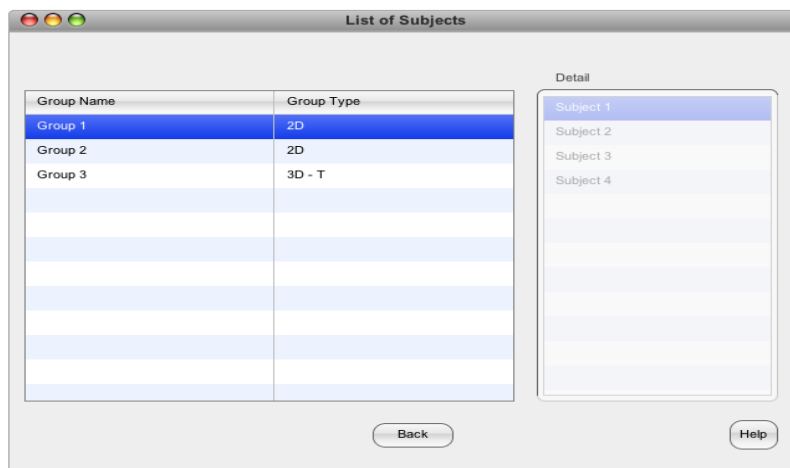


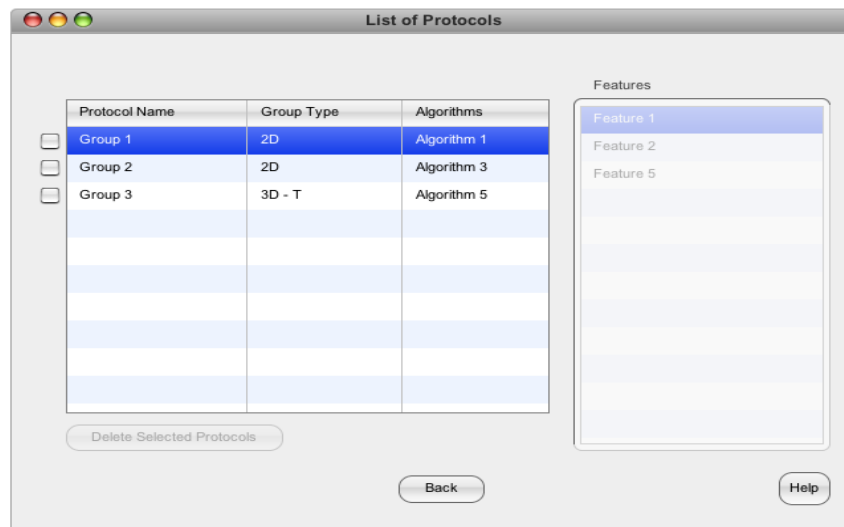
Figura 55: Mock-up della pagina di visualizzazione dei gruppi di Subject

<sup>4</sup>La gestione è prevista per tutte le entità tranne che per i `subjectG`

<sup>5</sup>Il termine è da intendersi come sostitutivo di: *subject<sub>G</sub>, gruppi di subject<sub>G</sub>, protocol<sub>G</sub> e dataset<sub>G</sub>*

### B.3.3 Visualizzazione dei Protocol

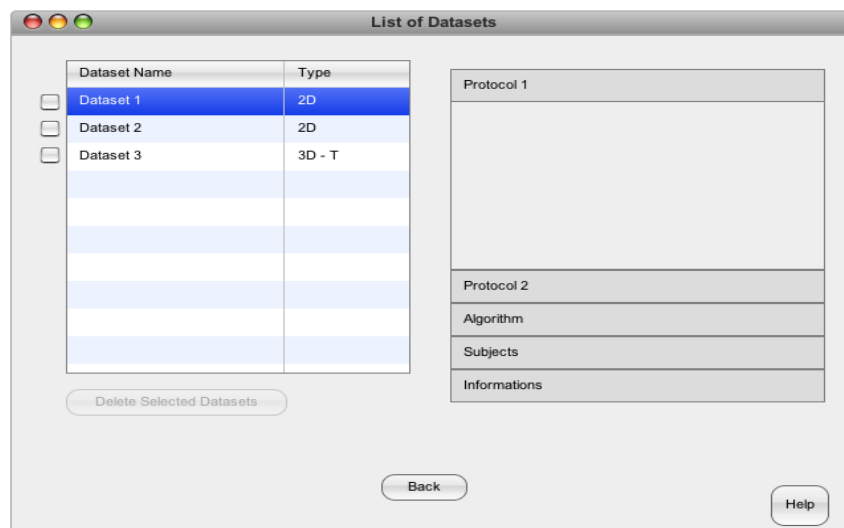
In questa pagina verranno visualizzati tutti i  $\text{protocol}_G$  creati. Per ogni  $\text{protocol}_G$  selezionato, verrà visualizzata una lista delle  $\text{feature}_G$  che lo compongono. Sarà inoltre possibile selezionare uno o più  $\text{protocol}_G$  per l'eliminazione.



