



Specifica Tecnica

Informazioni sul documento

Nome documento	Specifica Tecnica
Versione	v3.0.0
Data redazione	2014-01-10
Redattori	<ul style="list-style-type: none">• Adami Alberto• Feltre Beatrice• Luisetto Luca• Magnabosco Nicola• Scapin Davide
Verificatori	<ul style="list-style-type: none">• Bissacco Nicolò• Martignago Jimmy
Approvazione	<ul style="list-style-type: none">• Martignago Jimmy
Lista distribuzione	<ul style="list-style-type: none">• <i>Seven Monkeys</i>• <i>Prof. Tullio Vardanega</i>• <i>Prof. Riccardo Cardin</i>
Uso	Esterno

Sommario

Questo documento contiene la specifica tecnica di Romeo.

Diario delle Modifiche

Modifica	Autore & Ruolo	Data	Versione
<i>Approvazione del documento</i>	Luisetto Luca <i>Responsabile di Progetto</i>	2014-04-31	v3.0.0
<i>Verifica del documento prima dell'approvazione</i>	Adami Alberto <i>Verificatore</i>	2014-04-30	v2.2.0
<i>Apportate modifiche in base alle segnalazioni del verificatore</i>	Martignago Jimmy <i>Progettista</i>	2014-04-29	v2.1.1
<i>Verifica del documento prima dell'approvazione</i>	Adami Alberto <i>Verificatore</i>	2014-04-28	v2.1.0
<i>Riviste tecnologie utilizzate e digrammi dei Design Pattern</i>	Luisetto Luca <i>Progettista</i>	2014-04-24	v2.0.3
<i>Riviste relazione model seguendo indicazioni correzione</i>	Scapin Davide <i>Progettista</i>	2014-04-15	v2.0.2
<i>Rivisti diagrammi dei package seguendo indicazioni correzione</i>	Feltre Beatrice <i>Progettista</i>	2014-04-12	v2.0.1
<i>Approvazione del documento</i>	Scapin Davide <i>Responsabile di Progetto</i>	2014-02-18	v2.0.0
<i>Verifica del documento prima dell'approvazione</i>	Luisetto Luca <i>Verificatore</i>	2014-02-17	v1.2.0
<i>Apportate modifiche in base alle segnalazioni del verificatore</i>	Martignago Jimmy <i>Progettista</i>	2014-02-17	v1.1.1
<i>Verifica del documento prima dell'approvazione</i>	Bissacco Nicolò <i>Verificatore</i>	2014-02-16	v1.1.0
<i>Rimozione package adapters e revisione package core</i>	Feltre Beatrice <i>Progettista</i>	2014-02-16	v1.0.5
<i>Aggiunto package QtModel con relativa descrizione</i>	Adami Alberto <i>Progettista</i>	2014-02-15	v1.0.4
<i>Aggiunto diagramma contenente tutti gli slot dei controller</i>	Adami Alberto <i>Progettista</i>	2014-02-15	v1.0.3
<i>Aggiunti diagrammi di sequenza per il design pattern MVC</i>	Feltre Beatrice <i>Progettista</i>	2014-02-14	v1.0.2
<i>Apportate modifiche in base alle segnalazioni in sede di RP</i>	Martignago Jimmy <i>Progettista</i>	2014-02-14	v1.0.1
<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

1	Introduzione	1
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
2	Tecnologie utilizzate	2
2.1	C++	2
2.2	Qt	2
2.2.1	Signals & Slots	2
2.2.2	The Meta-Object System	3
2.2.3	Licenza	3
2.3	ITK	4
2.4	VTK	4
2.5	OpenCV	4
2.6	SQLite	5
3	Descrizione architettura	6
3.1	Premesse	6
3.2	Architettura generale	7
3.2.1	Controller	8
4	Componenti e classi	10
4.1	Romeo	10
4.1.1	Informazioni sul package	10
4.1.2	Descrizione	10
4.1.3	Package contenuti	10
4.1.4	Relazioni d'uso tra i componenti	10
4.2	Romeo::Model	11
4.2.1	Informazioni sul package	11
4.2.2	Descrizione	11
4.2.3	Package contenuti	11
4.2.4	Relazioni d'uso tra i componenti	11
4.3	Romeo::Model::Core	12
4.3.1	Informazioni sul package	12
4.3.2	Descrizione	12
4.3.3	Package contenuti	12
4.3.4	Relazioni d'uso tra i componenti	13
4.3.5	Interfacce contenute	13
4.3.6	Classi contenute	14
4.4	Romeo::Model::Core::Features	23
4.4.1	Informazioni sul package	23
4.4.2	Descrizione	23
4.4.3	Classi contenute	23
4.5	Romeo::Model::Core::Algorithms	26
4.5.1	Informazioni sul package	26
4.5.2	Descrizione	26

4.5.3	Classi contenute	26
4.6	Romeo::Model::Util	28
4.6.1	Informazioni sul package	28
4.6.2	Descrizione	28
4.6.3	Package contenuti	28
4.7	Romeo::Model::Util::DAO	29
4.7.1	Informazioni sul package	29
4.7.2	Descrizione	29
4.7.3	Classi contenute	29
4.8	Romeo::Model::Util::ExporterModel	33
4.8.1	Informazioni sul package	33
4.8.2	Descrizione	33
4.8.3	Interfacce contenute	33
4.8.4	Classi contenute	34
4.9	Romeo::Model::Util::ReaderModel	35
4.9.1	Informazioni sul package	35
4.9.2	Descrizione:	35
4.9.3	Interfacce contenute	35
4.10	Romeo::Model::Util::Log	37
4.10.1	Informazioni sul package	37
4.10.2	Descrizione	37
4.10.3	Classi contenute	37
4.11	Romeo::Model::qtModel	38
4.11.1	Informazioni sul package	38
4.11.2	Descrizione	38
4.11.3	Classi contenute	38
4.12	Romeo::Model::Help	39
4.12.1	Informazioni sul package	39
4.12.2	Descrizione	39
4.12.3	Classi contenute	39
4.13	Romeo::View	40
4.13.1	Informazioni sul package	40
4.13.2	Descrizione	40
4.13.3	Package contenuti	40
4.13.4	Relazioni tra i componenti	40
4.14	Romeo::View::Window	41
4.14.1	Informazioni sul package	41
4.14.2	Descrizione	41
4.14.3	Relazioni tra i componenti	41
4.14.4	Classi contenute	41
4.15	Romeo::View::Dialog	47
4.15.1	Informazioni sul Package	47
4.15.2	Descrizione	47
4.15.3	Classi contenute	47
4.16	Romeo::View::Component	49
4.16.1	Informazioni sul package	49
4.16.2	Descrizione	49
4.17	Romeo::Controllor	50
4.17.1	Informazioni sul package	50
4.17.2	Descrizione	50

4.17.3	Relazioni tra i componenti	50
4.17.4	Classi contenute	50
5	Design pattern	58
5.1	Design pattern architetturali	58
5.1.1	MVC	58
5.2	Design pattern creazionali	59
5.2.1	Factory	59
5.2.2	Singleton	60
5.3	Design pattern strutturali	61
5.3.1	Adapter	61
5.3.2	DAO (Data Access Object)	62
5.3.3	Proxy	63
5.4	Design pattern comportamentali	64
5.4.1	Strategy	64
6	Database Romeo	65
6.1	Descrizione testuale delle tabelle	66
6.1.1	Subject	66
6.1.2	GroupOfSubject	66
6.1.3	Dataset	66
6.1.4	Protocol	66
6.1.5	ClusterAlgorithm	67
6.1.6	Feature	67
6.1.7	Analysis	67
6.1.8	Analysis_Features_Save_Show	67
6.2	Descrizione delle associazioni	68
6.3	Progettazione logica	68
7	Diagrammi delle attività	70
7.1	Attività principali	70
7.2	New Subject	72
7.2.1	Load File	73
7.3	New Group	74
7.4	New Protocol	75
7.5	New Dataset	76
7.6	Show Subjects	77
7.7	Manage Groups	78
7.8	Manage Protocols	79
7.9	Manage Datasets	80
7.10	Do an Analys	81
7.10.1	Visualizzare i risultati dell'analisi	84
7.11	Visualizzare le analisi effettuate	85
7.12	Aprire la guida	86
8	Stime di fattibilità e risorse necessarie	87
9	Tracciamento	88
9.1	Tracciamento componenti-requisiti	88
9.2	Tracciamento requisiti-componenti	92

A	Descrizione dei design pattern	100
A.1	Design pattern architetturali	100
A.1.1	MVC	100
A.2	Design pattern creazionali	102
A.2.1	Factory	102
A.2.2	Singleton	103
A.3	Design pattern strutturali	104
A.3.1	Adapter	104
A.3.2	DAO (Data Access Object)	105
A.3.3	Proxy	105
A.4	Design pattern comportamentali	107
A.4.1	Strategy	107
B	Prototipo di UI	108
B.1	Welcome Page	108
B.2	Pagine di creazione	109
B.2.1	Creazione di un nuovo Subject	109
B.2.2	Creazione di un gruppo di Subject	109
B.2.3	Creazione di un Protocol	110
B.2.4	Creazione di un Dataset	110
B.3	Pagine di visualizzazione e modifica	111
B.3.1	Visualizzazione dei Subject	111
B.3.2	Visualizzazione dei gruppi di Subject	111
B.3.3	Visualizzazione dei Protocol	112
B.3.4	Visualizzazione dei Dataset	112
B.4	Analisi e visualizzazione risultati	113
B.4.1	Avvio analisi	113
B.4.2	Esecuzione analisi	113
B.4.3	Visualizzazione risultati	114
B.4.4	Visualizzazione dettaglio risultati	114

Elenco delle figure

1	Schema interazione oggetti tramite <i>Signals & Slots</i>	3
2	Vista package dell'architettura di Romeo	7
3	Diagrammi delle classi del componente controller	8
4	Diagramma package <i>Romeo</i>	10
5	Diagramma package <i>Romeo::Model</i>	11
6	Diagramma package <i>Romeo::Model::Core</i>	12
7	Diagramma package <i>Romeo::Model::Core::Features</i>	23
8	Diagramma package <i>Romeo::Model::Core::Algorithms</i>	26
9	Diagramma package <i>Romeo::Model::Util</i>	28
10	Diagramma package <i>Romeo::Model::Util::DAO</i>	29
11	Package <i>Romeo::Model::Util::ExporterModel</i>	33
12	Pacakege <i>Romeo::Model::Util::ReaderModel</i>	35
13	Diagramma package <i>Romeo::Model::Util::Log</i>	37
14	Diagramma package <i>Romeo::Model::qtModel</i>	38
15	Diagramma package <i>Romeo::Model::Help</i>	39
16	Componente <i>Romeo::View</i>	40
17	Relazioni tra le classi del package <i>Romeo::View</i>	41
18	Diagramma package <i>Romeo::View::Window</i>	42
19	Componente <i>Romeo::View::Dialog</i>	47
20	Componente <i>Romeo::View::Window</i>	49
21	Componente <i>Romeo::Controller</i>	51
22	Diagramma di attività del pattern MVC	58
23	Diagramma delle classi del pattern MVC	58
24	Utilizzo di Factory in Romeo	59
25	Utilizzo di Singleton in Romeo	60
26	Utilizzo Adapter in Romeo	61
27	Utilizzo di DAO in Romeo	62
28	Utilizzo di Proxy in Romeo	63
29	Utilizzo di Strategy in Romeo	64
30	Struttura del database di Romeo	65
31	Diagramma Attività - Attività principali dell'applicativo Romeo . . .	71
32	Diagramma Attività - Creazione nuovo Subject	72
33	Diagramma Attività - Caricamento di un file	73
34	Diagramma Attività - Creazione nuovo gruppo di Subject	74
35	Diagramma Attività - Creazione di un nuovo Protocol	75
36	Diagramma Attività - Creazione di un nuovo Dataset	76
37	Diagramma Attività - Visualizzazione dei Subject	77
38	Diagramma Attività - Gestione dei gruppi di Subject	78
39	Diagramma Attività - Gestione dei Protocol	79
40	Diagramma Attività - Gestione dei Dataset	80
41	Diagramma Attività - Avvio di un'analisi	81
42	Diagramma Attività - Esecuzione analisi per ogni Protocol	82
43	Diagramma Attività - Esecuzione analisi per ogni Protocol	83
44	Diagramma Attività - Visualizzazione dei risultati dell'analisi effettuata	84
45	Diagramma Attività - Visualizzazione di tutte le analisi effettuate . .	85
46	Diagramma Attività - Apertura della guida	86
47	Diagramma del design pattern MVC	100
48	Diagramma del design pattern Factory	102

49	Diagramma del design pattern Singleton	103
50	Diagramma del design pattern Adapter	104
51	Diagramma del design pattern DAO	105
52	Diagramma del design pattern Proxy	105
53	Diagramma del design pattern Strategy	107
54	Romeo: Mock-up della pagina di benvenuto	108
55	Mock-up della pagina di creazione di un nuovo Subject	109
56	Mock-up della pagina di creazione di un gruppo di Subject	109
57	Mock-up della pagina di creazione di un Protocol	110
58	Mock-up della pagina di creazione di un Dataset	110
59	Mock-up della pagina di visualizzazione dei Subject	111
60	Mock-up della pagina di visualizzazione dei gruppi di Subject	111
61	Mock-up della pagina di visualizzazione dei gruppi dei Protocol	112
62	Mock-up della pagina di visualizzazione dei Dataset	112
63	Mock-up della pagina di avvio analisi	113
64	Mock-up della finestra di analisi	113
65	Mock-up della finestra dei risultati	114
66	Mock-up della finestra di dettaglio dei risultati per Protocol	114
67	Mock-up della finestra di dettaglio dei risultati per Subject	115

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_G 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_G 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_G.

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 v3.0.0* 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 v4.0.0*
- **Norme di Progetto:** *Norme di Progetto v4.0.0*

1.4.2 Informativi

- **Descrizione design pattern:** http://sourcemaking.com/design_patterns;
- **Descrizione design pattern Factory:** [http://www.oodeesign.com/factory-pattern.html;](http://www.oodeesign.com/factory-pattern.html)
- **Documentazione Qt Signals & Slots:** [https://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html;](https://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html)
- **Documentazione Qt_G:** [http://qt-project.org/doc/qt-5/classes.html;](http://qt-project.org/doc/qt-5/classes.html)
- **Documentazione ITK_G:** [http://www.itk.org/ItkSoftwareGuide.pdf;](http://www.itk.org/ItkSoftwareGuide.pdf)
- **Documentazione VTK_G:** [http://www.vtk.org/doc/release/5.10/html/;](http://www.vtk.org/doc/release/5.10/html/)
- **Documentazione SQLite:** [http://www.sqlite.org/docs.html;](http://www.sqlite.org/docs.html)
- **Design Pattern: Elementi per il riuso di software a oggetti - E. Gamma, R. Helm, R. Johnson, J. Vlissides - 1^a Edizione (2002)** capitoli 3.5, 4.1, 4.5, 5.9.

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_G Qt_G per lo sviluppo delle classi dell'architettura. Questa scelta è stata fatta perché Qt_G offre i seguenti vantaggi:

- **Framework C++:** Qt_G è un framework_G basato principalmente sul linguaggio C++_G;
- **Forte componente grafica:** tale framework_G è notoriamente orientato ad una particolare cura verso l'aspetto front-end delle applicazioni. Data l'importanza della GUI_G nel progetto Romeo, Qt_G 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*_G, per cui diventa facile assegnare i comportamenti agli elementi grafici;
- **Qt Creator:** IDE_G multiplatforma per lo sviluppo di applicazioni Qt_G. Esso integra il controllo di versione appoggiandosi anche a Git_G;
- **Esperienza del gruppo:** tutti i componenti hanno già avuto contatto con il framework_G. Questa scelta mira anche a minimizzare il tempo di studio individuale delle tecnologie.

2.2.1 Signals & Slots

Il meccanismo di *Signals & Slots*_G, è una caratteristica fondamentale del framework_G Qt_G che si occupa di far comunicare tra di loro gli oggetti.

Questo aspetto è determinante nel momento in cui si vuole sviluppare una GUI_G, 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** (classe base del framework_G), possono contenere *Signals*_G e *Slots*_G.

In Qt_G, viene emesso un *Signal*_G 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*_G predefiniti, ma è sempre possibile creare delle sottoclassi che ereditano dalle precedenti ed implementare i *Signals*_G che si desiderano. Gli *Slots*_G sono delle funzioni che vengono invocate in risposta ad un particolare *Signal*_G. Anche in questo caso, alcune classi hanno degli *Slots*_G predefiniti, ma è possibile crearne di personalizzati ereditando le classi opportune. Per collegare il *Signal*_G di un oggetto con lo *Slot*_G di un altro, è necessario utilizzare la primitiva (vedi fig. 1):

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

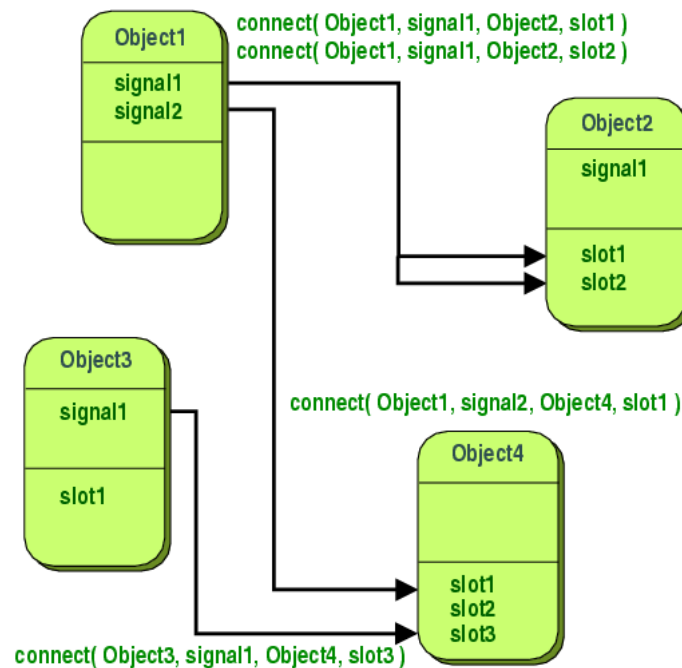


Figura 1: Schema interazione oggetti tramite *Signals & Slots*

Questo meccanismo è *type safe*: la segnatura del *Signal_G* deve combaciare con la segnatura dello *Slot_G* ad esso associato. Inoltre queste due entità sono estremamente disaccoppiate, nel senso che l'oggetto emettente il *Signal_G*, 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_G* di *Qt_G* ed inoltre, offre supporto per l'*RTTI_G* (RunTime Type Identification) 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_G* (RunTime Type Information)_G, *Signals_G* e *Slots_G*;
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_G *Qt_G* 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*)¹. 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*)². 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 le altre librerie con cui si svilupperà Romeo, sono licenziati compatibilmente con la GPL, si è deciso di adottare Qt_G con questa licenza.

2.3 ITK

Si è deciso di utilizzare la libreria ITK_G, 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_G e gli algoritmi di clustering_G.

ITK_G è stato scelto principalmente per i seguenti motivi:

- Indicazione dei proponenti;
- Libreria interamente scritta in C++ e quindi facilmente integrabile con Qt_G;
- Libreria ideata per operare su dati biomedici; integra quindi la possibilità di manipolare immagini e dati provenienti da fMRI_G 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_G, 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_G e NIfTI_G e di poterle quindi importare e visualizzare.

VTK_G è stato scelto principalmente per i seguenti motivi:

- Indicazione dei proponenti;
- Libreria interamente scritta in C++ e compatibile con Qt_G;
- Libreria multiplatforma;
- Ideata per operare su dati biomedici.

2.5 OpenCV

Si è deciso di utilizzare la libreria OpenCV³ per leggere i video in formato 2D e per convertirli in immagini in formato ITK_G.

OpenCV è stata scelta principalmente per i seguenti motivi:

- Libreria scritta in linguaggio C++ e quindi facilmente integrabile;
- Si tratta di una libreria multiplatforma.

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

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

³<http://opencv.org>

2.6 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;
- Offre il vantaggio di memorizzare l'intero database in un unico file. In questo modo SQLite non diffonde files all'interno del calcolatore portando a dei benefici in termini di memoria occupata.

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_G 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_G 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 separare le View dai rispettivi Controller, in quanto è emersa la volontà di separare maggiormente la rappresentazione grafica della View dalla propria gestione degli eventi. Si fornisce di seguito una breve legenda dei colori utilizzati per le `packageG` e classi in particolare:

- Per i `packageG`:
 - **sfondo bianco con bordo in grassetto:** per identificare il `packageG` Romeo che contiene l'intera architettura;
 - **sfondo giallo con bordo nero:** per identificare il `packageG` Model che contiene la logica di business;
 - **sfondo verde smeraldo con bordo nero:** per identificare il `packageG` View che contiene le componenti grafiche dell'applicativo;
 - **sfondo arancio con bordo nero:** per identificare il `packageG` Controller che contiene tutti i meccanismi per far far comunicare indirettamente logica di business e logica di presentazione;
 - **sfondo più chiaro con bordo nero:** per identificare i sotto-`packageG` contenuti nei `packageG` principali (model, view e controller); ad ogni livello il colore si attenua.
- Per le classi:
 - **sfondo giallo chiaro con bordo rosso:** per identificare le classi proprie dell'applicativo Romeo;
 - **sfondo verde acceso con bordo nero:** per identificare le classi di Qt_G che verranno utilizzate per realizzare Romeo;
 - **sfondo blu chiaro con bordo nero:** per identificare le classi di ITK_G che verranno utilizzate per realizzare Romeo.

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, ad alto livello, tra di loro.