**Figura 56:** Mock-up della pagina di visualizzazione dei gruppi dei Protocol

### B.3.4 Visualizzazione dei Dataset

In questa pagina verranno visualizzati i  $\text{dataset}_G$  presenti nel sistema. Per ogni  $\text{dataset}_G$  verranno visualizzati nel pannello dei dettagli, la lista dei  $\text{protocol}_G$ , l'algoritmo di  $\text{cluster}_G$ , i  $\text{subject}_G$  coinvolti nell'analisi ed eventuali informazioni aggiuntive. Sarà inoltre possibile selezionare uno o più  $\text{dataset}_G$  per l'eliminazione.



**Figura 57:** Mock-up della pagina di visualizzazione dei Dataset

## B.4 Analisi e visualizzazione risultati

### B.4.1 Avvio analisi

In questa pagina sarà possibile avviare un'analisi, selezionando un dataset tra quelli presenti nel sistema e specificando eventuali preferenze riguardo alla visualizzazione dei risultati intermedi e/o l'esportazione degli stessi.

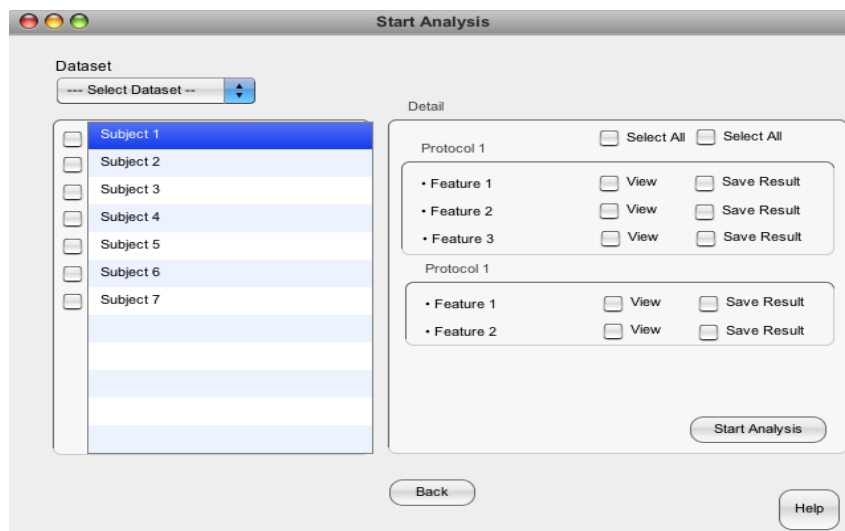


Figura 58: Mock-up della pagina di avvio analisi

### B.4.2 Esecuzione analisi

Una volta avviata l'analisi, si aprirà una finestra relativa allo stato di avanzamento della stessa. L'utente sarà inoltre in grado di visualizzare eventuali risultati intermedi (se precedentemente selezionati) e di terminare l'analisi in qualsiasi momento. Una volta che il primo risultato sarà disponibile, l'utente potrà scegliere se continuare a visualizzare i risultati intermedi, attraverso il pulsante "Continue", viceversa con il pulsante "Continue without showing results", potrà annullare la visualizzazione.

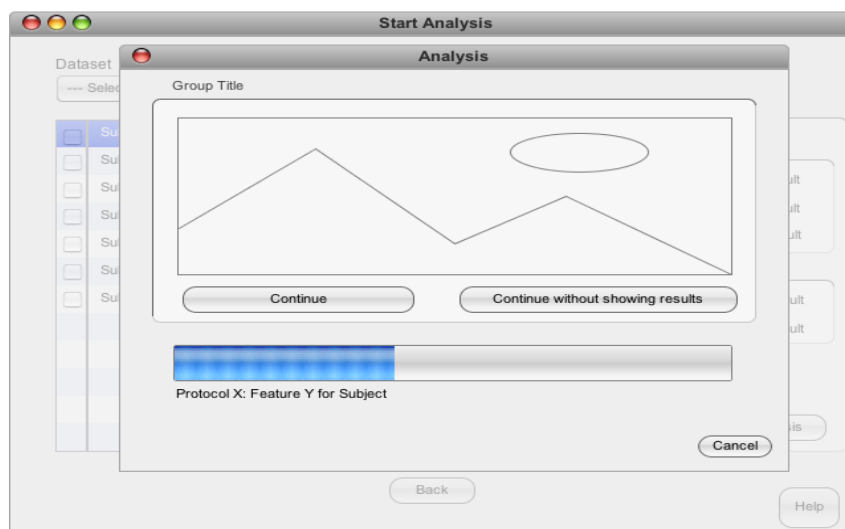


Figura 59: Mock-up della finestra di analisi

### B.4.3 Visualizzazione risultati

In questa pagina sarà possibile visualizzare i risultati delle analisi effettuate. Per ogni analisi verranno visualizzati il nome del dataset<sub>G</sub> coinvolto, lo stato dell'analisi (completata o non completata) e la data in cui è stata effettuata. Sarà inoltre possibile esportare direttamente tutti i risultati dell'analisi, attraverso il pulsante *“Export”* o visualizzare una pagina di dettaglio dei risultati, attraverso il link *“View Results”*. I risultati potranno essere filtrati per nome, data e stato.

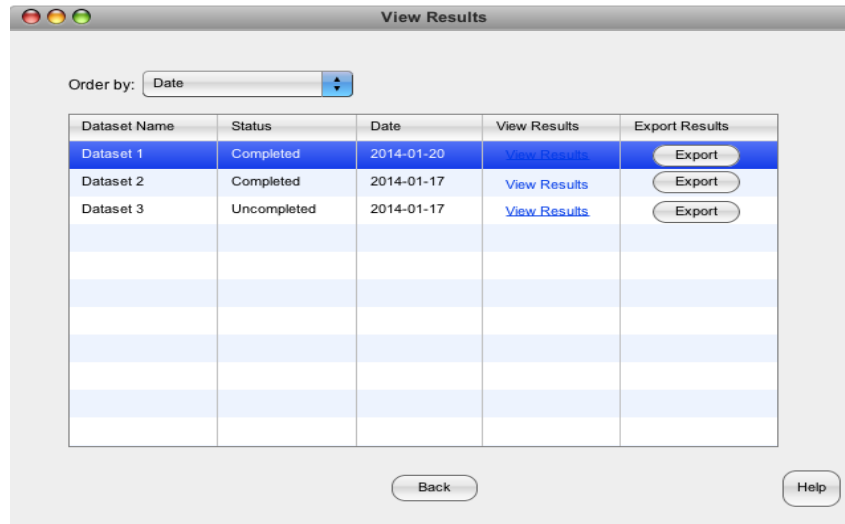


Figura 60: Mock-up della finestra dei risultati

### B.4.4 Visualizzazione dettaglio risultati

In questa pagina sarà possibile visualizzare in dettaglio i risultati di un'analisi, filtrati per protocol<sub>G</sub> (fig. 61) o per subject<sub>G</sub> (fig.62). Sarà possibile quindi, visualizzare un'anteprima delle immagini prima dell'esportazione.

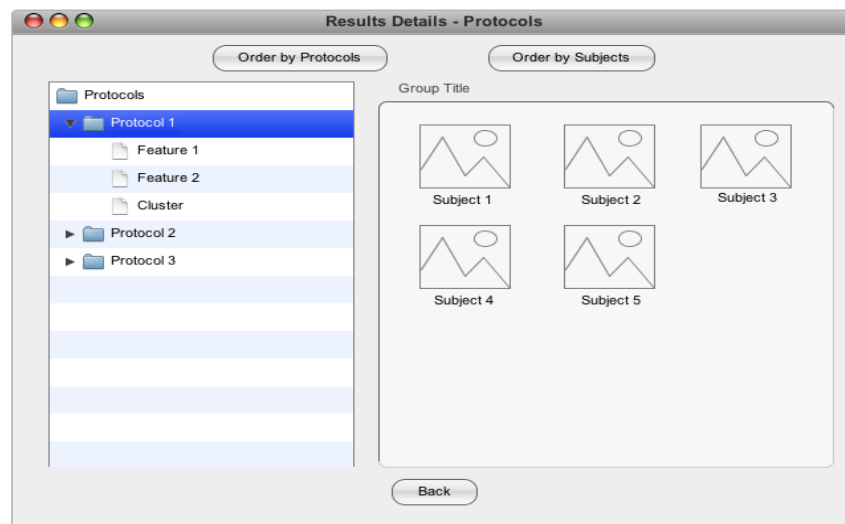
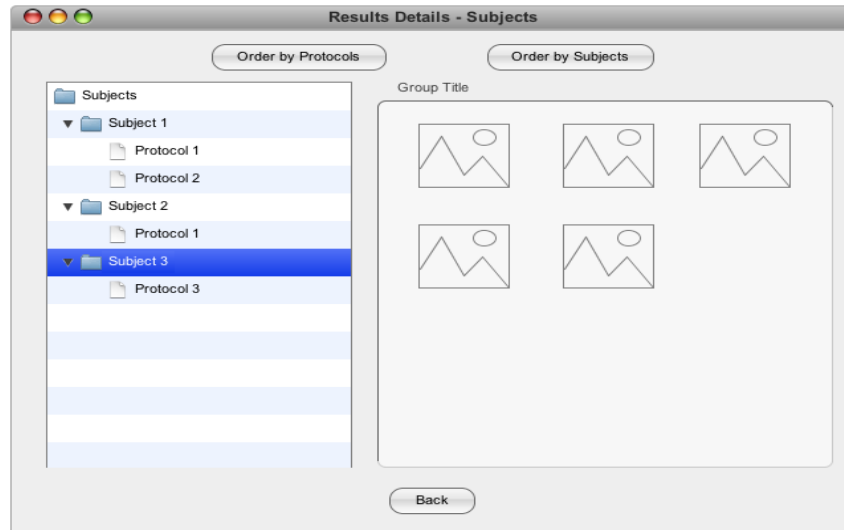


Figura 61: Mock-up della finestra di dettaglio dei risultati per Protocol



**Figura 62:** Mock-up della finestra di dettaglio dei risultati per Subject