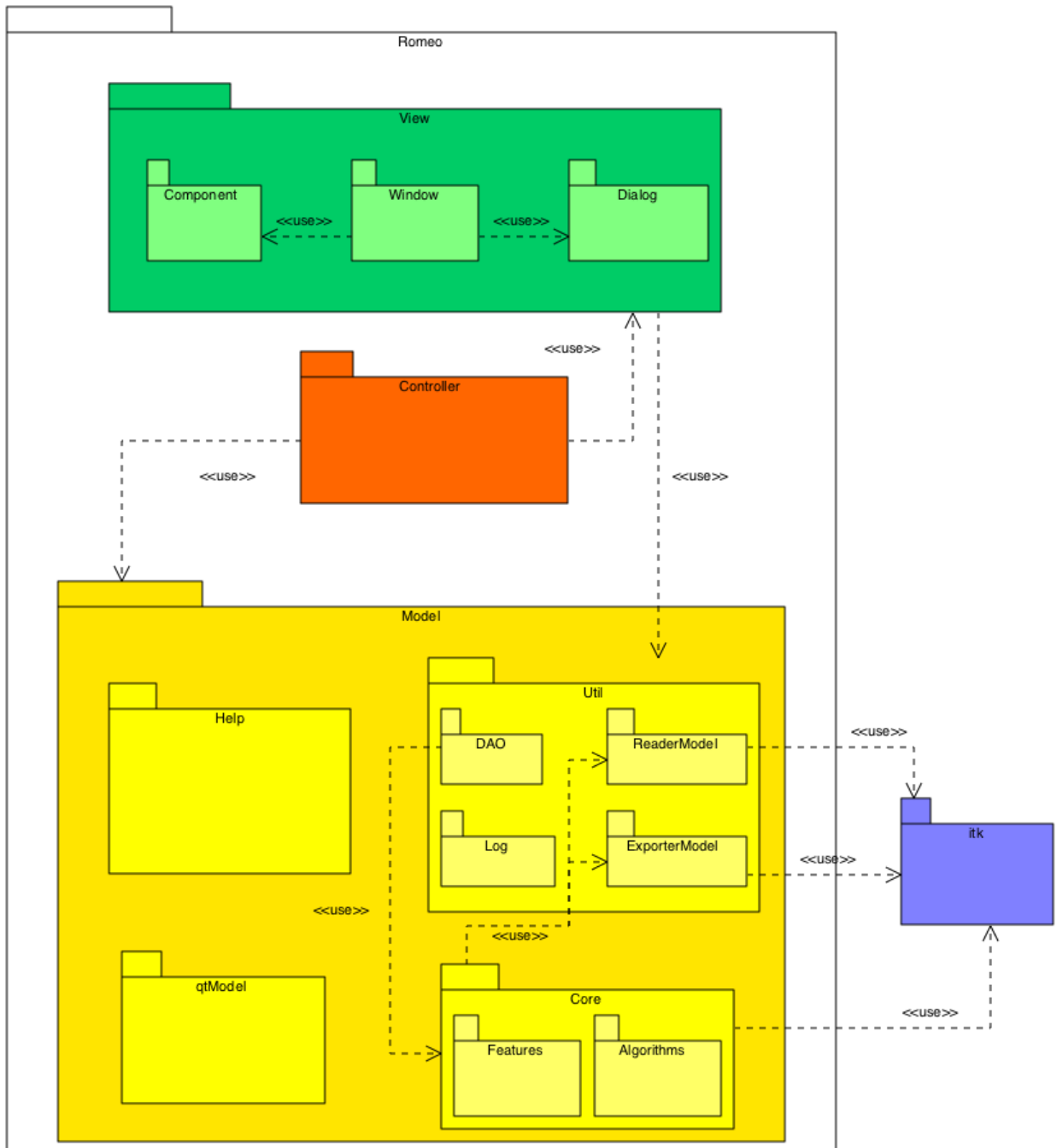


Figura 2: Vista package dell'architettura di Romeo

3.2.1 Controller

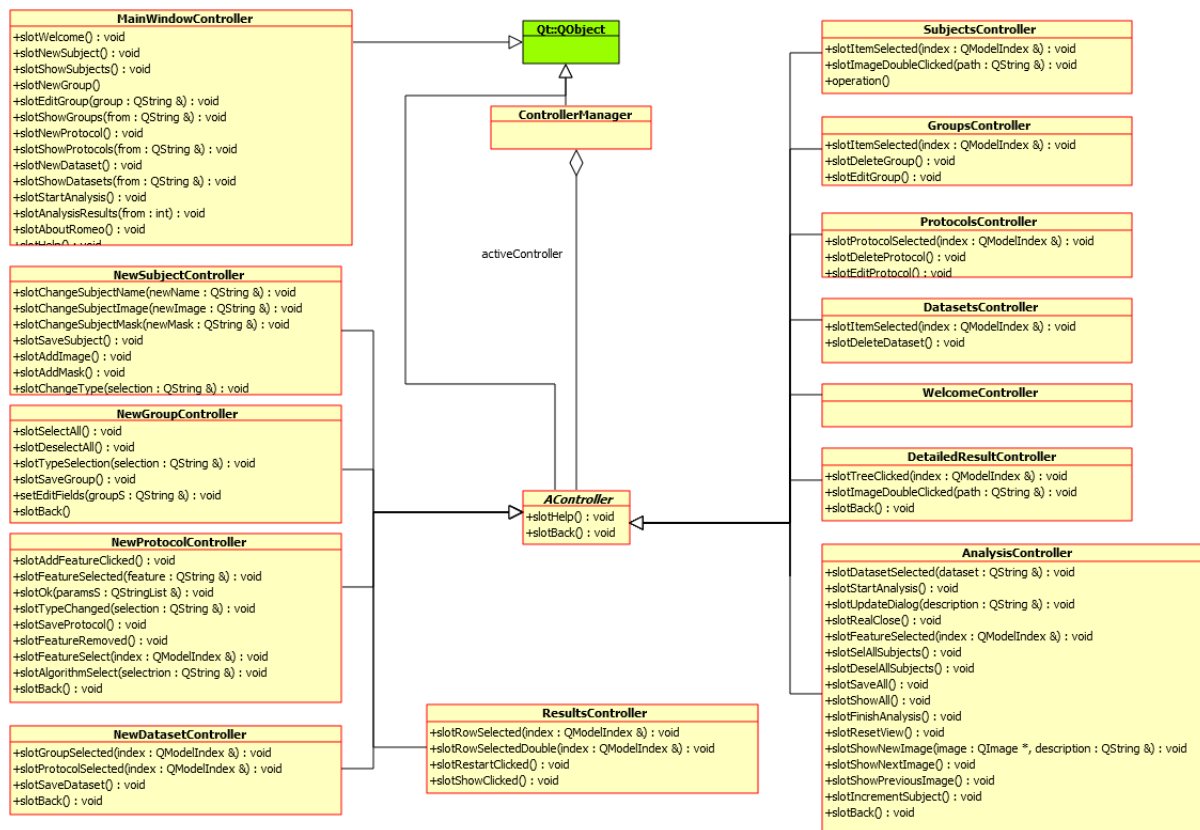


Figura 3: Diagrammi delle classi del componente controller

Nel diagramma della classi precedente sono illustrati tutti gli slot presenti nelle classi del package `Controller`.

I vari controller ricevono i vari `signal` emessi dalla view e hanno il compito di gestirli.

Ogni controller può:

- Aggiornare i dati visualizzati dalla rispettiva view;
- Modificare i dati del model;
- Aprire nuove view.

4 Componenti e classi

4.1 Romeo

4.1.1 Informazioni sul package

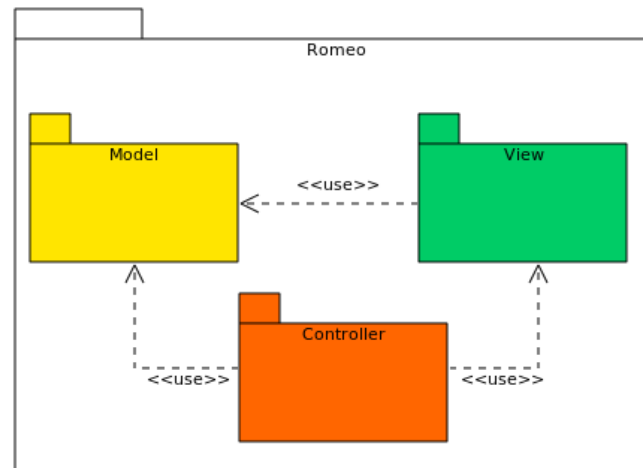


Figura 4: Diagramma package *Romeo*

4.1.2 Descrizione

Il package_G Romeo rappresenta il package_G globale del progetto.

Le relazioni tra i package_G Model, View e Controller rappresentano le relazioni tipiche del design pattern_G MVC_G.

4.1.3 Package contenuti

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

4.1.4 Relazioni d'uso tra i componenti

I package_G contenuti nel package_G Romeo, rispettano il design pattern_G MVC_G. 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

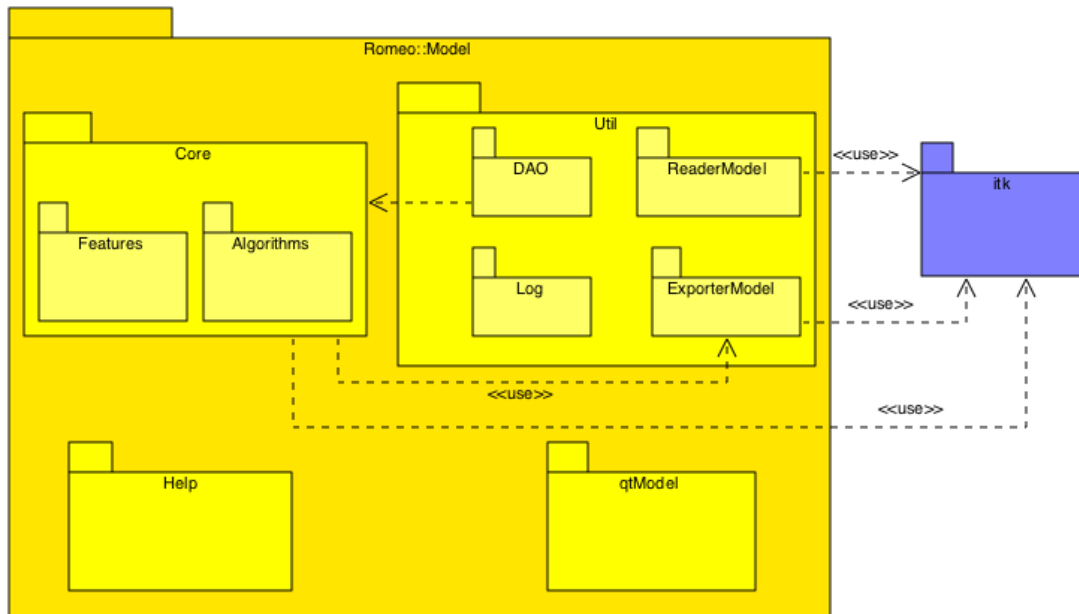


Figura 5: Diagramma package *Romeo::Model*

4.2.2 Descrizione

Package_G che rappresenta il componente model nell'architettura MVC_G.

4.2.3 Package contenuti

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

4.2.4 Relazioni d'uso tra i componenti

In una visuale ad alto livello si nota che il package_G Core usa un package_G esterno, itk, per la manipolazione delle immagini con costrutti già esistenti e ben consolidati. Inoltre usa il package_G ExporterModel contenuto in Util. Il sotto-package_G contenuto in Util usa il package_G Core, infine i package_G ReaderModel ed ExporterModel in Util usano il package_G esterno, itk, per la lettura e l'esportazione di immagini.

Infine il package `qtModel` viene utilizzato dalle classi del package `view`.

4.3 Romeo::Model::Core

4.3.1 Informazioni sul package

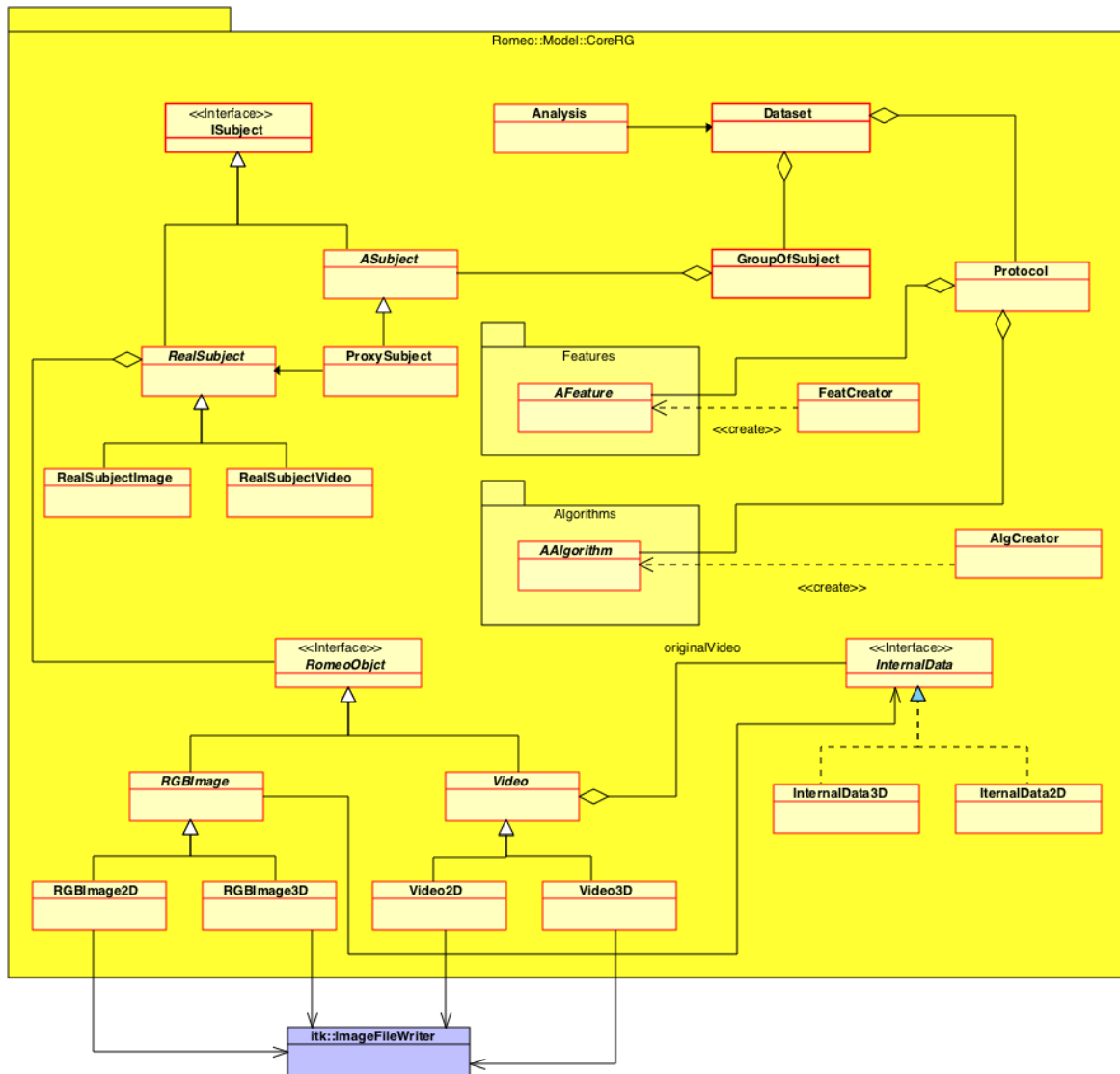


Figura 6: Diagramma package *Romeo::Model::Core*

4.3.2 Descrizione

Package_G contenente le classi rappresentanti le funzionalità principali del software. In questo package_G verrà utilizzato il design pattern_G Adapter, il quale consente di utilizzare alcune funzionalità offerte dalle librerie esterne.

4.3.3 Package contenuti

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

4.3.4 Relazioni d'uso tra i componenti

- La classe *ProxySubject* avrà un riferimento polimorfo ad un *RealSubject* e si occuperà di gestirne la creazione e le richieste;
- La classe *RealSubject* avrà un riferimento polimorfo ad un oggetto *RomeoObject*;
- La classe *RGBImage* avrà tre riferimenti polimorfi a *InternalData*, ognuno rappresentante un'immagine per ciascuno dei tre livelli di colore (Red, Green e Blue);
- La classe *Video* avrà un vettore di riferimenti ad oggetti *InternalData*, rappresentanti i *frame* del video;
- La classe *GroupOfSubject* avrà una lista di riferimenti polimorfi alla classe *ASubject*, tanti quanti sono i *Subject_G* presenti nel gruppo;
- La classe *Dataset* avrà un riferimento alla classe *GroupOfSubject* e ad uno o più *Protocol*;
- La classe *Analysis* avrà un riferimento alla classe *Dataset*, rappresentante il *Dataset_G* su cui eseguire l'analisi;
- Le classi *FeatCreator* e *AlgCreator* si occuperanno di creare le *feature_G* e gli algoritmi, tra quelli disponibili.

4.3.5 Interfacce contenute

ISubject

Descrizione: definisce l'interfaccia comune per le classi *ASubject* e *RealSubject*, fornisce un metodo per ottenere il formato interno per l'immagine del *Subject_G*. Rappresenta il componente Subject del design pattern *Proxy*.

Utilizzo: viene utilizzata quando le componenti di *Romeo_G* necessitano di riferirsi a un *Subject_G* per ottenere il formato interno ad esso associato.

Implementata da:

- *Romeo::Model::Core::ASubject*;
- *Romeo::Model::Core::RealSubject*.

Relazioni con altre classi:

- *Romeo::Model::Core::RomeoObject*: dipendenza uscente, i contratti forniti dall'interfaccia *ISubject* ritornano oggetti di tipo *RomeoObject*.

InternalData

Descrizione: definisce l'interfaccia per accedere alle informazioni sulle dimensioni di un'immagine.

Rappresenta il componente Target del design pattern **G** Adapter.

Utilizzo: viene utilizzata all'interno delle feature, ogni qualvolta ci si aspetta un'immagine generica.

Implementata da:

- `Romeo::Model::Core::InternalData2D`;
- `Romeo::Model::Core::InternalData3D`.

Relazioni con altre classi:

- `Romeo::Model::Core::RGBImage`: relazione entrante, la classe *RGBImage* contiene tre riferimenti ad oggetti *InternalData* (red, green e blue).
- `Romeo::Model::Core::Video`: relazione entrante, la classe *Video* contiene un vettore di oggetti *InternalData*, rappresentante i *frame* del video.

RomeoObject

Descrizione: interfaccia che rappresenta un generico dato da analizzare in `RomeoG`.

Utilizzo: viene utilizzato nei metodi delle feature e algoritmi per lavorare su un generico dato.

Implementata da:

- `Romeo::Model::Core::RGBImage`;
- `Romeo::Model::Core::Video`.

Relazioni con altre classi:

- Sottoclassi di `Romeo::Model::Core::AAlgorithm`: relazione entrante, le sottoclassi di *AAlgorithm* necessitano di avere accesso al tipo *RomeoObject* per l'implementazione di alcuni metodi;
- Sottoclassi di `Romeo::Model::Core::AFeature`: relazione entrante, le sottoclassi di *AFeature* necessitano di avere accesso al tipo *RomeoObject* per l'implementazione di alcuni metodi;
- `Romeo::Model::Core::Analysis`: relazione entrante, la classe *Analysis* utilizza il tipo *RomeoObject* all'interno di alcuni metodi.

4.3.6 Classi contenute

ASubject

Descrizione: classe astratta che rappresenta un generico *SubjectG* con le relative proprietà *Nome*, *Tipo del SubjectG*, *Immagine*, *Maschera* e *Data di creazione*.

Utilizzo: viene utilizzata in seguito alla ricezione di un *signalG* da parte dei controller che necessitano di riferirsi ad uno o più oggetti di tipo *ASubject*. Inoltre viene utilizzata dalle classi DAO e dalla classe *GroupOfSubject*.

Eredita da:

- Romeo::Model::Core::ISubject.

Ereditata da:

- Romeo::Model::Core::ProxySubject.

Relazioni con alte classi:

- Romeo::Model::Core::GroupOfSubject: relazione entrante, riferimenti all'insieme di Subject_G presenti nel gruppo;
- Romeo::Model::Util::DAO::SubjectDAO: relazione entrante, la classe SubjectDAO necessita del tipo ASubject per la sua implementazione;
- Romeo::Controller::SubjectsController: relazione entrante, la classe *SubjectsController* utilizza *ASubject* per visualizzare i Subject_G presenti nel sistema.

ProxySubject

Descrizione: classe che gestisce l'accesso ad un *RealSubject*, mantiene un riferimento che consente al proxy di accedere all'oggetto rappresentato *RealSubject*. Fornisce la stessa interfaccia della classe *ASubject*, consentendo di utilizzare un oggetto *ProxySubject* quando è richiesto un oggetto *ASubject*. Rappresenta il componente Proxy del design pattern_G Proxy.

Utilizzo: viene utilizzando quando è necessario creare un Subject_G all'interno di Romeo_G.

Eredita da:

- Romeo::Model::Core::ASubject;

Relazioni con altre classi:

- Romeo::Model::Core::RealSubject: relazione uscente, riferimento al *RealSubject* che il proxy sta gestendo;
- Romeo::Model::Core::RealSubjectImage: relazione uscente, tipo dinamico del riferimento posseduto dal *Proxy* quando sta gestendo un Subject_G 2D o 3D;
- Romeo::Model::Core::RealSubjectVideo: relazione uscente, tipo dinamico del riferimento posseduto dal *Proxy* quando sta gestendo un Subject_G 2D-t o 3D-t;
- Romeo::Util::DAO::SubjectDAO: relazione entrante, la classe *SubjectDAO* necessita del tipo *ASubject* per la sua implementazione;
- Romeo::Controller::NewSubjectController: relazione entrante, viene utilizzato il tipo *ProxySubject* per creare un nuovo Subject_G all'interno di Romeo_G.

RealSubject

Descrizione: classe astratta che caratterizza l'oggetto rappresentato dal *ProxySubject*, rappresenta un oggetto reale utilizzato dalla classe *ProxySubject*. Contiene la proprietà *imageFormat*. Le sottoclassi rappresentano i componenti RealSubject del design pattern_G Proxy.

Utilizzo: viene utilizzato dalla classe *ProxySubject*, la quale contiene un riferimento al *RealSubject* che sta gestendo.

Eredita da:

- `Romeo::Model::Core::ISubject`.

Ereditata da:

- `Romeo::Model::Core::RealSubjectImage`;
- `Romeo::Model::Core::RealSubjectVideo`.

Relazioni con altre classi:

- `Romeo::Model::Core::ProxySubject`: relazione entrante, riferimento al *RealSubject* che il *ProxySubject* sta controllando;
- `Romeo::Model::Core::RomeoObject`: relazione uscente, riferimento al formato interno rappresentante l'immagine del `SubjectG`.

RealSubjectImage

Descrizione: classe concreta che rappresenta un `SubjectG` nel formato bidimensionale (2D, 3D).

Rappresenta il componente `RealSubject` del design pattern_G `Proxy`.

Utilizzo: viene utilizzata dalla classe *ProxySubject* per ottenere l'oggetto di tipo *RomeoObject* che rappresenta l'immagine, quando il *ProxySubject* sta rappresentando un `SubjectG` 2D o 3D.

Eredita da:

- `Romeo::Model::Core::RealSubject`.

Relazioni con altre classi:

- `Romeo::Model::Core::ProxySubject`: relazione entrante, tipo dinamico del riferimento posseduto da un oggetto *ProxySubject*.
- `Romeo::Model::Core::RGBImage2D`: relazione uscente, tipo dinamico del riferimento al formato interno dell'immagine del `SubjectG`, quando si sta rappresentando un `SubjectG` 2d;
- `Romeo::Model::Core::RGBImage3D`: relazione uscente, tipo dinamico del riferimento al formato interno della maschera del `SubjectG`, quando si sta rappresentando un `SubjectG` 3d.

RealSubjectVideo

Descrizione: classe concreta che rappresenta un `SubjectG` di tipo 2D-t o 3D-t. Rappresenta il componente `RealSubject` del design pattern_G `proxy`.

Utilizzo: viene utilizzata dalla classe *ProxySubject* per ottenere l'oggetto di tipo *RomeoObject* che rappresenta l'immagine, quando il *ProxySubject* sta rappresentando un `SubjectG` 2D-t o 3D-t.

Eredita da:

- `Romeo::Model::Core::RealSubject`.

Relazioni con altre classi:

- `Romeo::Model::Core::ProxySubject`: relazione entrante, tipo dinamico del riferimento posseduto da un oggetto *ProxySubject*.
- `Romeo::Model::Core::Video2D`: relazione uscente, tipo dinamico del riferimento al formato interno dell'immagine del `SubjectG`, quando si sta rappresentando un `SubjectG 2D-t`;
- `Romeo::Model::Core::Video3D`: relazione uscente, tipo dinamico del riferimento al formato interno dell'immagine del `SubjectG`, quando si sta rappresentando un `SubjectG 3D-t`;

GroupOfSubject

Descrizione: classe che rappresenta un gruppo di `SubjectG` con le relative proprietà: *Nome del gruppo, tipo del gruppo, data di creazione e la lista di Subject_G presenti nel gruppo.*

Utilizzo: viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessitano di riferirsi ad uno o più gruppi di `SubjectG`. Inoltre viene utilizzata dalla classe *Dataset*, che contiene un riferimento al `GroupOfSubjectG` associato e dalle classi DAO.

Relazioni con altre classi:

- `Romeo::Model::Core::Dataset`: relazione entrante, riferimento al Gruppo di `SubjectG` del `DatasetG`;
- `Romeo::Model::Util::DAO::GroupDAO`: relazione entrante, la classe *GroupDAO* necessita del tipo per la sua implementazione;
- `Romeo::Model::Core::ASubject`: relazione uscente, riferimento alla lista dei `SubjectG` presenti nel gruppo;
- `Romeo::Controller::NewGroupController`: relazione entrante, necessita del tipo *GroupOfSubject* per la sua implementazione;
- `Romeo::Controller::GroupsController`: relazione entrante, necessita del tipo *GroupOfSubject* per la sua implementazione.

Protocol

Descrizione: classe che rappresenta un `ProtocolG` con le relative proprietà: *Nome, Tipo, Data di creazione, Lista di feature_G e Algoritmo di cluster_G.*

Utilizzo: viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessitano di riferirsi ad uno o più `ProtocolG`. Inoltre viene utilizzata dalla classe *Dataset*, che contiene un riferimento ai `ProtocolG` associati, e dalle classi DAO.

Relazioni con altre classi:

- `Romeo::Model::Core::Dataset`: relazione entrante, lista dei `ProtocolG` presenti nel `DatasetG`;
- `Romeo::Model::Util::DAO::ProtocolDAO`: relazione entrante, la classe *ProtocolDAO* necessita del tipo per la sua implementazione;
- `Romeo::Model::Core::Features::AFeature`: relazione uscente, lista delle `featureG` presenti nel `ProtocolG`;

- `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione uscente, riferimento all'algoritmo di cluster_G del Protocol_G;
- `Romeo::Controller::NewProtocolController`: relazione entrante, la classe necessita del tipo *Protocol* per la sua implementazione;
- `Romeo::Controller::ProtocolsController`: relazione entrante, la classe necessita del tipo *Protocol* per la sua implementazione.

Dataset

Descrizione: classe che rappresenta un Dataset_G con le relative proprietà: *Nome*, *Tipo*, *Data di creazione*, *Gruppo di Subject_G* e *Protocol_G*.

Utilizzo: viene utilizzata in seguito alla ricezione di un signal_G da parte dei controller che necessitano di riferirsi ad uno o più Dataset_G. Inoltre viene utilizzata dalla classe *Analysis*, che contiene un riferimento al Dataset_G associato all'analisi, e dalle classi DAO.

Relazioni con altre classi:

- `Romeo::Model::Core::Analysis`: relazione entrante, riferimento al Dataset_G su cui eseguire l'analisi;
- `Romeo::Model::Util::DAO::DatasetDAO` relazione entrante, la classe *DatasetDAO* necessita del tipo *Dataset* per la sua implementazione;
- `Romeo::Model::Core::GroupOfSubject`: relazione uscente, gruppo di Subject_G associato al Dataset_G;
- `Romeo::Model::Core::Protocol`: relazione uscente, lista dei Protocol_G presenti nel Dataset_G;
- `Romeo::Controller::NewDatasetController`: relazione entrante, la classe *NewDatasetController* necessita del tipo *Dataset* per la sua implementazione;
- `Romeo::Controller::DatasetsController`: relazione entrante, la classe *DatasetsController* necessita del tipo *Dataset* per la sua implementazione.

Analysis

Descrizione: classe che rappresenta un' analisi con le relative proprietà: *Subject_G da analizzare*, *Directory dei risultati*, *Feature_G di cui salvare i risultati*, *Feature_G di cui visualizzare i risultati* e *Data di creazione*.

Contesto di utilizzo: viene utilizzata dalla classe *AnalysisController*, alla ricezione di un signal_G per l'avvio di una nuova analisi.

Eredita da:

- `Qt::QThread`.

Relazioni con altre classi:

- `Romeo::Model::Util::DAO::AnalysisDAO`: relazione entrante, la classe *AnalysisDAO* necessita del tipo *Analysis* per la sua implementazione;
- `Romeo::Model::Core::Dataset`: relazione uscente, riferimento al Dataset_G su cui eseguire l'analisi;
- `Romeo::Model::Core::RomeoObject`: relazione uscente, necessita del tipo *RomeoObject* per la sua implementazione.

FeatCreator

Descrizione: classe Factory avente la responsabilità di creare un oggetto della classe `Romeo::Model::Core::Features::AFeature`, che rappresenta un'istanza della `featureG` da creare.

Rappresenta il componente Factory del design pattern `G` Factory.

A seconda dei parametri e nome della `FeatureG` passati, crea un oggetto rispetto ad un altro.

Utilizzo: viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessitano di utilizzare un oggetto di tipo `Romeo::Model::Core::Algorithms::AFeature`. A seconda dei parametri e nome della `featureG` passati, crea un oggetto rispetto ad un altro.

Relazioni con altre classi:

- Sottoclassi di `Romeo::Model::Core::Features::AFeature`: relazione uscente, la classe `FeatCreator` si occupa di creare le istanze delle varie `FeatureG` disponibili;
- `Romeo::Model::Core::Features::AFeature`: relazione uscente, tipo statico di una `featureG` creata dalla classe `FeatCreator`;
- `Romeo::Controller::NewProtocolController`: relazione entrante, la classe `NewProtocolController` necessita del tipo `FeatCreator` per la sua implementazione;
- `Romeo::Controller::ProtocolsController`: relazione entrante, la classe `ProtocolsController` necessita del tipo `FeatCreator` per la sua implementazione;
- `Romeo::Controller::AnalysisController`: relazione entrante, la classe `AnalysisController` necessita del tipo `FeatCreator` per la sua implementazione.

AlgCreator

Descrizione: classe Factory avente la responsabilità di creare un oggetto di tipo `Romeo::Model::Core::Algorithms::AAlgorithm`, che rappresenta un'istanza dell' algoritmo da creare.

Rappresenta il componente Factory del design pattern `G` Factory.

Utilizzo: viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessitano di utilizzare un oggetto di tipo `Romeo::Model::Core::Algorithms::AAlgorithm`. A seconda dei parametri e nome dell' algoritmo di `ClusterG` passati, crea un oggetto rispetto ad un altro.

Relazioni con altre classi:

- Sottoclassi di `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione uscente, la classe `AlgCreator` si occupa di creare le istanze dei vari Algoritmi di `ClusterG` disponibili;
- `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione uscente, tipo statico di un algoritmo creato dalla classe Factory;
- `Romeo::Controller::NewProtocolController`: relazione entrante, la classe `NewProtocolController` necessita del tipo `AlgCreator` per la sua implementazione;
- `Romeo::Controller::ProtocolsController`: relazione entrante, la classe `ProtocolsController` necessita del tipo `AlgCreator` per la sua implementazione;
- `Romeo::Controller::AnalysisController`: relazione entrante, la classe `AnalysisController` necessita del tipo `AlgCreator` per la sua implementazione.

InternalData2D

Descrizione: classe che implementa i contratti definiti da `InternalData` e rappresenta il formato interno bidimensionale sul quale operare. Implementa il design pattern Adapter e adatta la classe `itk::Image` della libreria ITK_G. Rappresenta la componente Adapter dell'omonimo design pattern_G.

Utilizzo: viene utilizzata nelle `featureG` come parametro di input da elaborare e rappresenta i segnali canali nelle immagini in formato RGB.

Eredita da:

- `Romeo::Model::Core::InternalData`.

Relazioni con altre classi:

- `Romeo::Model::Core::RGBImage2D`: relazione entrante, tipo dinamico del riferimento posseduto dalla classe `RGBImage2D`;
- `Romeo::Model::Core::Video2D`: relazione entrante, tipo dinamico del vettore posseduto dalla classe `Video2D`;
- `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione entrante, le classi rappresentanti gli algoritmi utilizzano il tipo `InternalData2D` nella loro implementazione;
- `Romeo::Model::Core::Features::AFeature`: relazione entrante, le classi rappresentanti le feature utilizzano il tipo `InternalData2D` nella loro implementazione.

InternalData3D

Descrizione: classe che rappresenta il formato interno per un'immagine di tipo 3D. Classe che "adatta" la classe `itk::Image` fornita dalla libreria esterna ITK_G. Rappresenta il componente Adapter del design pattern_G Adapter.

Utilizzo: viene utilizzata nelle `featureG` come parametro di input da elaborare e rappresenta i segnali canali nelle immagini in formato RGB.

Eredita da:

- `Romeo::Model::Core::InternalData`.

Relazioni con altre classi:

- `Romeo::Model::Core::RGBImage3D`: relazione entrante, tipo dinamico del riferimento posseduto dalla classe `RGBImage2D`;
- `Romeo::Model::Core::Video3D`: relazione entrante, tipo dinamico del vettore posseduto dalla classe `Video2D`;
- `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione entrante, le classi rappresentanti gli algoritmi utilizzano il tipo `InternalData3D` nella loro implementazione;
- `Romeo::Model::Core::Features::AFeature`: relazione entrante, le classi rappresentanti le feature utilizzano il tipo `InternalData3D` nella loro implementazione.

RGBImage

Descrizione: classe astratta che rappresenta un'immagine RGB, composta da tre livelli di colore *red*, *green* e *blue*. Definisce dei contratti per l'accesso ai tre livelli di colore, per la loro scomposizione e fusione.

Utilizzo: viene utilizzata quando una feature_G deve essere applicata ad un immagine e non ad un video.
Inoltre viene utilizzata per salvare le informazioni riguardanti il colore dell'immagine, che altrimenti sarebbe in scala di grigi.

Eredita da:

- Romeo::Model::Core::RomeoObject.

Realazioni con altre classi:

- Romeo::Model::Core::InternalData: relazione uscente, la classe *RGBImage* contiene tre riferimenti ad oggetti di tipo *InternalData* (red, green e blue);
- Romeo::Model::Core::Analysis: relazione entrante, la classe *Analysis* necessita del tipo *RGBImage* per la sua implementazione;
- Romeo::Model::Core::Algorithms::AAlgorihm: relazione entrante, la classe *AAlgorihm* necessita del tipo *RGBImage* per la sua implementazione;
- Romeo::Model::Core::Features::AFeature: relazione entrante, la classe *AFeature* necessita del tipo *RGBImage* per la sua implementazione.

RGBImage2D

Descrizione: classe che rappresenta un immagine bidimensionale, a colori in Romeo_G. Classe che “adatta” la classe *itk::image* fornita da *itk*.
Rappresenta il componente Adapter del design pattern_G Adapter.

Utilizzo: viene utilizzata dalle feature_G che elaborano immagini bidimensionali. Inoltre racchiude in se le informazioni sul colore delle immagini importate.

Eredita da:

- Romeo::Model::Core::RGBImage.

Relazioni con altre classi:

- Romeo::Model::Core::RealSubjectImage: relazione entrante, tipo dinamico del riferimento posseduto dalla classe;
- Romeo::Model::Core::Video2D: relazione entrante, tipo dinamico del vettore di immagini RGB;

RGBImage3D

Descrizione: classe che rappresenta un immagine tridimensionale, a colori in Romeo_G. Classe che “adatta” la classe *itk::image* fornita da *itk*.
Rappresenta il componente Adapter del design pattern_G Adapter.

Utilizzo: viene utilizzata dalle feature_G che elaborano immagini tridimensionali. Inoltre racchiude in se le informazioni sul colore delle immagini importate

Eredita da:

- Romeo::Model::Core::RGBImage.

Relazioni con altre classi:

- `Romeo::Model::Core::RealSubjectImage`: relazione entrante, tipo dinamico dei riferimenti posseduti dalla classe;
- `Romeo::Model::Core::Video3D`: relazione entrante, tipo dinamico del vettore di immagini RGB;

Video

Descrizione: classe astratta che rappresenta la classe base di tutti i video in `RomeoG`. È composta da un vettore di oggetti di tipo *InternalData*.

Utilizzo: viene utilizzata dalle `FeatureG` dinamiche quando ci si aspetta un video generico.

Eredita da:

- `Romeo::Model::Core::RomeoObject`.

Relazioni con altre classi:

- `Romeo::Model::Core::InternalData`: relazione uscente, vettore di immagini rappresentante i vari *frame* del video;
- `Romeo::Model::Core::Analysis`: relazione entrante, la classe *Analysis* necessita del tipo *Video* per la sua implementazione;
- `Romeo::Model::Core::Algorithms::AAlgorithm`: relazione entrante, la classe *AAlgorithm* necessita del tipo *Video* per la sua implementazione;
- `Romeo::Model::Core::Features::AFeature`: relazione entrante, la classe *AFeature* necessita del tipo *Video* per la sua implementazione.

Video2D

Descrizione; classe concreta che rappresenta un video bidimensionale in `RomeoG`.

Utilizzo: viene utilizzata dalle classi rappresentanti le `FeatureG` dinamiche 2D come dato da processare.

Eredita da:

- `Romeo::Model::Core::Video`.

Relazioni con altre classi:

- `Romeo::Model::Core::RealSubjectVideo`: relazione entrante, tipo dinamico del vettore posseduto

Video3D

Descrizione: classe concreta che rappresenta un video bidimensionale in `RomeoG`.

Utilizzo: viene utilizzata dalle `FeatureG` dinamiche 3D come dato da processare.

Eredita da:

- `Romeo::Model::Core::Video`.

Relazioni con altre classi:

- `Romeo::Model::Core::RealSubjectVideo`: relazione entrante, tipo dinamico del vettore posseduto dalla classe.

4.4 `Romeo::Model::Core::Features`

4.4.1 Informazioni sul package

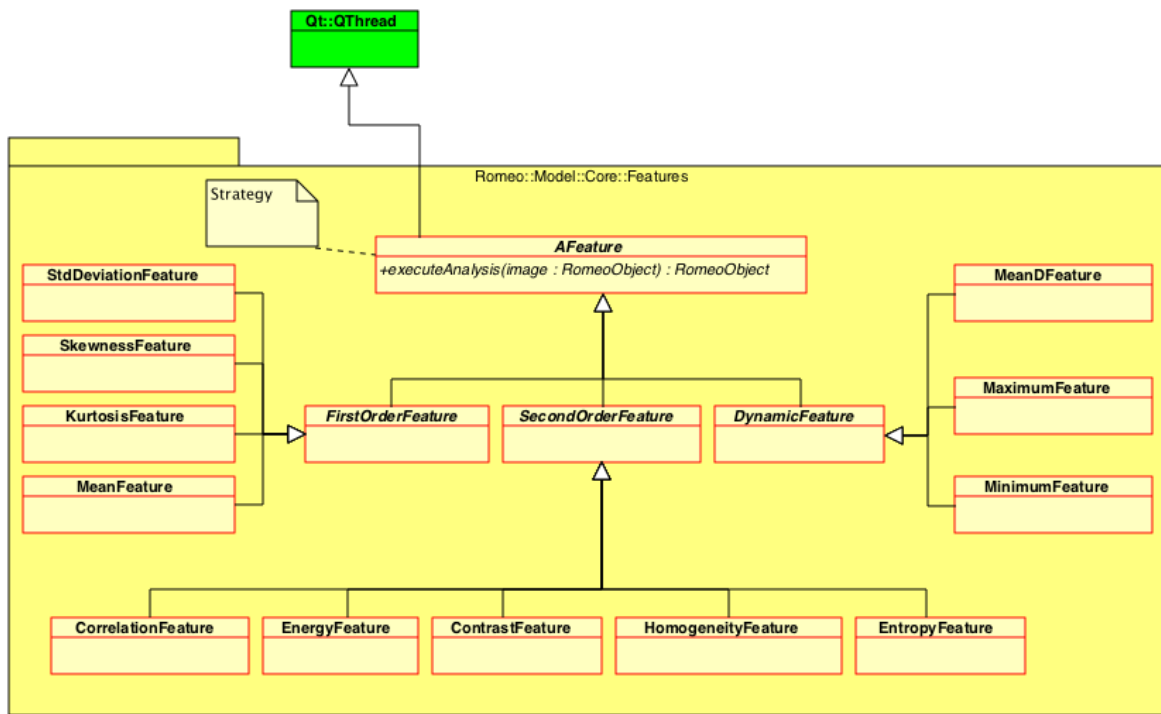


Figura 7: Diagramma package `Romeo::Model::Core::Features`

4.4.2 Descrizione

Package_G contenente le classi che permettono al model di utilizzare le features previste nei requisiti. Le classi di questo package_G sono implementate usando il design pattern_G Strategy.

4.4.3 Classi contenute

AFeature

Descrizione: classe astratta che rappresenta una generica feature_G. Definisce dei contratti per l'esecuzione delle feature_G, che dovranno essere implementati dalle sue sottoclassi. Rappresenta la componente Strategy dell'omonimo design pattern_G.

Utilizzo: fornisce i metodi per l'esecuzione di una feature_G su un'immagine bidimensionale o tridimensionale.

Eredita da:

- `Qt::QThread`.

Ereditata da:

- Romeo::Model::Core::Adapters::Features::FirstOrder;
- Romeo::Model::Core::Adapters::Features::SecondOrder;
- Romeo::Model::Core::Adapters::Features::DynamicFeature.

Relazioni con altre classi:

- Romeo::Model::Core::Protocol: relazione entrante, lista delle Feature_G presenti nel Protocol_G;
- Romeo::Model::Core::FeatCreator: relazione entrante, tipo statico di una Feature_G creata dalla classe Factory;
- Romeo::Model::Util::DAO::FeatureDAO: relazione entrante, la classe FeatureDAO necessita del tipo FeatureDAO per la sua implementazione.

FirstOrder

Descrizione: classe astratta che rappresenta una generica feature_G del primo ordine. Deriva da AFeature e rimane astratta perchè non implemente i contratti di quest'ultima. Tale classe è caratterizzata da un campo dati *window size*. Le sue sottoclassi rappresenteranno le componenti ConcreteStrategy del design pattern_G Strategy.

Utilizzo: viene utilizzata durante un'analisi recuperare le informazioni sulla dimensione della finestra, la lista dei parametri e il tipo della feature_G.

Eredita da:

- Romeo::Model::Core::Features::AFeature

Ereditata da:

- Romeo::Model::Core::Features::StdDeviationFeature;
- Romeo::Model::Core::Features::SkewnessFeature;
- Romeo::Model::Core::Features::KurtosisFeature;
- Romeo::Model::Core::Features::MeanFeature.

Relazioni con altre classi:

- Romeo::Model::Core::FeatCreator: relazione entrante, la classe FeatCreator utilizza il tipo FIRSTORDER per la sua implementazione.

SecondOrder

Descrizione: classe astratta che rappresenta una generica feature_G del secondo ordine. Deriva da AFEATURE e rimane astratta perchè non implementa i contratti di quest'ultima. Le sue sottoclassi rappresentano le componenti ConcreteStrategy del design pattern_G Strategy.

Utilizzo: viene utilizzata durante un'analisi recuperare le informazioni sulla dimensione della finestra, la lista dei parametri e il tipo della feature_G.

Eredita da:

- Romeo::Model::Core::Features::AFeature.

Ereditata da:

- Romeo::Model::Core::Features::CorrelationFeature;
- Romeo::Model::Core::Features::EnergyFeature;
- Romeo::Model::Core::Features::ContrastFeature;
- Romeo::Model::Core::Features::HomogeneityFeature;
- Romeo::Model::Core::Features::EntropyFeature.

Relazioni con altre classi:

- Romeo::Model::Core::FeatCreator: relazione entrante, la classe *FeatCreator* utilizza il tipo *SecondOrder* per la sua implementazione.

DynamicFeature

Descrizione: classe astratta che rappresenta una generica feature_G dinamica. Deriva da *AFeature* e rimane astratta perchè non implementa i contratti di quest'ultima. Le sue sottoclassi rappresenteranno le componenti ConcreteStrategy del design pattern_G Strategy.

Utilizzo: viene utilizzata durante un'analisi recuperare la lista dei parametri e il tipo della feature_G.

Eredita da:

- Romeo::Model::Core::Features::AFeature.

Ereditata da:

- Romeo::Model::Core::Features::MeanDFeature;
- Romeo::Model::Core::Features::MaximumFeature;
- Romeo::Model::Core::MinimumFeature.

Relazioni con altre classi:

- Romeo::Model::Core::FeatCreator: relazione entrante, la classe *FeatCreator* utilizza il tipo DYNAMICFEATURE per la sua implementazione.

4.5 Romeo::Model::Core::Algorithms

4.5.1 Informazioni sul package

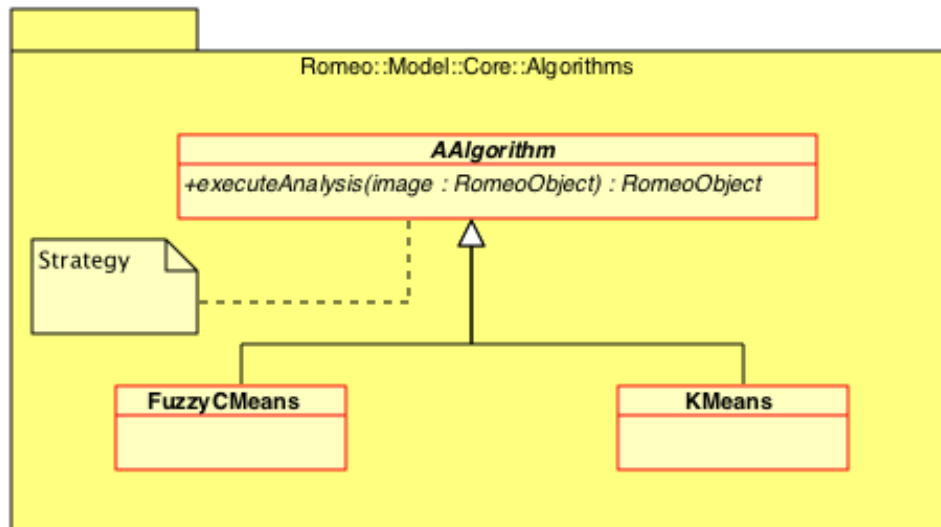


Figura 8: Diagramma package *Romeo::Model::Core::Algorithms*

4.5.2 Descrizione

Package_G contenente le classi che permettono al model di utilizzare gli algoritmi di clustering_G previsti nei requisiti. Le classi di questo package_G sono implementate utilizzando il design pattern_G Strategy.

4.5.3 Classi contenute

AAlgorithm

Descrizione: classe astratta che rappresenta un generico algoritmo di clustering_G, secondo il design pattern_G Strategy. Definisce dei contratti per l'esecuzione degli algoritmi, che dovranno essere implementati dalle sue sottoclassi. Rappresenta il componente Strategy dell'omonimo design pattern_G.

Utilizzo: fornisce i metodi per l'esecuzione di un algoritmo di clustering_G su un'immagine bidimensionale o tridimensionale.

Ereditata da:

- Romeo::Model::Core::Algorithms::FuzzyCMeans;
- Romeo::Model::Core::Algorithms::KMeans.

Relazioni con altre classi:

- Romeo::Model::Core::Protocol: relazione entrante riferimento all'algoritmo contenuto dal Protocol_G;
- Romeo::Model::Core::AlgCreator: relazione entrante, tipo statico di un algoritmo di cluster_G creato dalla classe Factory;
- Romeo::Model::Core::RomeoObject: relazione uscente, il metodo `executeAnalysis` lavora su un oggetto di tipo statico *RomeoObject*;

- `Romeo::Model::Core::RGBImage`: relazione uscente, la classe *AAlgorithm* utilizza il tipo `RGBImage` per la sua implementazione;
- `Romeo::Model::Core::RGBImage2D`: relazione uscente, la classe *AAlgorithm* utilizza il tipo *RGBImage2D* per eseguire un'analisi su un'immagine 2D;
- `Romeo::Model::Core::RGBImage3D`: relazione uscente, la classe *AAlgorithm* utilizza il tipo `RGBImage3D` per eseguire un'analisi su un'immagine 3D;
- `Romeo::Model::Core::Video`: relazione uscente, la classe *AAlgorithm* utilizza il tipo `VIDEO` per la sua implementazione;
- `Romeo::Model::Core::Video2D`: relazione uscente, la classe *AAlgorithm* utilizza il tipo *Video2D* per l'analisi su Video in formato 2D;
- `Romeo::Model::Core::Video3D`: relazione uscente, la classe *AALGORITHM* utilizza il tipo *Video3D* per l'analisi su Video in formato 3D;
- `Romeo::Model::Util::DAO::AlgorithmDAO`: relazione entrante, la classe *AlgorithmDAO* necessita del tipo *AAlgorithm* per la sua implementazione.

FuzzyCMeans

Descrizione: classe che implementa l'algoritmo di clustering_G FuzzyCMeans. Rappresenta il componente ConcreteStrategy del design pattern_G Strategy.

Utilizzo: viene utilizzata durante un'analisi per applicare l'algoritmo a un Dataset_G.

Eredita da:

- `Romeo::Model::Core::Algorithms::AAlgorithm`.

Relazioni con altre classi:

- `Romeo::Model::Core::AlgCreator`: relazione entrante, la classe *AlgCreator* necessita di vedere il tipo *FuzzyCMeans* per la sua implementazione.

KMeans

Descrizione: classe che implementa l'algoritmo di clustering_G K-Means. Rappresenta il componente ConcreteStrategy del design pattern_G Strategy.

Utilizzo: viene utilizzata durante un'analisi per applicare l'algoritmo a un Dataset_G.

Eredita da:

- `Romeo::Model::Core::Algorithms::AAlgorithm`.

Relazioni con altre classi:

- `Romeo::Model::Core::AlgCreator`: relazione entrante, la classe *AlgCreator* necessita di vedere il tipo `KMEANS` per la sua implementazione.

4.6 Romeo::Model::Util

4.6.1 Informazioni sul package

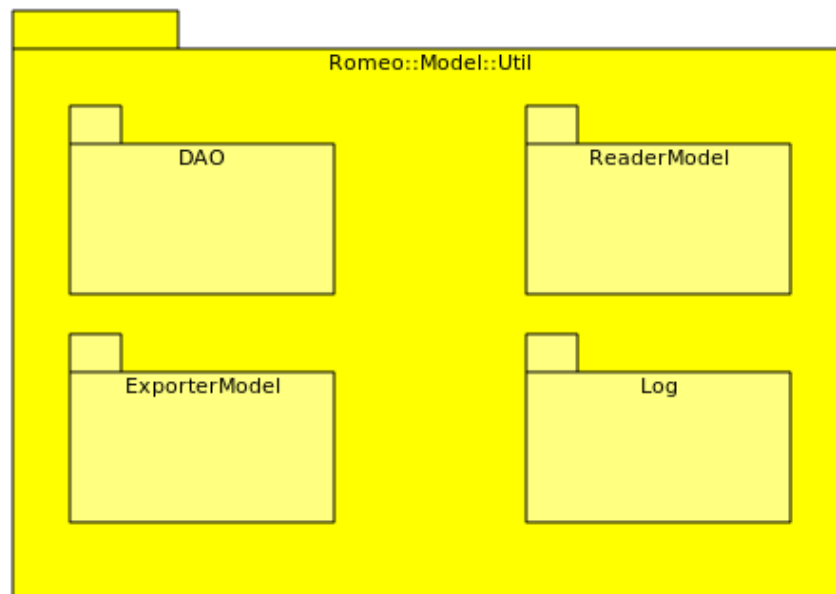


Figura 9: Diagramma package *Romeo::Model::Util*

4.6.2 Descrizione

L'obiettivo di questo package_G è quello di fornire una serie di classi di utilità e di supporto per le funzionalità core dell'applicativo.

4.6.3 Package contenuti

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

4.7 Romeo::Model::Util::DAO

4.7.1 Informazioni sul package

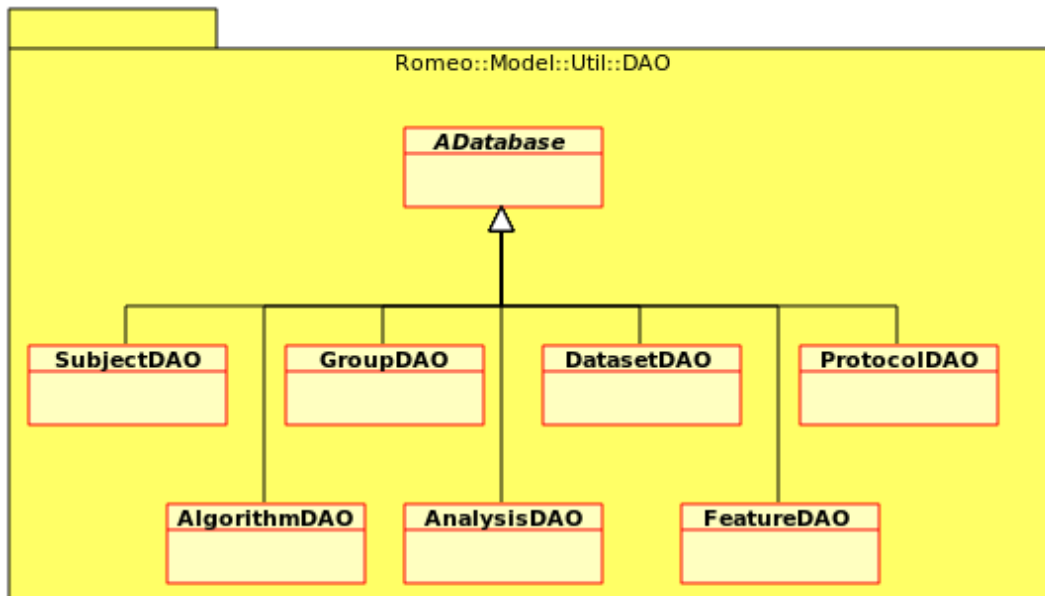


Figura 10: Diagramma package *Romeo::Model::Util::DAO*

4.7.2 Descrizione

Package_G che gestisce l'interfacciamento con il database del sistema⁴.

Per ogni tabella del database è stata creata una classe che estende la classe astratta *ADatabase*.

4.7.3 Classi contenute

ADatabase

Descrizione: classe astratta che fornisce i metodi di connessione e disconnessione al database e dei metodi per notificare eventuali errori dovuti al collegamento con componenti esterne, nel nostro caso, il database. Tale classe viene solamente estesa dalle classi che opereranno sulle tabelle del database.

Utilizzo: la classe viene creata alla creazione di una sua sottoclasse.

Eredita da:

- Qt::QSqlDatabase.

Ereditata da:

- Romeo::Model::Util::DAO::SubjectDAO;
- Romeo::Model::Util::DAO::GroupDAO;

⁴Per maggiori informazioni vedere la sezione 6

- `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.`

SubjectDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella `SubjectG` del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti i `SubjectG` creati dall'utente o utilizzati da `RomeoG`.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Model::Core::ASubject`: relazione uscente, la classe *SubjectDAO* necessita del tipo *ASubject* per la sua implementazione;
- `Romeo::Model::Core::ProxySubject`: relazione uscente, la classe *SubjectDAO* necessita del tipo *ProxySubject* per la sua implementazione;
- `Romeo::Controller::NewSubjectController`: relazione entrante, la classe *NewSubjectController* utilizza *SubjectDAO* per la creazione di nuovi `SubjectG` in `RomeoG`;
- `Romeo::Controller::SubjectsController`: relazione entrante, la classe *SubjectsController* utilizza *SubjectDAO* per ottenere la lista di `SubjectG` presenti nel sistema.

GroupDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella `GroupOfSubject` del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Gruppi di `Subject` creati dall'utente o utilizzati da `RomeoG`.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Model::Core::GroupOfSubject`: relazione uscente, la classe *SubjectDAO* necessita del tipo *GroupOfSubject* per la sua implementazione;
- `Romeo::Controller::NewGroupController`: relazione entrante, la classe *NewGroupController* utilizza *GroupOfSubject* per creare un nuovo gruppo di `SubjectG` in `RomeoG`;
- `Romeo::Controller::GroupsController`: relazione entrante, la classe *GroupsController* utilizza *GroupOfSubject* per visualizzare i gruppi di `SubjectG` esistenti.

ProtocolDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella Protocol del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Protocol_G creati dall'utente o utilizzati da Romeo_G.

Eredita da:

- Romeo::Model::Util::DAO::ADatabase.

Relazioni con altre classi:

- Romeo::Model::Core::Protocol: relazione uscente, la classe *ProtocolDAO* necessita del tipo *Protocol* per la sua implementazione;
- Romeo::Controller::NewProtocolController: relazione entrante, la classe *NewProtocolController* utilizza *Protocol* per creare nuovi Protocol_G in Romeo_G;
- Romeo::Controller::ProtocolsController: relazione entrante, la classe *ProtocolsController* utilizza *Protocol* per visualizzare i *Protocol* presenti in Romeo_G.

DatasetDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella Protocol del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti i Dataset_G creati dall'utente o utilizzati da Romeo_G.

Eredita da:

- Romeo::Model::Util::DAO::ADatabase.

Relazioni con altre classi:

- Romeo::Model::Core::Dataset: relazione uscente, la classe *DatasetDAO* necessita del tipo *Dataset* per la sua implementazione;
- Romeo::Controller::NewDatasetController: relazione entrante, la classe *NewDatasetController* utilizza *Dataset* per creare nuovi Dataset_G in Romeo_G;
- Romeo::Controller::DatasetsController: relazione entrante, la classe *DatasetsController* utilizza *Dataset* per visualizzare i *Dataset* presenti in Romeo_G.

AlgorithmDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella Algorithm del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti gli Algoritmi di Cluster_G creati dall'utente o utilizzati da Romeo_G.

Eredita da:

- Romeo::Model::Util::DAO::ADatabase.

Relazioni con altre classi:

- `Romeo::Model::Core::AAlgorithm`: relazione uscente, la classe *AlgorithmDAO* necessita del tipo *AAlgorithm* per la sua implementazione;
- `Romeo::Controller::NewProtocolController`: relazione entrante, la classe *NewProtocolController* utilizza *AlgorithmDAO* per aggiungere un algoritmo di cluster_G al *Protocol_G* che l'utente sta creando;
- `Romeo::Controller::ProtocolsController`: relazione entrante, la classe *ProtocolsController* utilizza *AlgorithmDAO* per ottenere gli algoritmi di cluster_G dei vari *Protocol_G* presenti nel sistema.

FeatureDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella Feature del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti le feature_G create dall'utente o utilizzate da Romeo_G.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Model::Core::AFeature`: relazione uscente, la classe *FeatureDAO* necessita del tipo *AAlgorithm* per la sua implementazione;
- `Romeo::Controller::NewProtocolController`: relazione entrante, la classe *NewProtocolController* utilizza *FeatureDAO* per aggiungere/eliminare le feature_G al *Protocol_G* che l'utente sta creando;
- `Romeo::Controller::ProtocolsController`: relazione entrante, la classe *ProtocolsController* utilizza *AlgorithmDAO* per ottenere le feature_G dei vari *Protocol_G* presenti nel sistema.

AnalysisDAO

Descrizione: classe che rappresenta l'oggetto incaricato di operare con la tabella Analysis del database.

Utilizzo: la classe verrà utilizzata quando si dovrà salvare nel database, o recuperare da esso informazioni riguardanti le feature_G create dall'utente o utilizzata da Romeo_G.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Model::Core::Analysis`: relazione uscente, la classe *AnalysisDAO* necessita del tipo *Analysis* per la sua implementazione;
- `Romeo::Controller::AnalysisController`: relazione entrante, la classe *AnalysisController* utilizza *AnalysisDAO* per avviare una nuova analisi in Romeo_G;
- `Romeo::Controller::ResultsController`: relazione entrante, la classe *ResultsController* utilizza *AnalysisDAO* per ottenere la lista di analisi effettuate;

- `Romeo::Controller::DetailedResultController`: relazione entrante, la classe *DetailedResultController* utilizza *ANALYSISDAO* per ottenere i dettagli di una specifica analisi effettuata.

4.8 Romeo::Model::Util::ExporterModel

4.8.1 Informazioni sul package

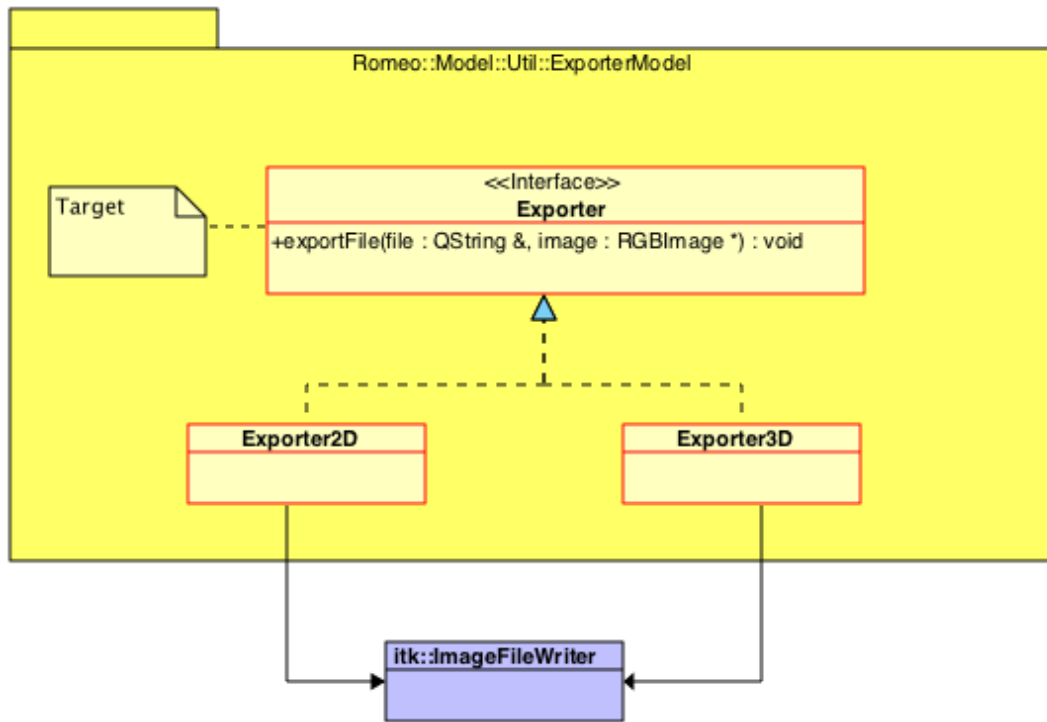


Figura 11: Package *Romeo::Model::Util::ExporterModel*

4.8.2 Descrizione

Package_G contenente le classi che si occupano di trasformare le immagini dal formato interno usato per l'analisi, al formato desiderato dall'utente, tra quelli previsti dai requisiti. Le classi di questo package_G sono implementate tramite il design pattern_G Adapter utilizzando le classi fornite dalla libreria esterna ITK_G.

4.8.3 Interfacce contenute

Exporter

Descrizione: interfaccia che fornisce un contratto per esportare i risultati delle analisi. Viene implementata dalle classi che specializzano l'esportazione per tipologia di file. Rappresenta il componente Target del design pattern_G Adapter.

Utilizzo: viene implementata dalle classi `Exporter2D` ed `Exporter3D` che ne implementano i metodi.

Implementata da:

- `Romeo::Model::Util::ExporterModel::Exporter2D`;
- `Romeo::Model::Util::ExporterModel::Exporter3D`.

Relazioni con altre classi:

- `Romeo::Model::Core::Analysis`: relazione entrante, la classe `Analysis` utilizza l'interfaccia `Exporter` per esportare i risultati durante l'analisi.
- `Romeo::Model::Core::RGBImage` relazione uscente, l'interfaccia utilizza la classe `RGBImage` quando deve esportare un'immagine.

4.8.4 Classi contenute

Exporter2D

Descrizione: 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 o dal `signalG` che la utilizza.

Utilizzo: la classe viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessita di esportare un'immagine di tipo 2D. Classe che viene utilizzata come adattatore della classe `ImageFileWriter`, fornita dalla libreria esterna `ITKG`. Rappresenta il componente `Adapter` del design pattern `Adapter`.

Eredita da:

- `Romeo::Model::Util::ExporterModel::Export`.

Relazioni con altre classi:

- `Romeo::Model::Core::Analysis`: relazione entrante, la classe `Analysis` utilizza la classe `AnalyzeExporter` per esportare i risultati dell'analisi in formato `AnalyzeG`.
- `Romeo::Model::Core::RGBImage` relazione uscente, l'interfaccia utilizza la classe `RGBImage` quando deve esportare un'immagine.

Exporter3D

Descrizione: classe che si occupa di esportare un'immagine di tipo `AnalyzeG`, nel percorso indicato dall'utente o dal `signalG` che la utilizza.

Utilizzo: la classe viene utilizzata in seguito alla ricezione di un `signalG` da parte dei controller che necessita di esportare un'immagine di tipo 3D. Classe che viene utilizzata come adattatore della classe `ImageFileWriter`, fornita dalla libreria esterna `ITKG`. Rappresenta il componente `Adapter` del design pattern `Adapter`.

Eredita da:

- `Romeo::Model::Util::ExporterModel::Export`.

Relazioni con altre classi:

- `Romeo::Model::Core::Analysis`: relazione entrante, la classe `Analysis` utilizza la classe `ImageExporter` per esportare i risultati dell'analisi in un formato immagine generico.
- `Romeo::Model::Core::RGBImage` relazione uscente, l'interfaccia utilizza la classe `RGBImage` quando deve esportare un'immagine.

4.9 Romeo::Model::Util::ReaderModel

4.9.1 Informazioni sul package

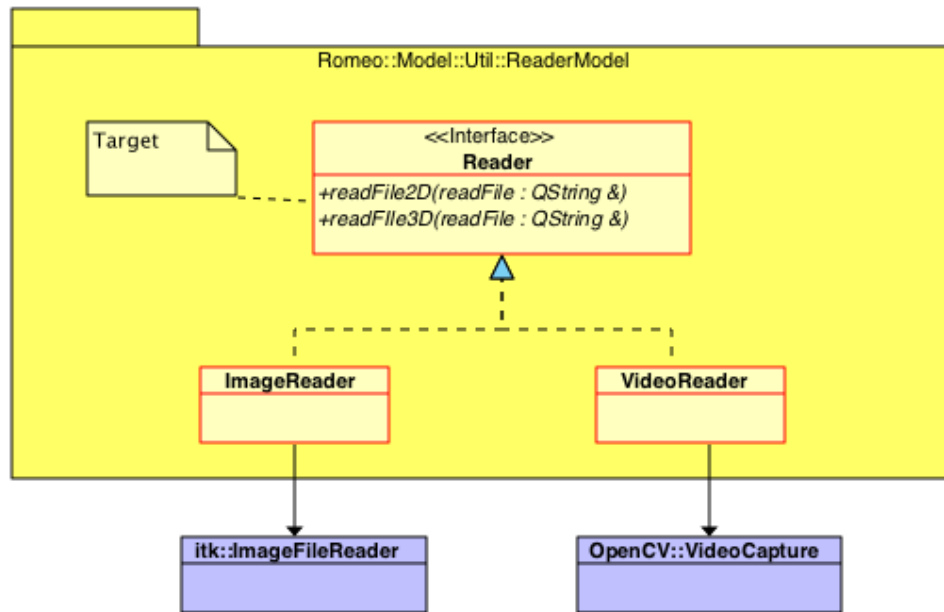


Figura 12: Pacakge `Romeo::Model::Util::ReaderModel`

4.9.2 Descrizione:

Package_G contenente le classi che si occupano di leggere le immagini e trasformarle nel formato interno usato all'interno di Romeo.

Le classi di questo package_G sono implementate tramite il design pattern_G Adapter adattando le classi fornite dalla libreria esterna ITK_G.

4.9.3 Interfacce contenute

Reader

Descrizione: interfaccia che fornisce un contratto per la lettura di immagini e la trasformazione di esse nel formato interno. Viene implementata dalle classi che specializzano la lettura per tipologia di file.

Rappresenta il componente Target del design pattern_G Adapter.

Utilizzo: viene implementata dalle classi `ImageReader` e `VideoReader` che ne implementano i metodi.

Implementata da:

- `Romeo::Model::Util::ReaderModel::ImageReader`;
- `Romeo::Model::Util::ReaderModel::VideoReader`.

Relazioni con altre classi:

- `Romeo::Model::Core::RealSubject`: relazione entrante, la classe `RealSubject` necessita dell'interfaccia `Reader` per la sua implementazione.

ImageReader

Descrizione: classe che si occupa di caricare immagini di tipo 2D e 3D non time dependent. Viene utilizzata come adattatore dalla classe `ImageFileReader` fornita dalla libreria esterna ITK_G.

Rappresenta il componente Adapter del design pattern_G Adapter.

Utilizzo: viene utilizzata quando c'è necessità di operare sull'immagine, quindi durante l'analisi e durante la creazione dei subject.

Eredita da:

- `Romeo::Model::Util::ReaderModel::Reader`.

Relazioni con altre classi:

- `Romeo::Model::Core::RGBImage`: relazione entrante, la classe `RGBImage` necessita della classe `ImageReader` per la sua implementazione;
- `Romeo::Model::Core::RBGImage2D`: relazione entrante, la classe `RBGImage2D` necessita della classe `ImageReader` per la sua implementazione.
- `Romeo::Model::Core::RGBImage3D`: relazione entrante, la classe `RGBImage3D` necessita della classe `ImageReader` per la sua implementazione.
- `Romeo::Model::Core::InternalData2D`: relazione entrante, la classe `InternalData2D` necessita della classe `ImageReader` per la sua implementazione.
- `Romeo::Model::Core::InternalData3D`: relazione entrante, la classe `InternalData3D` necessita della classe `ImageReader` per la sua implementazione.

VideoReader

Descrizione: classe che si occupa di caricare video di tipo 2D e 3D. Classe che viene utilizzata come adattatore della classe `VideoFileReader` fornita dalla libreria esterna ITK_G. Rappresenta il componente Adapter del design pattern_G Adapter.

Utilizzo: viene utilizzata quando c'è necessità di operare sui video, quindi durante l'analisi e durante la creazione dei subject.

Eredita da:

- `Romeo::Model::Util::ReaderModel::Reader`.

Relazioni con altre classi:

- `Romeo::Model::Core::Video`: relazione entrante, la classe `Video` necessita della classe `VideoReader` per la sua implementazione;
- `Romeo::Model::Core::Video2D`: relazione entrante, la classe `Video2D` necessita della classe `VideoReader` per la sua implementazione.
- `Romeo::Model::Core::Video3D`: relazione entrante, la classe `Video3D` necessita della classe `VideoReader` per la sua implementazione.

4.10 Romeo::Model::Util::Log

4.10.1 Informazioni sul package

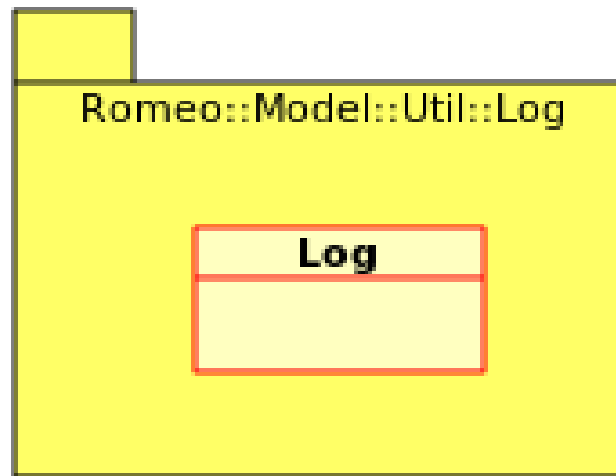


Figura 13: Diagramma package_G *Romeo::Model::Util::Log*

4.10.2 Descrizione

Package_G 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

Descrizione: 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.

4.11 Romeo::Model::qtModel

4.11.1 Informazioni sul package

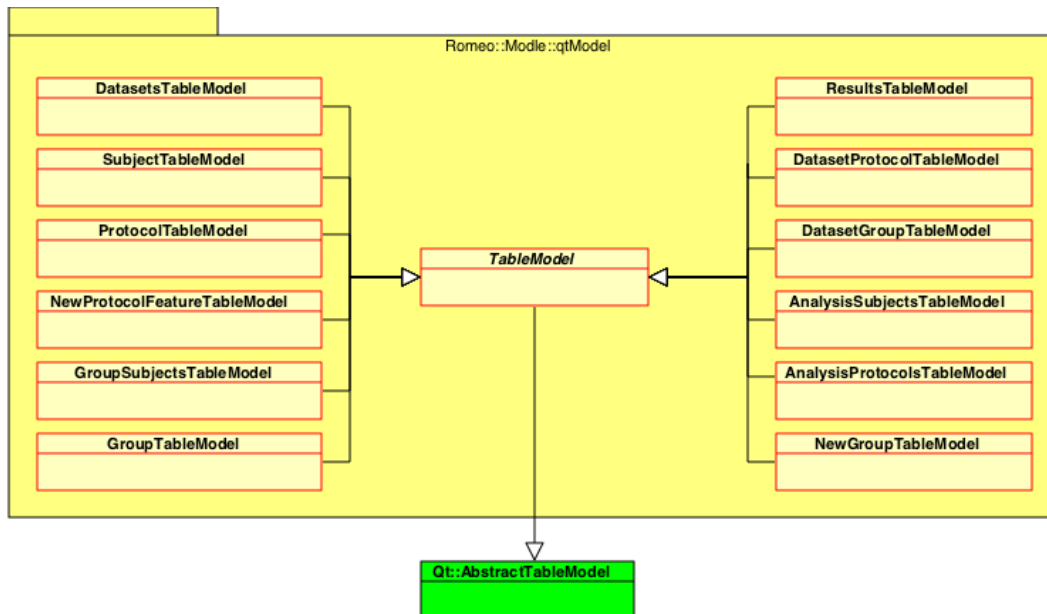


Figura 14: Diagramma package *Romeo::Model::qtModel*

4.11.2 Descrizione

Package_G contenente le classi che estendono i Model per le tabelle e liste proprietari di Qt_G, utilizzate delle *QTableView* e *QListView* presenti nelle view.

4.11.3 Classi contenute

TableModel

Descrizione: classe astratta che fornisce un contratto per la generazione del model Qt_G per le *QTableView*.

Eredita da:

- Qt::QAbstractTableModel.

Ereditata da:

- Romeo::Model::qtModel::DatasetsTableModel;
- Romeo::Model::qtModel::SubjectTableModel;
- Romeo::Model::qtModel::ProtocolTableModel;
- Romeo::Model::qtModel::NewProtocolFeatureTableModel;
- Romeo::Model::qtModel::GroupSubjectsTableModel;
- Romeo::Model::qtModel::GroupTableModel;
- Romeo::Model::qtModel::ResultsTableModel;
- Romeo::Model::qtModel::DatasetProtocolTableModel;

- `Romeo::Model::qtModel::DatasetGroupTableModel;`
- `Romeo::Model::qtModel::AnalysisSubjectsTableModel;`
- `Romeo::Model::qtModel::AnalysisProtocolsTableModel;`
- `Romeo::Model::qtModel::NewGroupTableModel.`

4.12 Romeo::Model::Help

4.12.1 Informazioni sul package

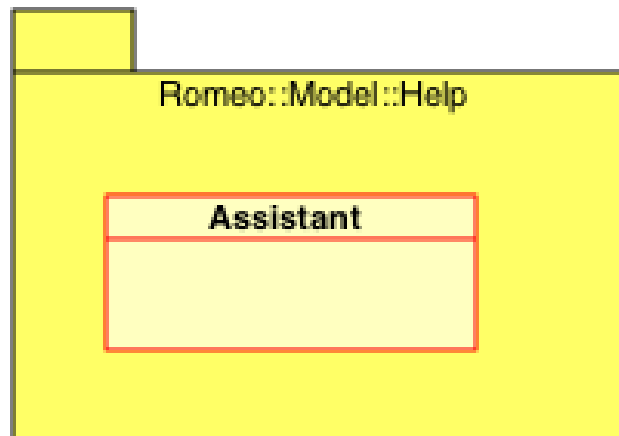


Figura 15: Diagramma package `Romeo::Model::Help`

4.12.2 Descrizione

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

4.12.3 Classi contenute

Assistant

Descrizione: classe che rappresenta il processo Qt Assistant, utilizzato per la gestione della guida utente.

Utilizzo: viene utilizzata per gestire la richiesta dell'utente, che vuole visualizzare una determinata pagina all'interno della guida utente.

Eredita da:

- `Qt::QProcess`

Relazioni con altre classi:

- `Romeo::Controller::MainWindowController`: relazione entrante, la classe `MainWindowController` utilizza la classe per avviare la guida utente.

4.13 Romeo::View

4.13.1 Informazioni sul package

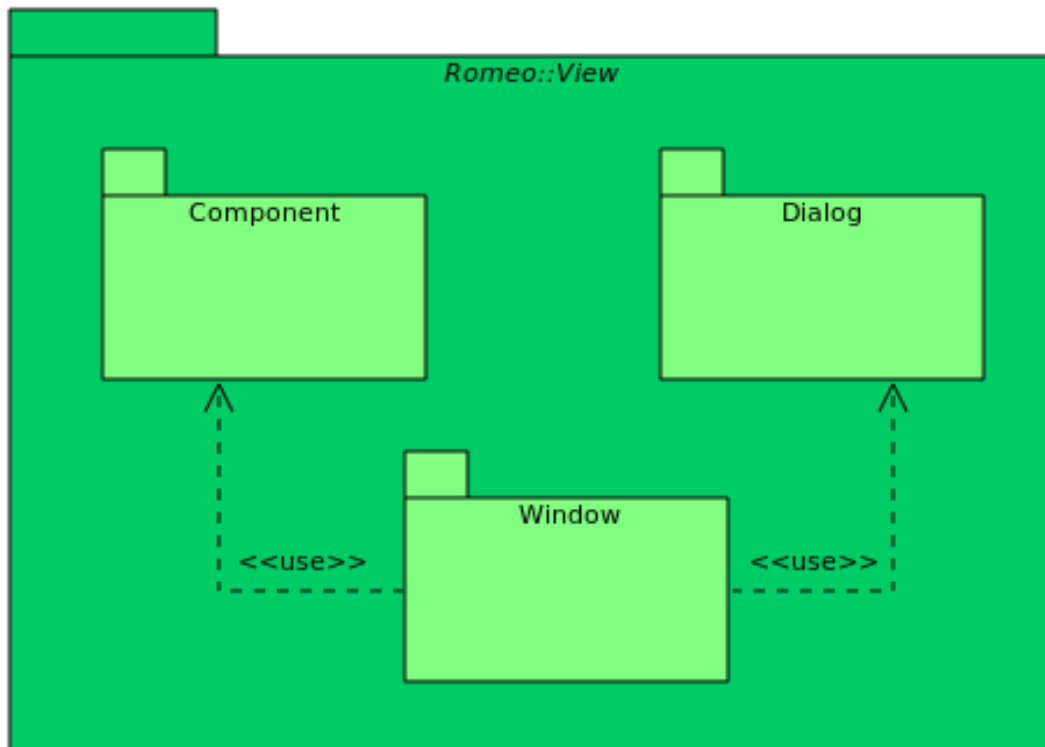


Figura 16: Componente Romeo::View

4.13.2 Descrizione

Package_G che rappresenta la componente View dell'architettura MVC_G.

4.13.3 Package contenuti

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

4.13.4 Relazioni tra i componenti

Il package_G Window utilizzerà vari componenti del package_G Component e del package_G Dialog per generare le varie finestre e dialoghi. Il diagramma seguente illustra le relazioni interne a Romeo::View tra le classi contenute nei vari package_G.

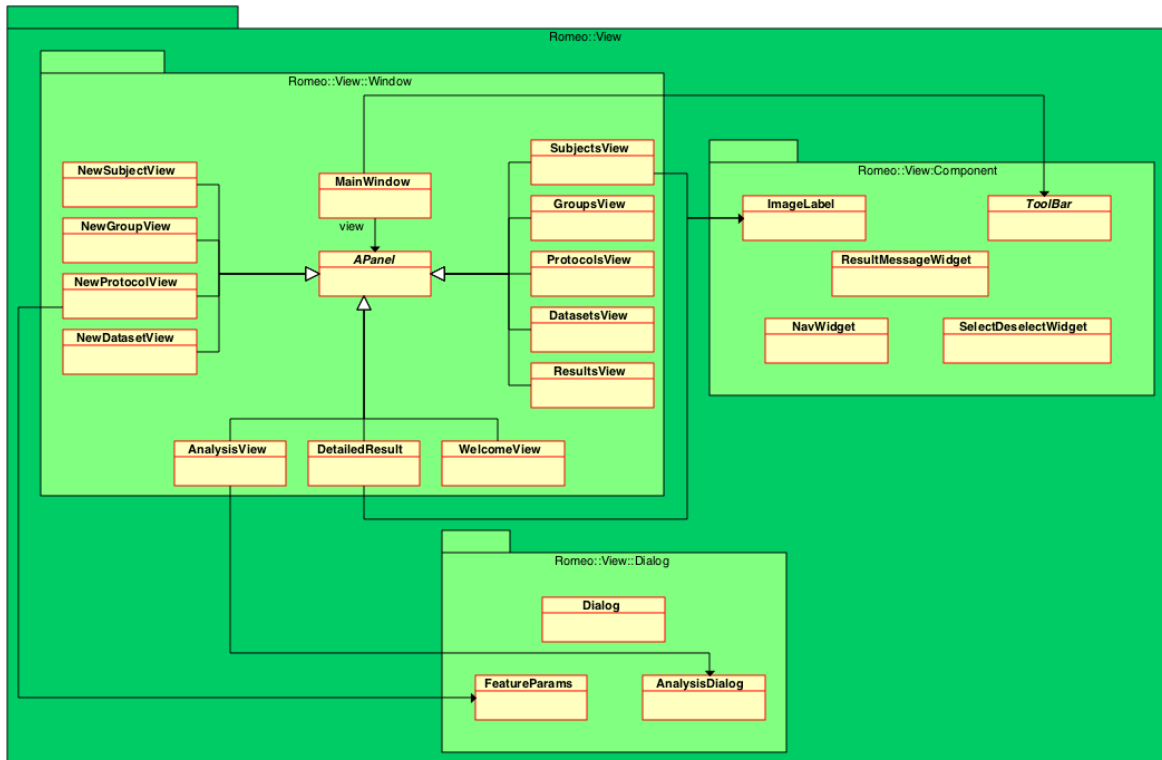


Figura 17: Relazioni tra le classi del package Romeo::View

4.14 Romeo::View::Window

4.14.1 Informazioni sul package

4.14.2 Descrizione

Package \mathcal{G} che contiene l'insieme delle “finestre” con le quali l'utente può interagire durante l'esecuzione di Romeo.

4.14.3 Relazioni tra i componenti

La classe MainWindow contiene un riferimento polimorfo alla classe \mathcal{A} , classe astratta che rappresenta una generica finestra del programma.

4.14.4 Classi contenute

MainWindow

Descrizione: classe che rappresenta la finestra principale dell'applicativo Romeo con la quale l'utente interagisce. Viene creata *unicamente* al primo avvio del programma e rimane attiva fino alla chiusura del programma. È implementata tramite il design pattern \mathcal{G} Singleton.

Eredita da:

- Qt::QMainWindow.

Relazioni con altre classi:

- Romeo::View::Window::APanel: relazione uscente, riferimento all'oggetto APanel che la MainWindow sta attualmente rappresentando;

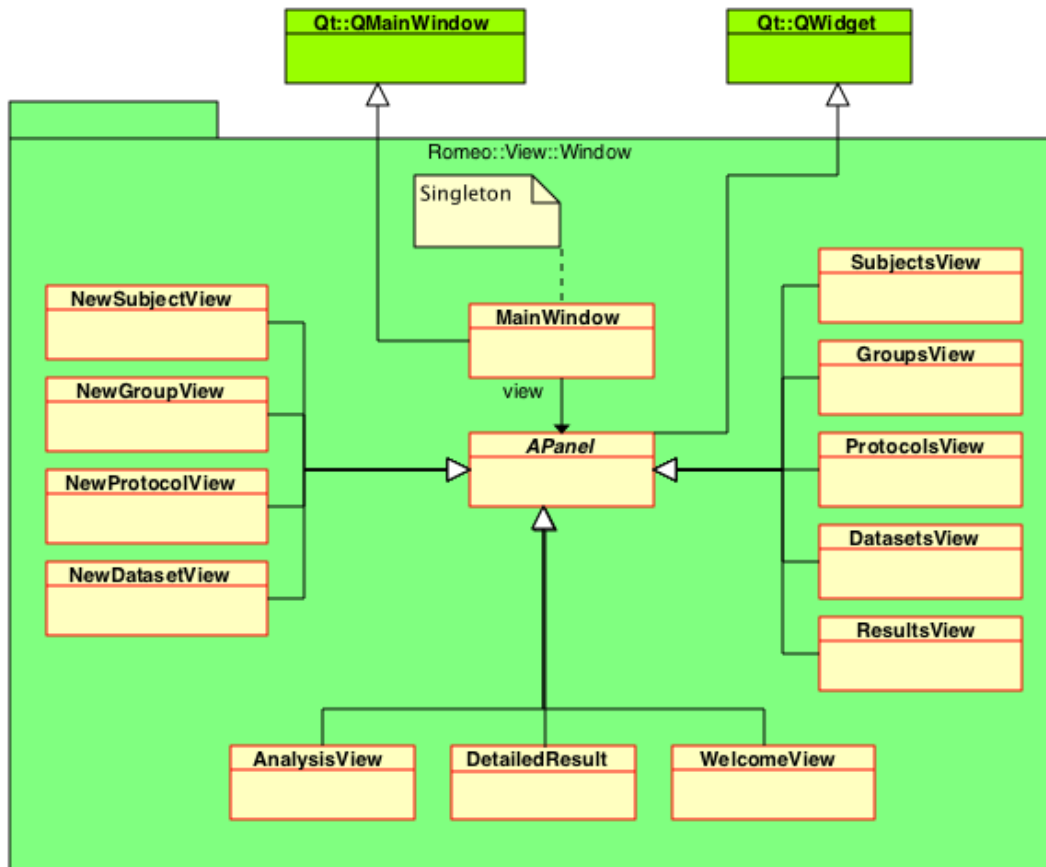


Figura 18: Diagramma package *Romeo::View::Window*

- *Romeo::View::Component::MenuBar*: relazione uscente, riferimento al menù con il quale interagisce l'utente;
- *Romeo::View::Component::ToolBar*: relazione uscente, riferimento alla toolbar con la quale interagisce l'utente;
- *Romeo::Controller::MainWindowController*: relazione entrante, riferimento alla *MainWindow* che il controller sta "controllando".

APanel

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

Eredita da:

- *Qt::QWidget*.

Ereditata da:

- *Romeo::View::Window::NewSubjectView*;
- *Romeo::View::Window::NewGroupView*;
- *Romeo::View::Window::NewProtocolView*;
- *Romeo::View::Window::NewDatasetView*;

- `Romeo::View::Window::AnalysisView;`
- `Romeo::View::Window::DetailedResult;`
- `Romeo::View::Window::WelcomeView;`
- `Romeo::View::Window::SubjectsView;`
- `Romeo::View::Window::GroupsView;`
- `Romeo::View::Window::ProtocolsView;`
- `Romeo::View::Window::DatasetsView;`
- `Romeo::View::Window::ResultsView.`

Relazioni con altre classi:

- `Romeo::Controller::AController:` relazione entrante, riferimento alla generica “view” che il controller sta “controllando”.

WelcomeView

Descrizione: classe che rappresenta la view iniziale del sistema. Permette all’utente di scegliere una tra le varie funzionalità del programma. Viene creata all’avvio del programma e permette all’utente di selezionare una delle funzionalità presenti. Inoltre viene creata quando da una delle view l’utente decide di ritornare alla `WelcomeView`. Emette un `signal_G` in seguito alla scelta effettuata.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Controller::WelcomeController:` relazione entrante, tipo dinamico del riferimento posseduto del Controller.

NewSubjectView

Descrizione: permette la creazione di un nuovo `Subject_G`. Viene creata ogni qualvolta l’utente seleziona, dalla `WelcomeView`, la funzionalità di creazione di un nuovo `Subject_G`. Emette un `signal_G` in seguito alla conferma, da parte dell’utente, della creazione di un nuovo `Subject_G`.

Eredita da:

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

Relazione con altre classi:

- `Romeo::Controller::NewSubjectController:` relazione entrante, tipo dinamico del riferimento posseduto dal Controller.

NewGroupView

Descrizione: 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.

Inoltre permette la modifica di un gruppo di `subjectG` preesistente. Emette un `signalG` in seguito alla conferma, da parte dell'utente, della creazione di un nuovo gruppo di `SubjectG`.

Eredita da:

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

Relazione con altre classi:

- `Romeo::Controller::NewGroupController`: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- `Romeo::Model::QtModel::NewGroupTableModel`: relazione uscente, riferimento al model della tabella visualizzata nella view.

NewProtocolView

Descrizione: 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 `signalG` in seguito alla conferma, da parte dell'utente, della creazione di un nuovo `ProtocolG`.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Controller::NewProtocolController`: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- `Romeo::Model::QtModel::NewProtocolFeatureTableModel`: relazione uscente, riferimento al model della tabella visualizzata nella view.

NewDatasetView

Descrizione: 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 `signalG` in seguito alla conferma, da parte dell'utente, della creazione di un nuovo gruppo di `DatasetG`.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::Controller::NewDatasetController`: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- `Romeo::Model::QtModel::DatasetGroupTableModel`: relazione uscente, riferimento al model della tabella presente nella view che visualizza i `Dataset`;
- `Romeo::Model::QtModel::DatasetProtocolTableModel`: relazione uscente, riferimento al model della tabella presente nella view che visualizza i `Protocol`.

SubjectsView

Descrizione: permette la visualizzazione dei Subject_G memorizzati nel sistema. Viene creata ogni qualvolta l'utente seleziona, dalla WelcomeView, l'opzione di visualizzazione della lista dei Subject_G. Essa comunica con il Controller per acquisire la lista dei Subject_G.

Eredita da:

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

Relazioni con altre classi:

- Romeo::Controller::SubjectsController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- Romeo::Model::QtModel::SubjectTableModel: relazione uscente, riferimento al model della tabella visualizzata nella view.

GroupsView

Descrizione: classe che rappresenta la view per la visualizzazione, l'eliminazione e la modifica dei vari gruppi di Subject_G presenti nel sistema. Viene creata ogni qualvolta l'utente seleziona, dalla WelcomeView, l'opzione di visualizzazione dei lista dei gruppi di Subject_G. Comunica con il relativo controller per ottenere la lista dei gruppi di Subject_G, modificare un gruppo (aggiungendo o togliendo Subject_G) e per eliminare eventuali gruppi selezionati.

Eredita da:

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

Relazioni con altre classi:

- Romeo::Controller::GroupsController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- Romeo::Model::QtModel::GroupTableModel: relazione uscente, riferimento al model della tabella presente nella view che visualizza i Group Of Subjects;
- Romeo::Model::QtModel::GroupSubjectsTableModel: relazione uscente, riferimento al model della tabella presente nella view che visualizza i Subject.

ProtocolsView

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

Eredita da:

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

Relazioni con altre classi:

- Romeo::Controller::ProtocolsController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- Romeo::Model::QtModel::ProtocolTableModel: relazione uscente, riferimento al model della tabella visualizzata nella view.

DatasetsView

Descrizione: classe che rappresenta la view per la visualizzazione e l'eliminazione dei vari Dataset_G esistenti. Viene creata ogni qualvolta l'utente seleziona, dalla WelcomeView, l'opzione di visualizzazione della lista dei Dataset_G presenti nel sistema. Comunica con il relativo controller per visualizzare i dettagli di un Dataset_G e per eliminare gli eventuali Dataset_G selezionati.

Eredita da:

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

Relazioni con altre classi:

- Romeo::Controller::DatasetsController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller.

AnalysisView

Descrizione: 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 Dataset_G sia presente nel sistema. Emette un signal_G in seguito all'interazione con l'utente, per l'avvio dell'analisi.

Eredita da:

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

Relazioni con altre classi:

- Romeo::Controller::AnalysisController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller;
- Romeo::Model::QtModel::AnalysisSubjectsTableModel: relazione uscente, riferimento al model della tabella presente nella view che visualizza i Subjects;
- Romeo::Model::QtModel::AnalysisProtocolsTableModel: relazione uscente, riferimento al model della tabella presente nella view che visualizza i Protocol.

ResultsView

Descrizione: 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.

Relazioni con altre classi:

- Romeo::Controller::ResultsController: relazione entrante, tipo dinamico del riferimento posseduto dal Controller.

DetailedResult

Descrizione: 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 nell'AnalysisView. Comunica inoltre con il relativo controller per acquisire la lista dei Subject_G e dei Protocol_G coinvolti nell'analisi e per avviare l'esportazione dei risultati.

Eredita da:

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

4.15 Romeo::View::Dialog

4.15.1 Informazioni sul Package

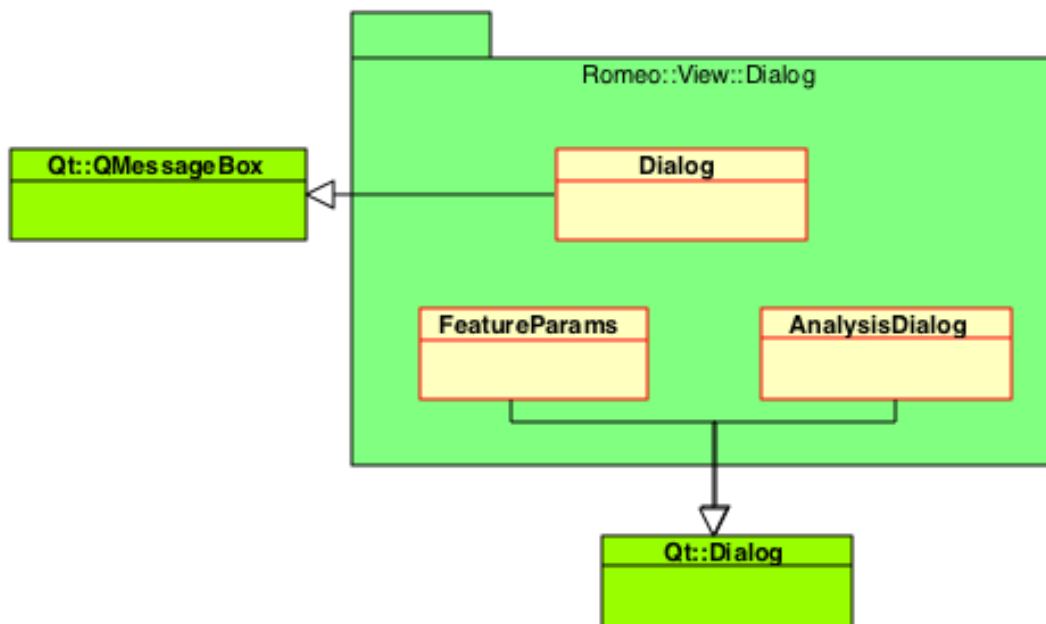


Figura 19: Componente Romeo::View::Dialog

4.15.2 Descrizione

Package_G che contiene l'insieme dei “dialoghi” con i quali l'utente può interagire durante l'esecuzione di Romeo.

4.15.3 Classi contenute

FeatureParams

Descrizione: classe che rappresenta il dialogo con il quale l'utente interagisce per selezionare la feature_G da aggiungere al protocol_G. Inoltre permette all'utente di settare i vari parametri.

Eredita da:

- Qt::QDialog;

Relazioni con altre classi:

- `Romeo::Controller::NewProtocolController`: relazione entrante, il controller si occupa di creare un'istanza di `FeatureParams` quando l'utente sceglie di aggiungere una nuova feature `G` ad un `ProtocolG`.

AnalysisDialog

Descrizione: 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 funziona di avvio analisi dalla finestra `AnalysisView`. Emette `signalG` verso il relativo controller, riguardo alle opzioni di visualizzazione dei risultati, oltre che per l'annullamento dell'analisi.

Eredita da:

- `Qt::QDialog`.

Relazioni con altre classi:

- `Romeo::Controller::AnalysisController`: relazione entrante, l'`AnalysisController` quando richiesto si occupa della sua creazione e del suo aggiornamento;

Dialog

Descrizione: classe che rappresenta i possibili messaggi che il sistema può inviare all'utente durante l'utilizzo di `Romeo`.

Eredita da:

- `Qt::QMessageBox`.

4.16 Romeo::View::Component

4.16.1 Informazioni sul package

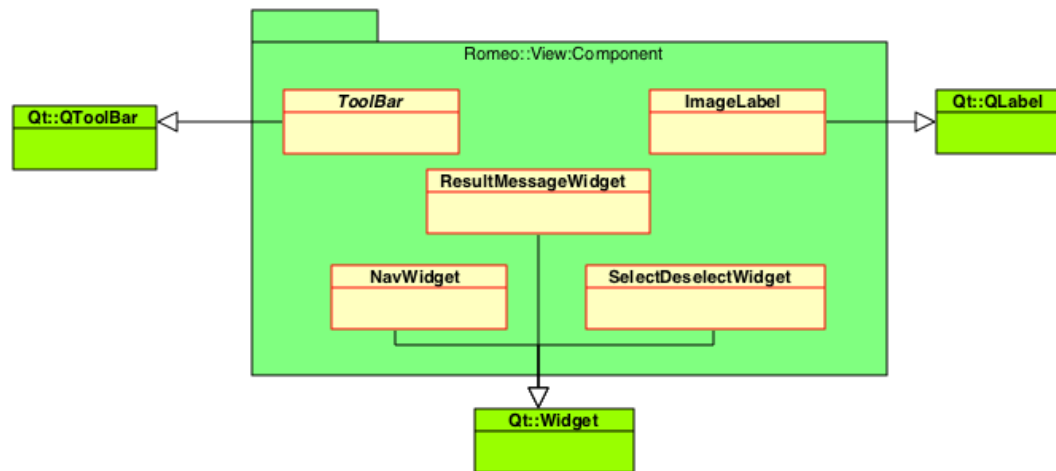


Figura 20: Componente Romeo::View::Window

4.16.2 Descrizione

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

ToolBar

Descrizione: classe che rappresenta una *toolbar* nella parte alta delle viste la quale permette all’utente di navigare all’interno del programma ed accedere a tutte le funzionalità. Viene visualizzata in tutte le viste tranne nella *WelcomeView*.

Eredita ta:

- `Qt::QToolBar`;

Relazioni con altre classi:

- `Romeo::View::Window::MainWindow`: relazione entrante, riferimento alla tool bar con la quale l’utente interagisce.

NavWidget

Descrizione: classe che rappresenta il widget costituisce la parte bassa delle view; contiene pulsanti utili, per esempio il pulsante *back*, *help*, *save* a seconda della view che riferisce un oggetto di questa classe.

Eredita da:

- `Qt::QWidget`.

SelectDeselectWidget classe che rappresenta il widget che da la possibilità all’utente di selezionare/deselezionare l’intero elenco di elementi.

In questo modo si permette all’utente di velocizzare alcune operazioni, invece di dover selezionare un elemento alla volta.

Eredita da:

- Qt::QWidget.

ImageLabel

Descrizione: classe che rappresenta le immagini presenti nelle viste *SubjectView* e *DetailedResults* con cui l'utente può interagire.

Eredita da:

- Qt::QLabel.

Relazioni con altre classi:

- Romeo::View::Window::SubjectView
- Romeo::View::Window::ImageResult

ResultMessageWidget

Descrizione: classe che rappresenta dei messaggi visivi per l'utente, di successo o di errore, in tutte le viste in base alle azioni che intraprende l'utente.

Eredita da:

- Qt::QWidget.

4.17 Romeo::Controller**4.17.1 Informazioni sul package****4.17.2 Descrizione**

Package_G che rappresenta la componente Controller dell'architettura MVC_G.

4.17.3 Relazioni tra i componenti

La classe ControllerManager contiene dei riferimenti ai vari controller attivi.

4.17.4 Classi contenute**MainWindowController**

Descrizione: classe che implementa il design pattern_G Singleton e gestisce tutti i Signal_G sulle voci della toolbar.

Eredita da:

- Qt::QObject.

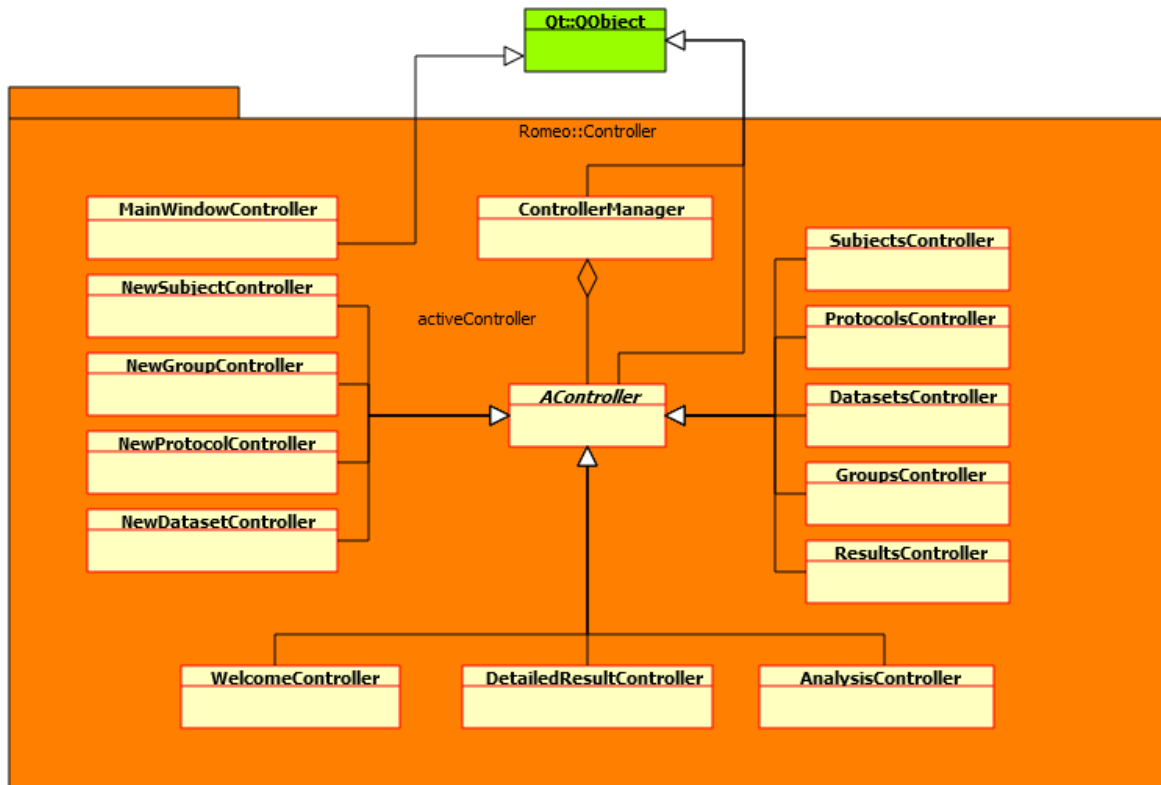


Figura 21: Componente Romeo::Controller

Relazioni con altre classi:

- Romeo::Controller::WelcomeController: relazione entrante, la classe WelcomeController per la sua implementazione necessita di chiamare slot presenti di MainWindowController;
- Romeo::View::Window::MainWindow: relazione uscente, riferimento ad un oggetto MainWindow.
- Romeo::View::Window::NewSubjectView: relazione uscente, il controller si occupa di creare un'istanza di NewSubjectView e di aggiornare il contenuto centrale mostrato dalla MainWindow;
- Romeo::Controller::NewSubjectController: relazione uscente, il controller si occupa di creare un'istanza di NewSubjectController per controllare l'istanza di NewSubjectView precedentemente creata;
- Romeo::View::Window::NewGroupView: relazione uscente, il controller si occupa di creare un'istanza di NewGroupView e di aggiornare il contenuto centrale mostrato dalla MainWindow;
- Romeo::Controller::NewGroupController: relazione uscente, il controller si occupa di creare un'istanza di NewGroupController per controllare l'istanza di NewGroupView precedentemente creata;
- Romeo::View::Window::NewProtocolView: relazione uscente, il controller si occupa di creare un'istanza di NewProtocolView e di aggiornare il contenuto centrale mostrato dalla MainWindow;
- Romeo::Controller::NewProtocolController: relazione uscente, il controller si occupa di creare un'istanza di NewProtocolController per controllare l'istanza di NewProtocolView precedentemente creata;

- `Romeo::View::Window::NewDatasetView`: relazione uscente, il controller si occupa di creare un'istanza di `NewDatasetView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::NewDatasetController`: relazione uscente, il controller si occupa di creare un'istanza di `NewDatasetController` per controllare l'istanza di `NewDatasetView` precedentemente creata;
- `Romeo::View::Window::SubjectsView`: relazione uscente, il controller si occupa di creare un'istanza di `SubjectsView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::SubjectsController`: relazione uscente, il controller si occupa di creare un'istanza di `SubjectsController` per controllare l'istanza di `SubjectsView` precedentemente creata;
- `Romeo::View::Window::GroupsView`: relazione uscente, il controller si occupa di creare un'istanza di `GroupsView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::GroupsController`: relazione uscente, il controller si occupa di creare un'istanza di `GroupsController` per controllare l'istanza di `GroupsView` precedentemente creata;
- `Romeo::View::Window::ProtocolsView`: relazione uscente, il controller si occupa di creare un'istanza di `ProtocolsView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::ProtocolsController`: relazione uscente, il controller si occupa di creare un'istanza di `ProtocolsController` per controllare l'istanza di `ProtocolsView` precedentemente creata;
- `Romeo::View::Window::DatasetsView`: relazione uscente, il controller si occupa di creare un'istanza di `DatasetsView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::DatasetsController`: relazione uscente, il controller si occupa di creare un'istanza di `DatasetsController` per controllare l'istanza di `DatasetsView` precedentemente creata;
- `Romeo::View::Window::AnalysisView`: relazione uscente, il controller si occupa di creare un'istanza di `AnalysisView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::DatasetsController`: relazione uscente, il controller si occupa di creare un'istanza di `AnalysisController` per controllare l'istanza di `AnalysisView` precedentemente creata;
- `Romeo::View::Window::ResultsView`: relazione uscente, il controller si occupa di creare un'istanza di `ResultsView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::ResultsController`: relazione uscente, il controller si occupa di creare un'istanza di `ResultsController` per controllare l'istanza di `ResultsView` precedentemente creata;
- `Romeo::View::Window::WelcomeView`: relazione uscente, il controller si occupa di creare un'istanza di `WelcomeView` e di aggiornare il contenuto centrale mostrato dalla `MainWindow`;
- `Romeo::Controller::WelcomeController`: relazione uscente, il controller si occupa di creare un'istanza di `WelcomeController` per controllare l'istanza di `WelcomeView` precedentemente creata;

ControllerManager

Descrizione: classe che implementa il design pattern **G** 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& Slot** **G** delle **Qt**.

Eredita da:

- **Qt::QObject**.

Relazioni con altre classi:

- **Romeo::Controller::AController**: relazione uscente, riferimento all'oggetto **AController** attualmente presente in memoria.

AController

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

Eredita da:

- **Qt::QObject**.

Relazione con altre classi:

- **Romeo::View::Window::APanel**: relazione uscente, riferimento al generico oggetto **APanel** che il controller sta "controllando".

Ereditato da:

- **Romeo::Controller::NewSubjectController**;
- **Romeo::Controller::NewGroupController**;
- **Romeo::Controller::NewProtocolController**;
- **Romeo::Controller::NewDatasetController**;
- **Romeo::Controller::SubjectsController**;
- **Romeo::Controller::ProtocolsController**;
- **Romeo::Controller::DatasetsController**;
- **Romeo::Controller::GroupsController**;
- **Romeo::Controller::WelcomeController**;
- **Romeo::Controller::AnalysisController**;
- **Romeo::Controller::ResultsController**;
- **Romeo::Controller::AnalysisController**.

NewSubjectController

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

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::NewSubjectView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

NewGroupController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo gruppo di Subject_G. Viene utilizzata per gestire i Signal_G emessi dalla View, riguardanti la creazione di un gruppo di Subject_G e per invocare i relativi metodi del Model.

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::NewGroupView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

NewProtocolController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo Protocol_G. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un Protocol_G e per invocare i relativi metodi del Model.

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::NewProtocolView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

NewDatasetController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la creazione di un nuovo Dataset_G. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la creazione di un nuovo Dataset_G e per invocare i relativi metodi del Model.

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::NewDatasetView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

SubjectsController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei Subject_G presenti nel sistema. Viene utilizzata per gestire i Signal_G emessi dalla View, riguardanti la visualizzazione dei dettagli del Subject_G selezionato, invocando i relativi metodi del Model.

Eredita da:

- Romeo::Controller::AController

Relazioni con altre classi:

- Romeo::View::Window::SubjectsView: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

GroupsController

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

Classi ereditate:

- Romeo::Controller::AController

Relazioni con altre classi:

- Romeo::View::Window::GroupsView: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

ProtocolsController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei Protocol_G presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione ed eliminazione dei Protocol_G facenti parte del sistema. Inoltre gestisce i signal per la visualizzazione delle informazioni sulle Feature_G ed algoritmi di clustering_G che formano il Protocol_G selezionato.

Eredita da:

- Romeo::Controller::Acontroller

Relazioni con altre classi:

- Romeo::View::Window::ProcolsView: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

DatasetsController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione dei Dataset_G presenti nel sistema. Viene utilizzata per gestire i Signal emessi dalla View, riguardanti la visualizzazione ed eliminazione dei Dataset_G 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**

Relazioni con altre classi:

- **Romeo::View::Window::DatasetsView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

WelcomeController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante la visualizzazione pagina iniziale. Viene utilizzata per gestire i **SignalG** emessi dalla View, riguardanti la visualizzazione della pagina selezionata, tramite dei pulsanti presenti nella **WelcomeView**.

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::WelcomeView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller.

AnalysisController

Descrizione: classe che rappresenta il controller che si occupa di gestire l'interazione con l'utente durante l'avvio e l'esecuzione di un'analisi. Viene utilizzata per gestire i **SignalG** emessi dalla View, riguardanti l'avvio dell'analisi e l'esecuzione dell'analisi.

Eredita da:

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

Relazioni con altre classi:

- **Romeo::View::Window::AnalysisView**: relazione uscente, tipo dinamico del riferimento ad un oggetto APanel posseduto dal controller;
- **Romeo::View::Dialog::ExecuteAnalysisView**: relazione uscente, il controller si occupa di creare un oggetto **ExecuteAnalysisView** e di visualizzarlo.

ResultsController

Descrizione: 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 **SignalG** emessi dalla View dei risultati di una particolare analisi.

Eredita da:

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

Relazioni con altre classi:

- `Romeo::View::Window::ResultsView`: relazione uscente, tipo dinamico del riferimento ad un oggetto `APanel` posseduto dal controller;
- `Romeo::View::Window::DetailedResultView`: relazione uscente, il controller si occupa di creare un oggetto `DetailedResultView` e di visualizzarlo come contenuto centrale della `MainWindow`;
- `Romeo::View::Window::MainWindow`: relazione uscente, il controller necessita dell'oggetto `MainWindow` per aggiornare il contenuto da essa visualizzato.

5 Design pattern

5.1 Design pattern architetturali

5.1.1 MVC

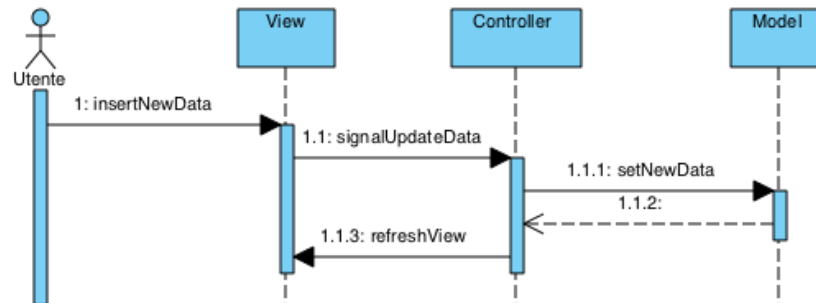


Figura 22: Diagramma di attività del pattern MVC

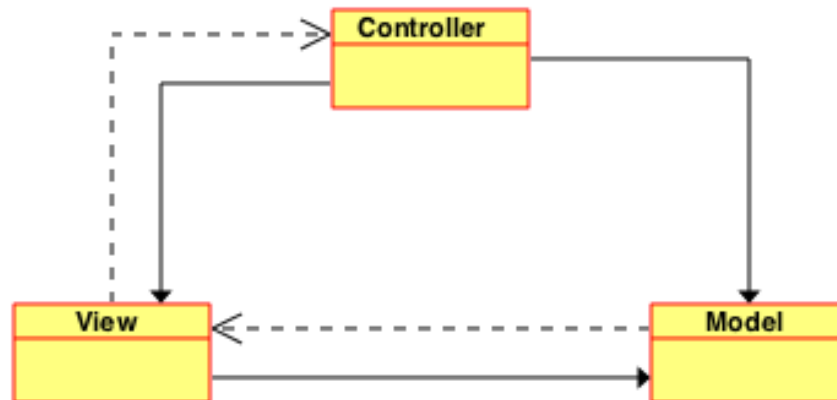


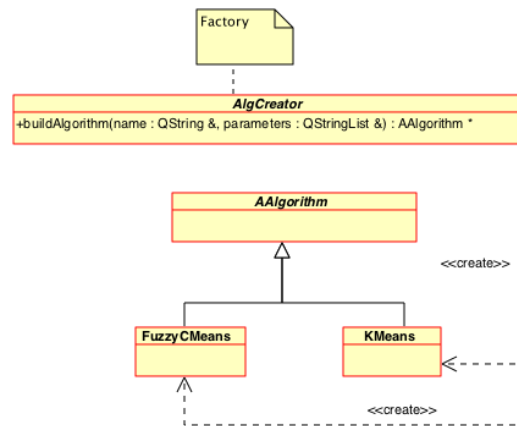
Figura 23: Diagramma delle classi del pattern MVC

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.

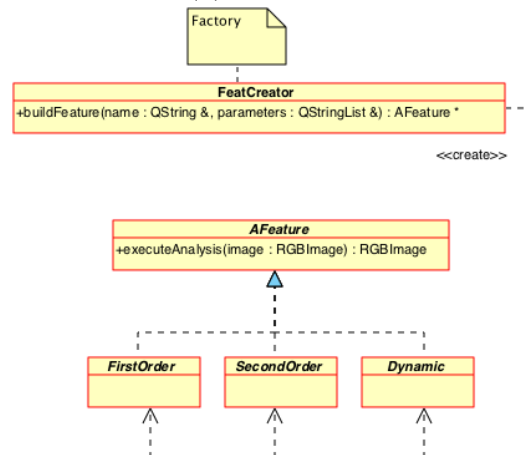
Contesto: il pattern MVC viene utilizzato per l'architettura generale dell'applicazione. Viene utilizzato il sistema Signal e Slot di Qt descritto nella sezione 2.2.1 per far comunicare i componenti tra loro. Ogni modifica fatta sulla View da parte dell'utente, viene inviata al Controller che da un comando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

5.2 Design pattern creazionali

5.2.1 Factory



(a) AlgCreator



(b) FeatCreator

Figura 24: Utilizzo di Factory in Romeo

Scopo dell'utilizzo: viene utilizzato il design pattern **Factory** quando una classe non è a conoscenza di che tipo di oggetto sia necessario istanziare. Le classi Factory hanno il compito di provvedere alla creazione di oggetti.

La scelta delle Concrete Product da creare verrà fatta in base ai parametri passati al metodo di creazione della classi Factory.

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::Features::AFeatures** (Product);
- **Romeo::Model::Core::AlgCreator** è Factory per tutte le classi (Concrete Product) che ereditano da **Romeo::Model::Core::Algorithms::AAlgorithm** (Product).

5.2.2 Singleton

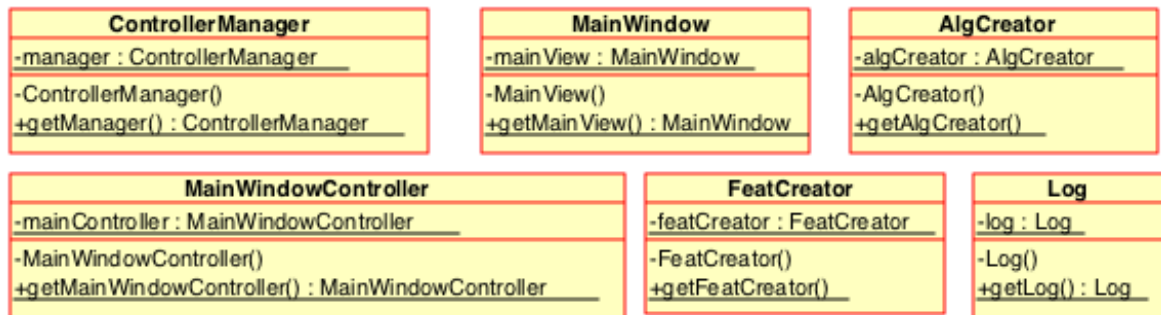


Figura 25: Utilizzo di Singleton in Romeo

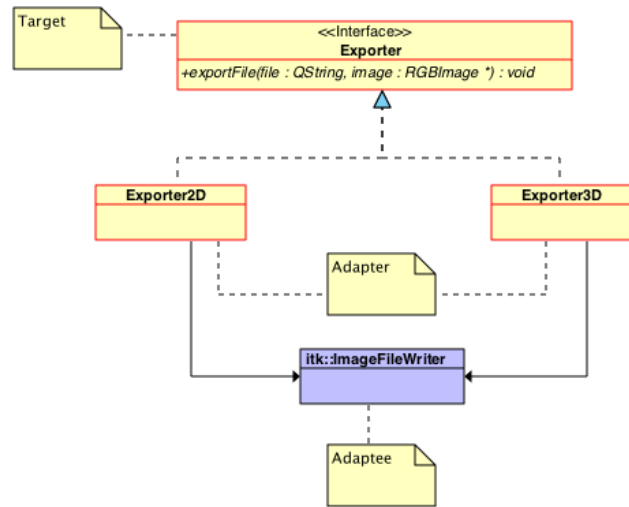
Scopo dell'utilizzo: viene utilizzato il design pattern Singleton per garantire che durante l'esecuzione dell'applicazione esista una sola istanza della classe in questione.

Contesto: le classi che necessitano di un'unica istanza sono:

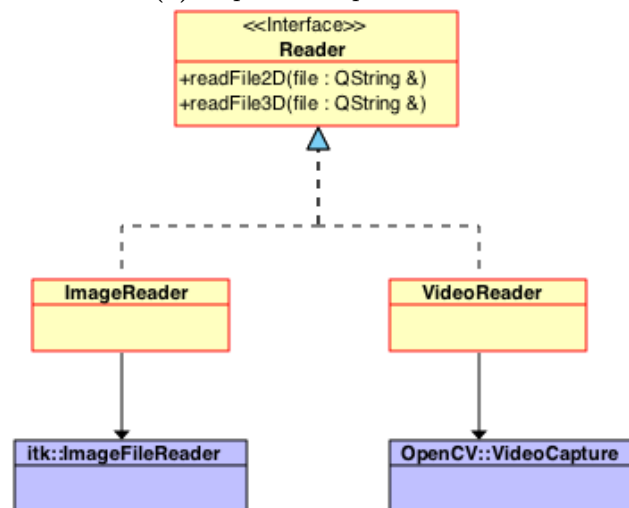
- `Romeo::Controller::ControllerManager;`
- `Romeo::View::Window::MainWindow;`
- `Romeo::Controller::MainWindowController;`
- `Romeo::Model::FacadeModel;`
- `Romeo::Model::Core::AlgCreator;`
- `Romeo::Model::Core::FeatCreator;`

5.3 Design pattern strutturali

5.3.1 Adapter



(a) ExporterAdapter



(b) ReaderAdapter

Figura 26: Utilizzo Adapter in Romeo

Scopo dell'utilizzo: il design pattern **G** Adapter (Object Adapter) viene utilizzato per convertire l'interfaccia di una classe in un'altra interfaccia richiesta dal client. In questo modo viene utilizzata un interfaccia stabile all'interno dell'applicativo. Nel nostro caso vengono adattate delle classi fornite dalla libreria esterna ITK **G**.

Contesto: le classi che utilizzano tale pattern sono:

- `Romeo::Model::Util::ExporterModel::Exporter2D` adatta la classe `itk::ImageFileWriter`;
- `Romeo::Model::Util::ExporterModel::Exporter3D` adatta la classe `itk::ImageFileWriter`;
- `Romeo::Model::Util::ReaderModel::ImageReader` adatta la classe `itk::VideoFileReader`;

- `Romeo::Model::Util::ReaderModel::VideoReader` adatta la classe `OpenCV::VideoCapture`;
- `Romeo::Model::Core::InternalData2D` adatta la classe `itk::Image`;
- `Romeo::Model::Core::InternalData2D` adatta la classe `itk::image`.

5.3.2 DAO (Data Access Object)

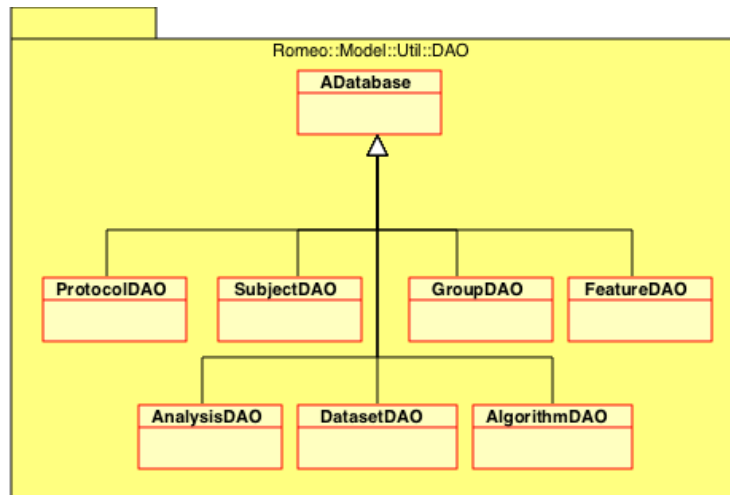


Figura 27: Utilizzo di DAO in Romeo

Scopo dell'utilizzo: viene utilizzato il design pattern DAO per permettere l'interfaciamento 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`.

5.3.3 Proxy

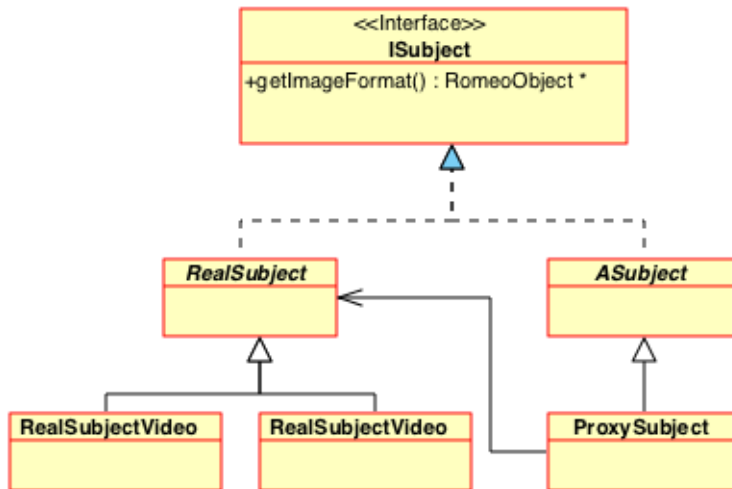


Figura 28: Utilizzo di Proxy in Romeo

Scopo dell'utilizzo: viene utilizzato il design pattern_G per fornire un surrogato di un altro oggetto e per controllare l'accesso a tale oggetto. Tramite l'utilizzo di tale design pattern_G è possibile rimandare la creazione di oggetti costosi solo quando essi sono necessari. In Romeo non viene creato un oggetto RealSubject fino a chè non è necessario.

Contesto: le classi che utilizzano tale pattern sono:

- `Romeo::Model::Core::ProxySubject;`

5.4 Design pattern comportamentali

5.4.1 Strategy

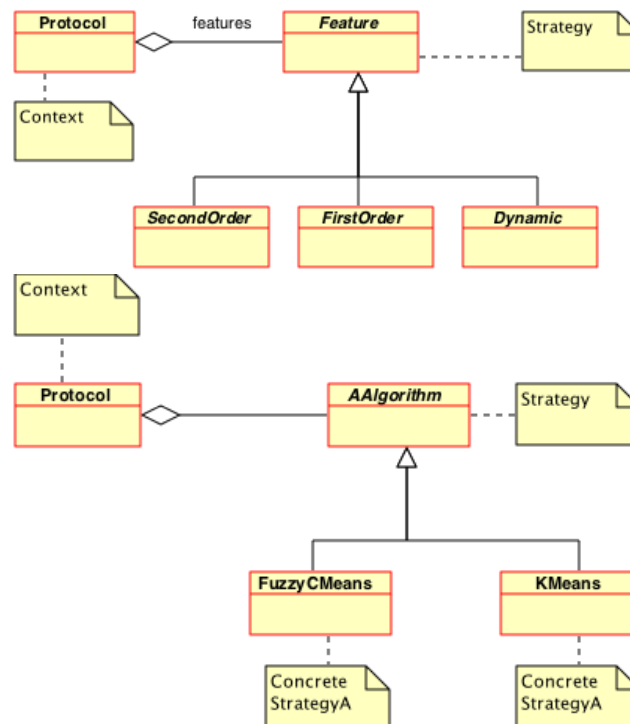


Figura 29: Utilizzo di Strategy in Romeo

Scopo dell'utilizzo: viene utilizzato il design pattern Strategy per isolare algoritmi e renderli intercambiabili tra di loro. Tale pattern viene utilizzato per incapsulare gli algoritmi di clustering e le feature previste dai requisiti. Si noti che per non appesantire troppo il diagramma non sono state riportate le Strategy concrete riguardanti le Feature.

Contesto: le classi che utilizzano tale pattern sono:

- `Romeo::Model::Core::Algorithms::FuzzyCMeans;`
- `Romeo::Model::Core::Algorithms::KMeans;`
- `Romeo::Model::Core::Features::StdDeviationFeature;`
- `Romeo::Model::Core::Features::SkewnessFeature;`
- `Romeo::Model::Core::Features::KurtosisFeature;`
- `Romeo::Model::Core::Features::MeanFeature;`
- `Romeo::Model::Core::Features::CorrelationFeature;`
- `Romeo::Model::Core::Features::EnergyFeature;`
- `Romeo::Model::Core::Features::HomogeneityFeature;`
- `Romeo::Model::Core::Features::EntropyFeature;`
- `Romeo::Model::Core::Features::MeanDFeature;`
- `Romeo::Model::Core::Features::MaximumFeature;`
- `Romeo::Model::Core::Features::MinimumFeature.`

6 Database Romeo

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

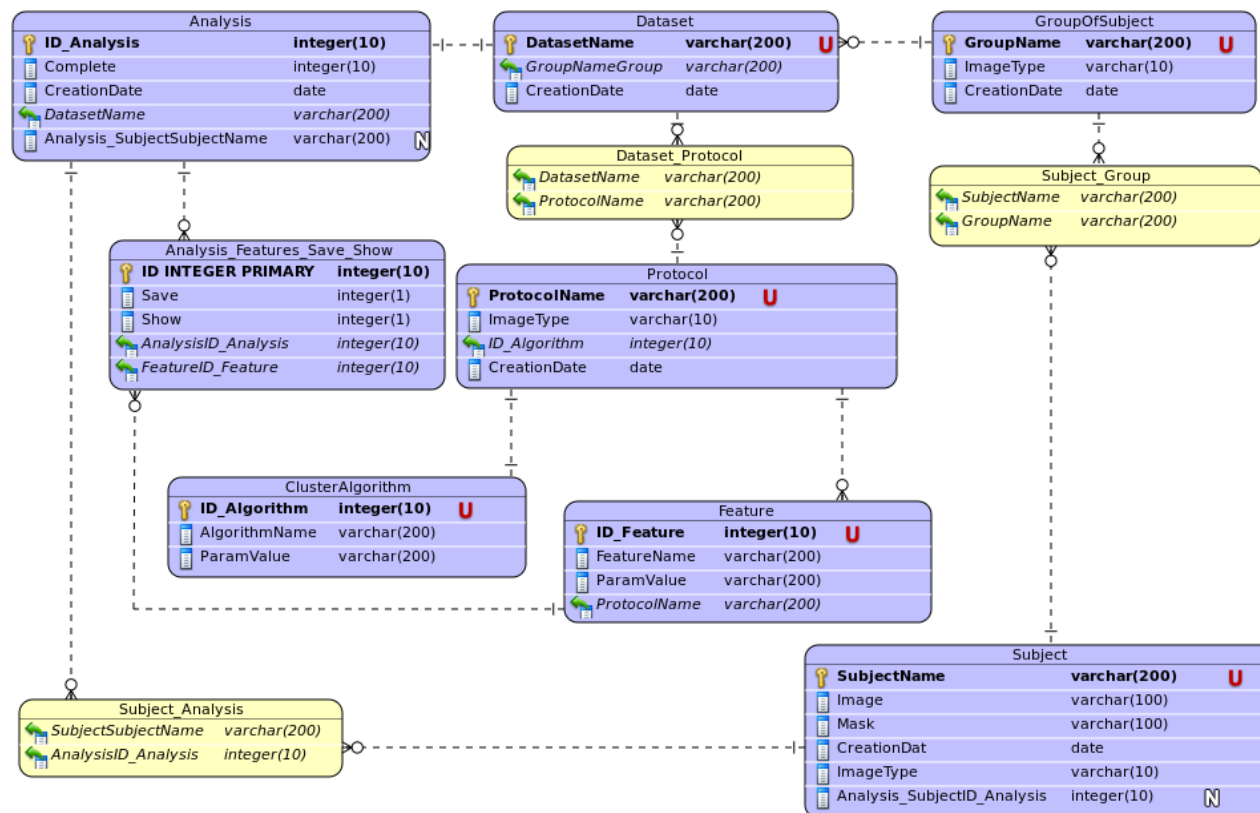


Figura 30: Struttura del database di Romeo

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.

6.1 Descrizione testuale delle tabelle

6.1.1 Subject

La tabella *Subject* raccoglie tutte le informazioni necessarie per identificare un *Subject_G*.

Attributi:

- **SubjectName:** *varchar(200)* «PK» nome univoco, identificativo di un *Subject_G*;
- **Image:** *varchar(200)* memorizza il percorso del filesystem dove è presente l'immagine associata al *Subject_G*;
- **Mask:** *varchar(200)* memorizza il percorso del filesystem dove è presente la maschera associata al *Subject_G*;
- **ImageType:** *varchar(10)* memorizza il tipo del *Subject_G*;
- **CreationDate:** *date* memorizza la data e l'ora di creazione del *Subject_G* all'interno del programma.

6.1.2 GroupOfSubject

La tabella *GroupOfSubject* raccoglie tutte le informazioni necessarie per un gruppo di *Subject_G*.

Attributi:

- **GroupName:** *varchar(200)* «PK» nome univoco, identificativo di un gruppo di *Subject_G*;
- **ImageType:** *varchar(10)* rappresenta il tipo di immagine che dovranno avere i *Subject_G* contenuti all'interno del gruppo;
- **CreationDate:** *date* memorizza la data e l'ora di creazione del gruppo di *Subject_G* all'interno del software.

6.1.3 Dataset

La tabella *Dataset* raccoglie tutte le informazioni necessarie per un *Dataset_G*.

Attributi:

- **DatasetName:** *varchar(200)* «PK» nome univoco, identificativo di un *Dataset_G*;
- **CreationDate:** *date* memorizza la data e l'ora di creazione del *Dataset_G* all'interno del software.

6.1.4 Protocol

La tabella *Protocol* raccoglie tutte le informazioni necessarie per un *Protocol_G*.

Attributi:

- **ProtocolName:** *varchar(200)* «PK» nome univoco, identificativo di Protocol_G;
- **ImageType:** *varchar(10)* rappresenta il tipo di immagine a cui il Protocol_G verrà applicato;
- **CreationDate:** *date* memorizza la data e l'ora di creazione del Protocol_G all'interno del programma.

6.1.5 ClusterAlgorithm

La tabella *ClusterAlgorithm* contiene tutte le istanze degli algoritmi di clustering_G che l'utente ha creato.

Attributi:

- **ID_Algorithm:** *Integer(10)* «PK» codice univoco, identificativo di una particolare istanza di algoritmo di clustering_G;
- **AlgorithmName:** *varchar(200)* nome dell'algoritmo di clustering_G;
- **ParamValue:** *varchar(200)* elenco dei valori dei parametri separati da punto.

Si noti che, per istanza di algoritmi di clustering_G, si intende un algoritmo di clustering_G istanziato con determinati valori dei parametri, definiti dall'utente.

6.1.6 Feature

La tabella *Feature* contiene tutte le istanze delle feature_G che l'utente ha creato.

Attributi:

- **ID_Feature:** *Integer(10)* «PK» codice univoco, identificativo di una particolare istanza di feature_G;
- **FeatureName:** *varchar(200)* nome della feature_G;
- **ParamValue:** *varchar(200)* elenco dei valori dei parametri separati da punto.

Si noti che, per istanza di feature_G, si intende una feature_G istanziata con determinati valori dei parametri, definiti dall'utente.

6.1.7 Analysis

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

Attributi:

- **ID_Analysis:** *Integer(10)* «PK» codice univoco, identificativo di un'analisi;
- **Complete:** *Integer(1)* se vale "1", l'analisi è stata portata a termine, altrimenti vale "0" e indica che l'analisi è stata interrotta;
- **CreationDate:** *date* memorizza la data e l'ora di dell'ultima analisi eseguita dal programma.

6.1.8 Analysis_Features_Save_Show

La tabella *Analysis_Features_Save_Show* contiene le informazioni relative a quali feature_G dell'analisi devono essere esportate e visualizzate durante l'analisi.

Attributi:

- **ID:** *Integer(10)* «PK» codice identificativo univoco;
- **Save:** *Integer(1)* se vale “1”, il risultato della feature_G viene esportato durante l’analisi, altrimenti se vale “0” il risultato della feature_G non viene esportato durante l’analisi;
- **Show:** *Integer(1)* se vale “1”, il risultato della feature_G viene visualizzato durante l’analisi, altrimenti se vale “0” il risultato della feature_G non viene visualizzato durante l’analisi.

6.2 Descrizione delle associazioni

- **“contain”:** associazione tra *GroupOfSubject* e *Subject*. Un gruppo può contenere uno o più Subject_G ed un Subject_G può appartenere o meno a più gruppi;
- **“hasA”:** associazione tra *Dataset* e *Protocol*. Un Dataset_G include uno o più Protocol_G ed un Protocol_G può essere contenuto o meno in più Dataset_G;
- **“on”:** associazione tra *Analysis* e *Subject*. Un’analisi è fatta su uno o più Subject_G ed un Subject_G può essere contenuto o meno in più analisi;
- **“include”:** associazione tra *Dataset* e *Group*. Un Dataset_G include un unico gruppo di Subject_G ed un gruppo può essere incluso o meno in più Dataset_G;
- **“prevedere”:** associazione tra *Protocol* e *ClusterAlgorithm*. Un Protocol_G contiene al più un algoritmo di clustering_G ed una particolare istanza di un algoritmo di clustering_G, può essere presente in uno o più Protocol_G;
- **“calculate”:** associazione tra *Protocol* e *Feature*. Un Protocol_G può calcolare più feature_G oppure nessuna ed una particolare istanza di una feature_G, può essere presente in un solo Protocol_G;
- **“isDoing”:** associazione tra *Analysis* e *Dataset*. Un’analisi viene fatta su un particolare Dataset_G ed un Dataset_G può essere presente o meno su più analisi;
- **“belongs”:** associazione tra *Analysis_Features_Save_Show* e *Analysis*. Le opzioni sulla feature_G appartengono ad un’analisi, un’analisi ha più opzioni sulle feature_G;
- **“isOf”:** associazione tra *Analysis_Features_Save_Show* e *Feature*. Le opzioni sono di una feature_G, una feature_G può avere più opzioni;

6.3 Progettazione logica

A fronte di quanto descritto precedentemente (sezione 6.2), emerge la necessità di integrare lo schema del database con nuove tabelle. Questo è dovuto al fatto che sono presenti associazioni molti a molti. In figura 30 queste tabelle 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*.
- **Analysis_Subject:** rappresenta l’associazione “on” e ha come attributi le «PK» di *Analysis* e *Subject*.

La traduzione dello schema concettuale in schema logico, ha portato all’aggiunta, in alcune tabelle, di alcuni attributi come «FK» che riferiscono la chiave primaria «PK» di altre tabelle.

- **DatasetName:** «FK» in *Analysis* che rappresenta il Dataset_G su cui viene fatta l’analisi;

- **GroupName:** «FK» in *Dataset* che rappresenta il gruppo di Subject_G contenuto nel Dataset_G;
- **ProtocolName:** «FK» in *Feature* che rappresenta il Protocol_G a cui è associata l'istanza della feature_G;
- **ID_Algorithm** «FK» in *Protocol* che rappresenta l'eventuale algoritmo di clustering_G associato al Protocol_G;
- **ID_Feature** «FK» in *Analysis_Features_Save_Show* che rappresenta la feature_G da esportare o visualizzare durante l'analisi;
- **ID_Analysis** «FK» in *Analysis_Features_Save_Show* che rappresenta a quale analisi la feature_G da esportare o visualizzare è associata;

7 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. 31) 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.

7.1 Attività principali

Una volta avviato il programma, l'utente può:

- **Avviare un'analisi;**
- **Creare:** nuovi Subject_G, nuovi gruppi di Subject_G, nuovi Protocol_G e nuovi Dataset_G;
- **Visualizzare:** i Subject_G, i gruppi di Subject_G, i Protocol_G e i Dataset_G presenti nel sistema, oltre ai risultati della analisi finora effettuate;
- **Modificare:** i gruppi di Subject_G (aggiungendo o togliendo uno o più Subject_G);
- **Eliminare:** Protocol_G e Dataset_G;
- **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.

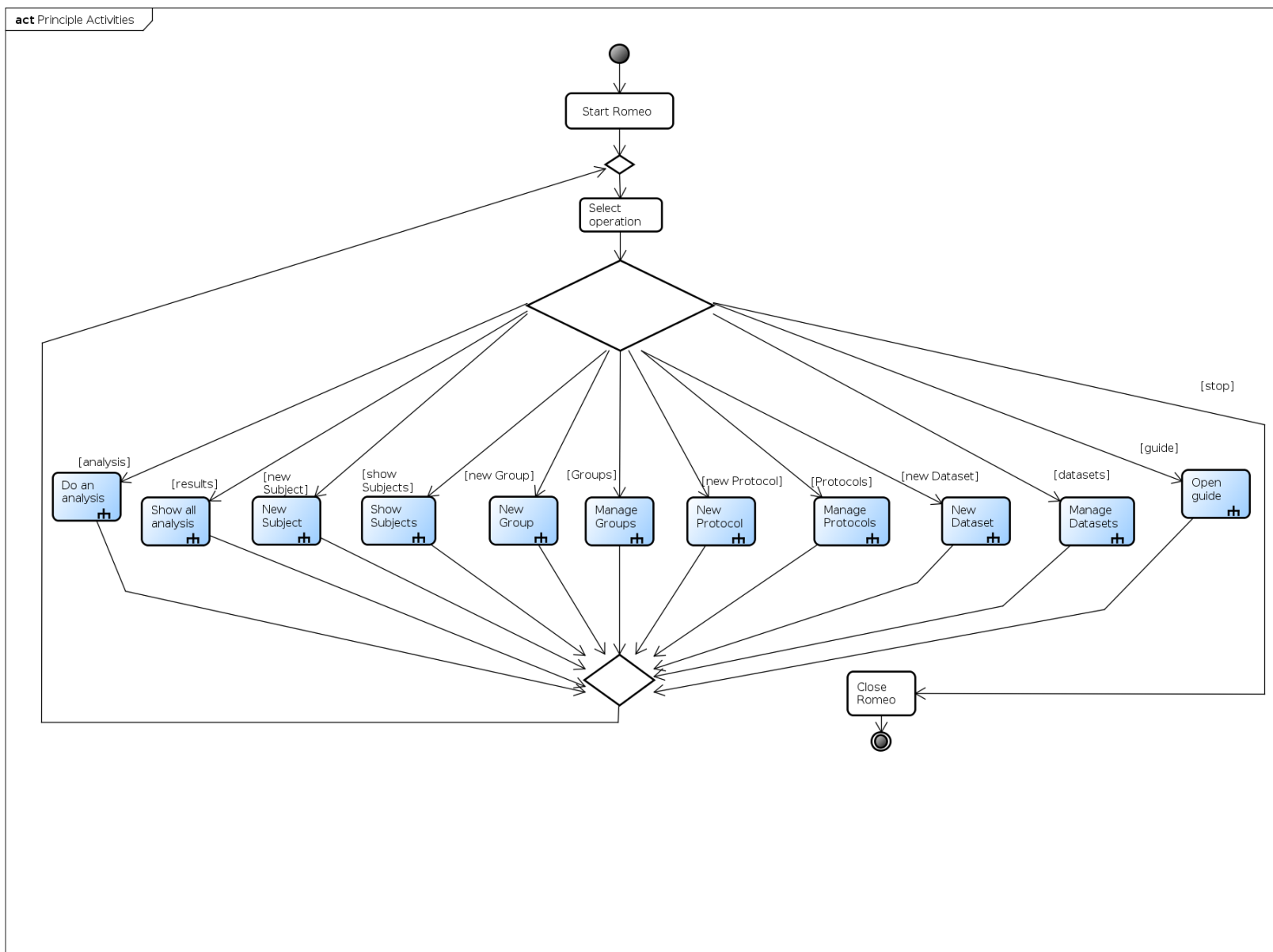


Figura 31: Diagramma Attività - Attività principali dell'applicativo Romeo

7.2 New Subject

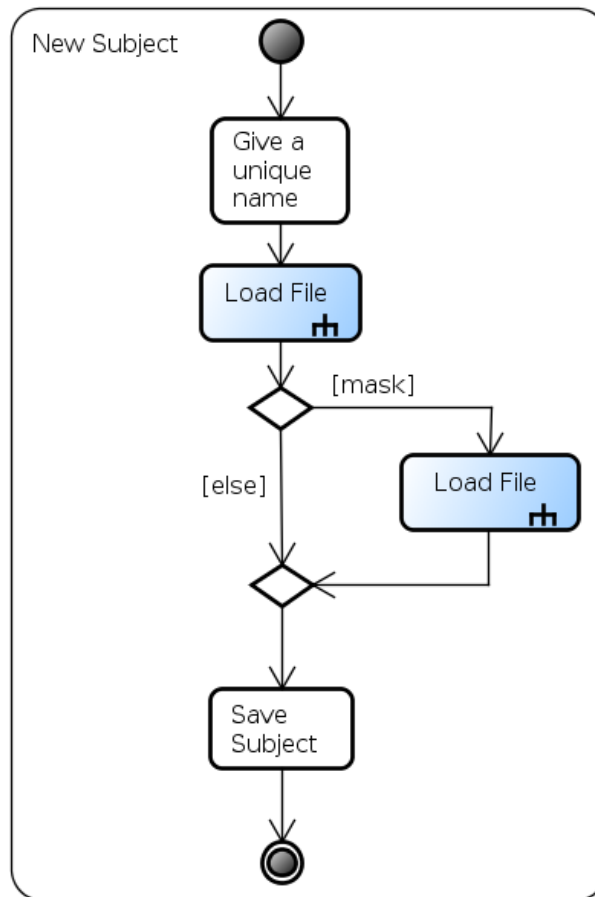


Figura 32: Diagramma Attività - Creazione nuovo Subject

Descrizione

L'attività di creazione di un nuovo Subject_G (fig. 32), prevede innanzitutto l'assegnazione di un nome univoco al Subject_G 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_G.

7.2.1 Load File

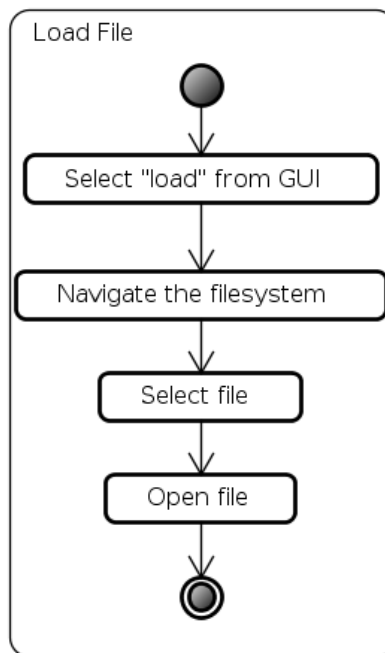


Figura 33: Diagramma Attività - Caricamento di un file

Descrizione

L'attività di caricamento di un file (fig. 33), 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.

7.3 New Group

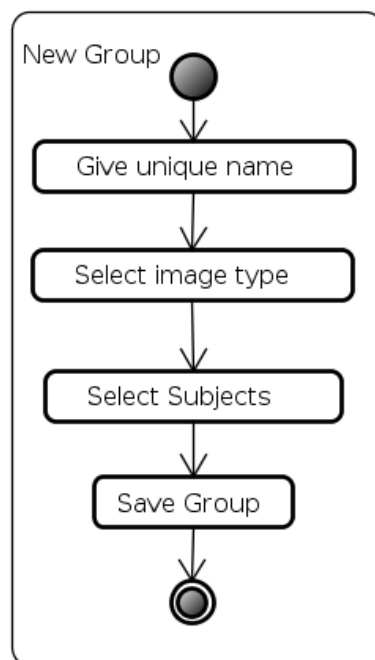


Figura 34: Diagramma Attività - Creazione nuovo gruppo di Subject

Descrizione

L'attività di creazione di un nuovo gruppo di Subject_G (fig. 34), 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_G da inserire nel gruppo, scegliendo tra quelli che hanno un'immagine associata del tipo precedentemente scelto. Infine si procede con il salvataggio del gruppo.

7.4 New Protocol

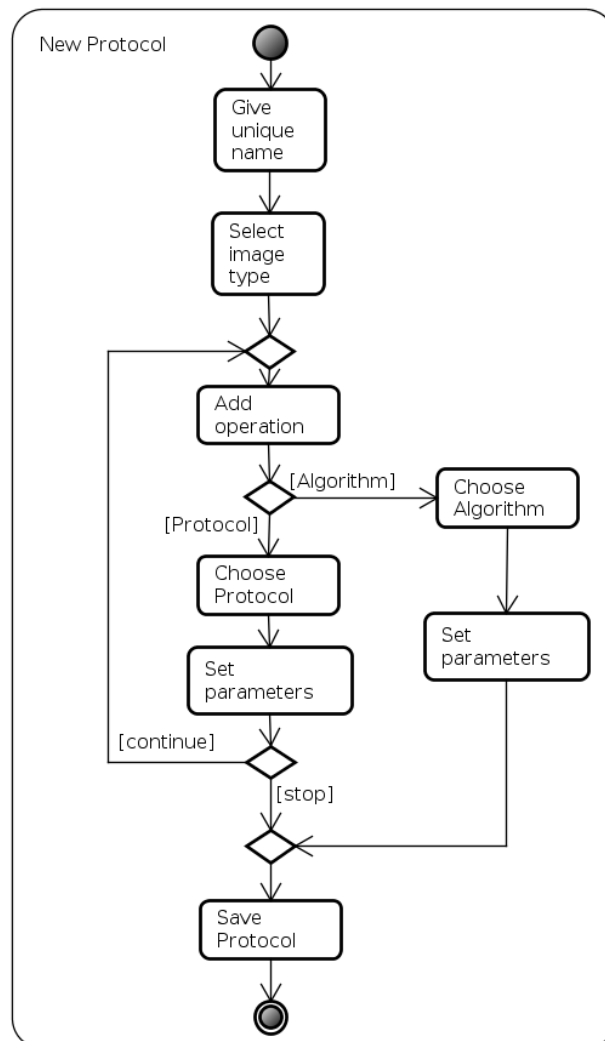


Figura 35: Diagramma Attività - Creazione di un nuovo Protocol

Descrizione

L'attività di creazione di un nuovo Protocol_G (fig. 35), prevede in primo luogo l'assegnazione di un nome univoco al Protocol_G e la scelta del tipo di immagine a cui dovrà essere applicato. È possibile poi selezionare le feature extractors_G che si vogliono utilizzare, dando dei valori ai parametri richiesti, e/o selezionare l'algoritmo di clustering_G 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_G è pronto per essere salvato.

7.5 New Dataset

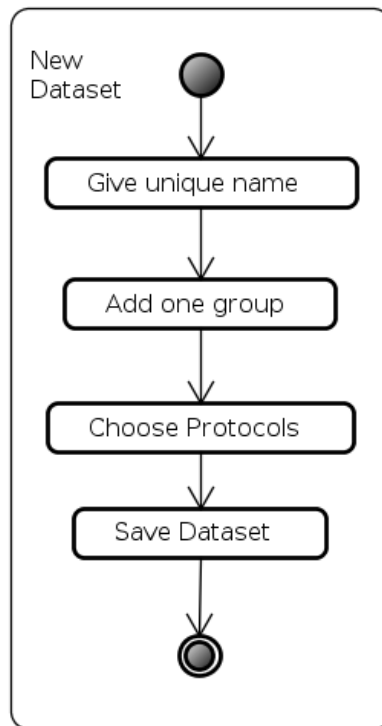


Figura 36: Diagramma Attività - Creazione di un nuovo Dataset

Descrizione

L'attività di creazione di un nuovo Dataset_G (fig. 36), prevede in primo luogo l'assegnazione di un nome univoco al Dataset_G in creazione e l'inserimento, in quest'ultimo, di un unico gruppo di Subject_G. Successivamente, è necessario scegliere uno o più protocol_G da applicare al gruppo di Subject_G. I Protocol_G che potranno essere associati, saranno solo quelli compatibili in base al tipo di immagine del gruppo. Creata quest'associazione, il Dataset_G è pronto per essere salvato.

7.6 Show Subjects

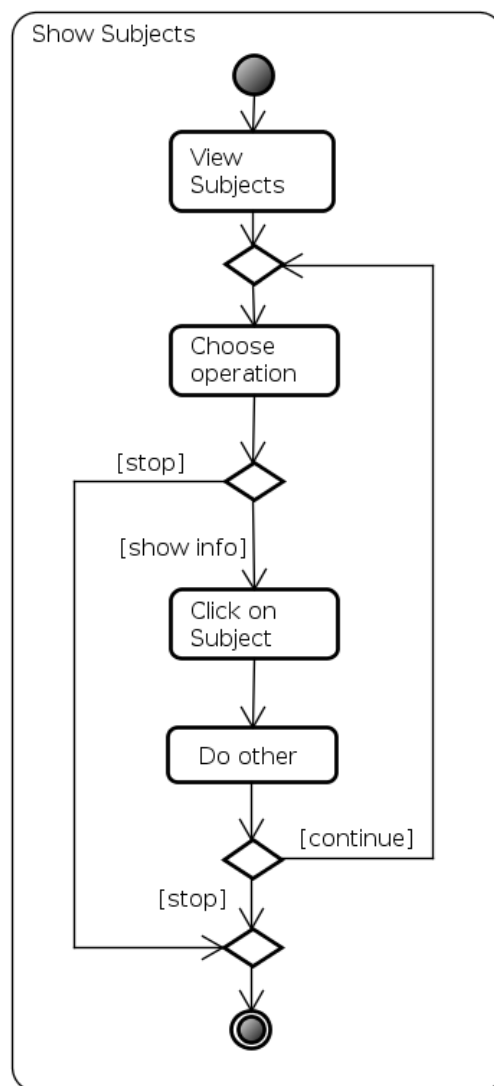


Figura 37: Diagramma Attività - Visualizzazione dei Subject

Descrizione

L'utente avrà a disposizione l'elenco di tutti i Subject_G 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.

7.7 Manage Groups

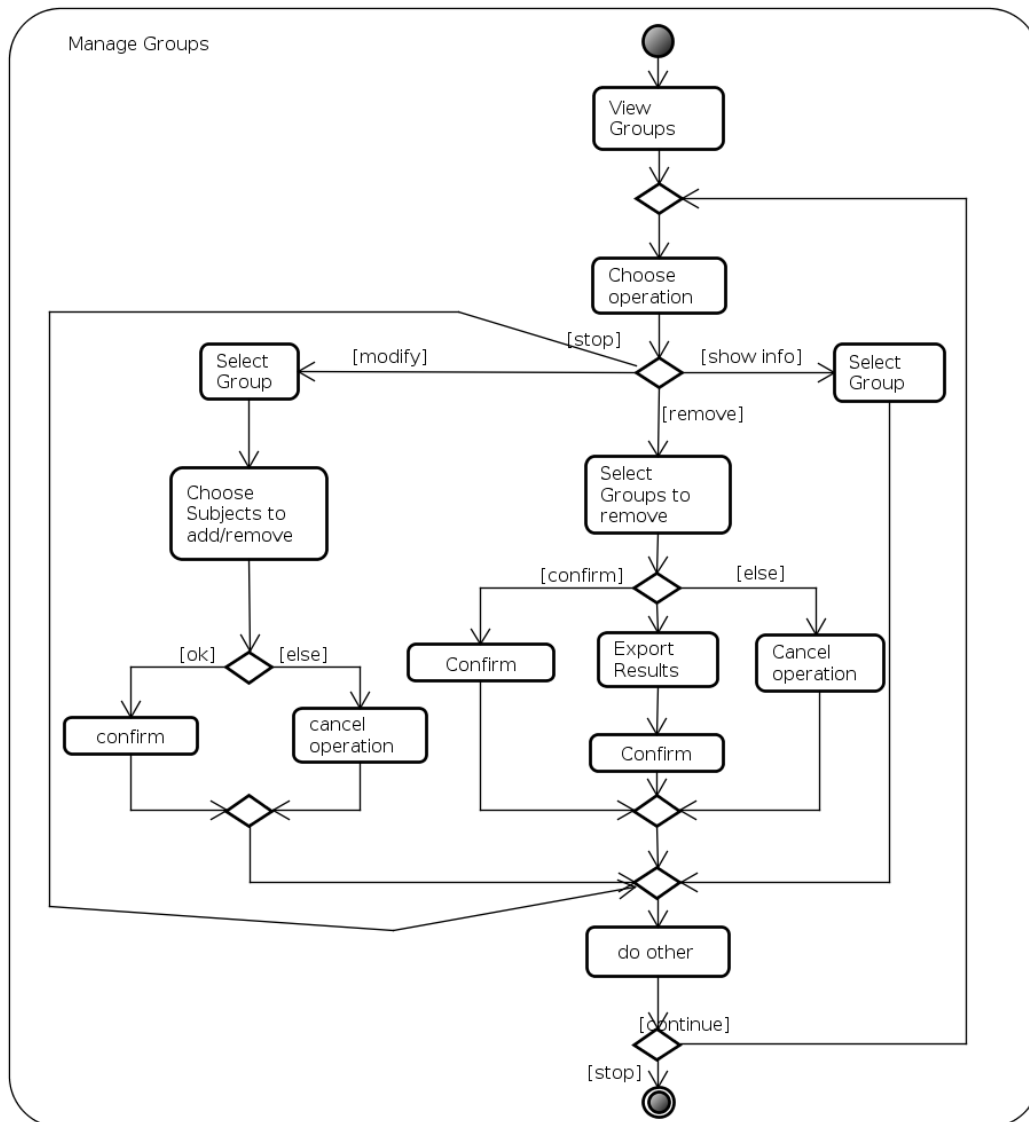


Figura 38: Diagramma Attività - Gestione dei gruppi di Subject

Descrizione

L'attività di gestione dei gruppi di Subject_G (fig. 38), 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_G appartenenti, tipo di immagine, ecc...) e modificarlo. La modifica consiste nel selezionare il gruppo e scegliere quali Subject_G eliminare o inserire.

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

7.8 Manage Protocols

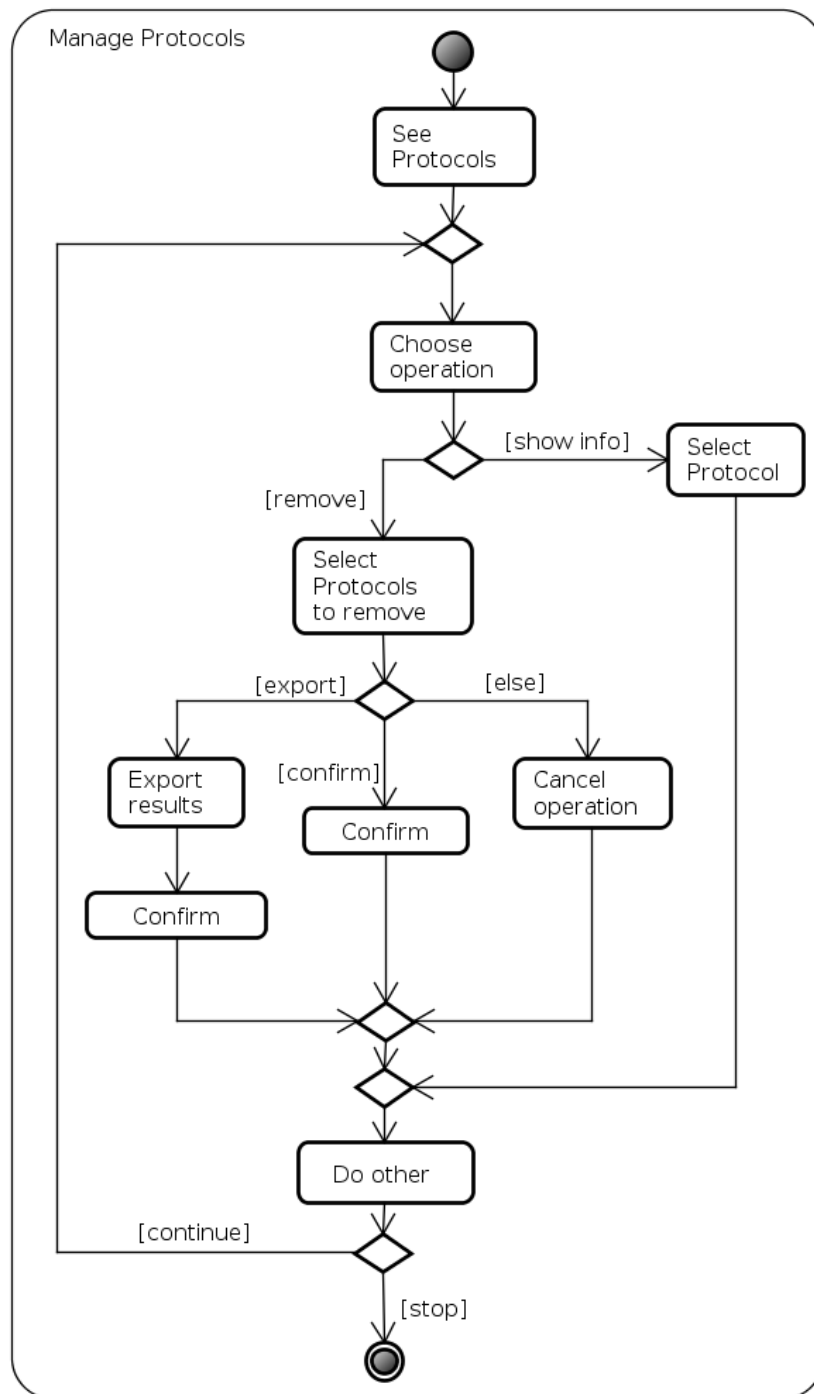


Figura 39: Diagramma Attività - Gestione dei Protocol

Descrizione

L'attività di gestione dei Protocol_G (fig. 39), dà la possibilità all'utente di visualizzare i Protocol_G presenti in quel momento nel sistema. Per ogni Protocol_G, l'utente può decidere se visualizzarne le informazioni d'interesse o se rimuoverlo. La cancellazione di un Protocol_G necessita della conferma dell'utente.

7.9 Manage Datasets

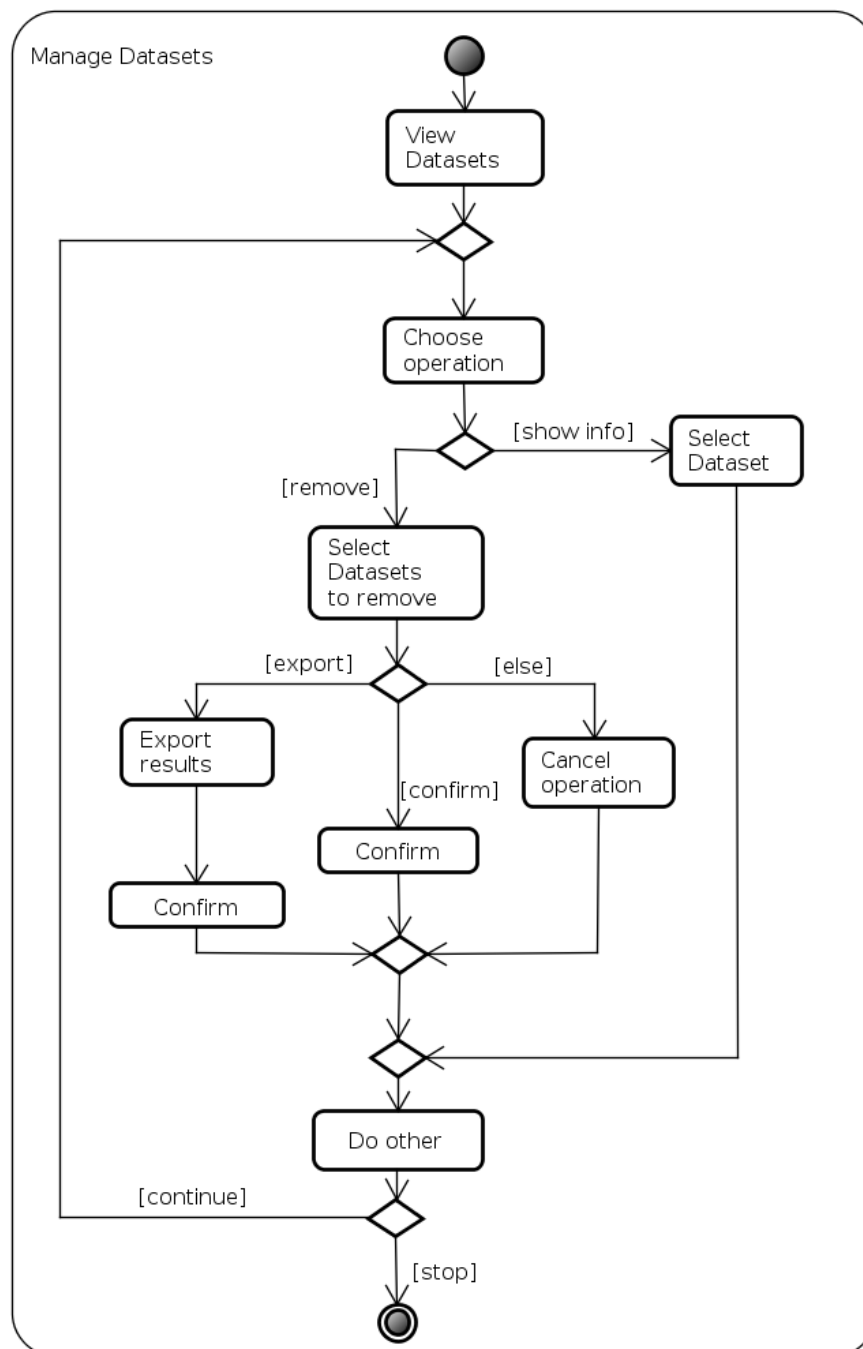


Figura 40: Diagramma Attività - Gestione dei Dataset

Descrizione

L'attività di gestione dei Dataset_G (fig. 40), dà la possibilità all'utente di visualizzare i Dataset_G presenti in quel momento nel sistema. Per ogni Dataset_G, è possibile decidere se visualizzarne le informazioni d'interesse o se rimuoverlo. La cancellazione di un Dataset_G necessita della conferma da parte dell'utente.

7.10 Do an Analys

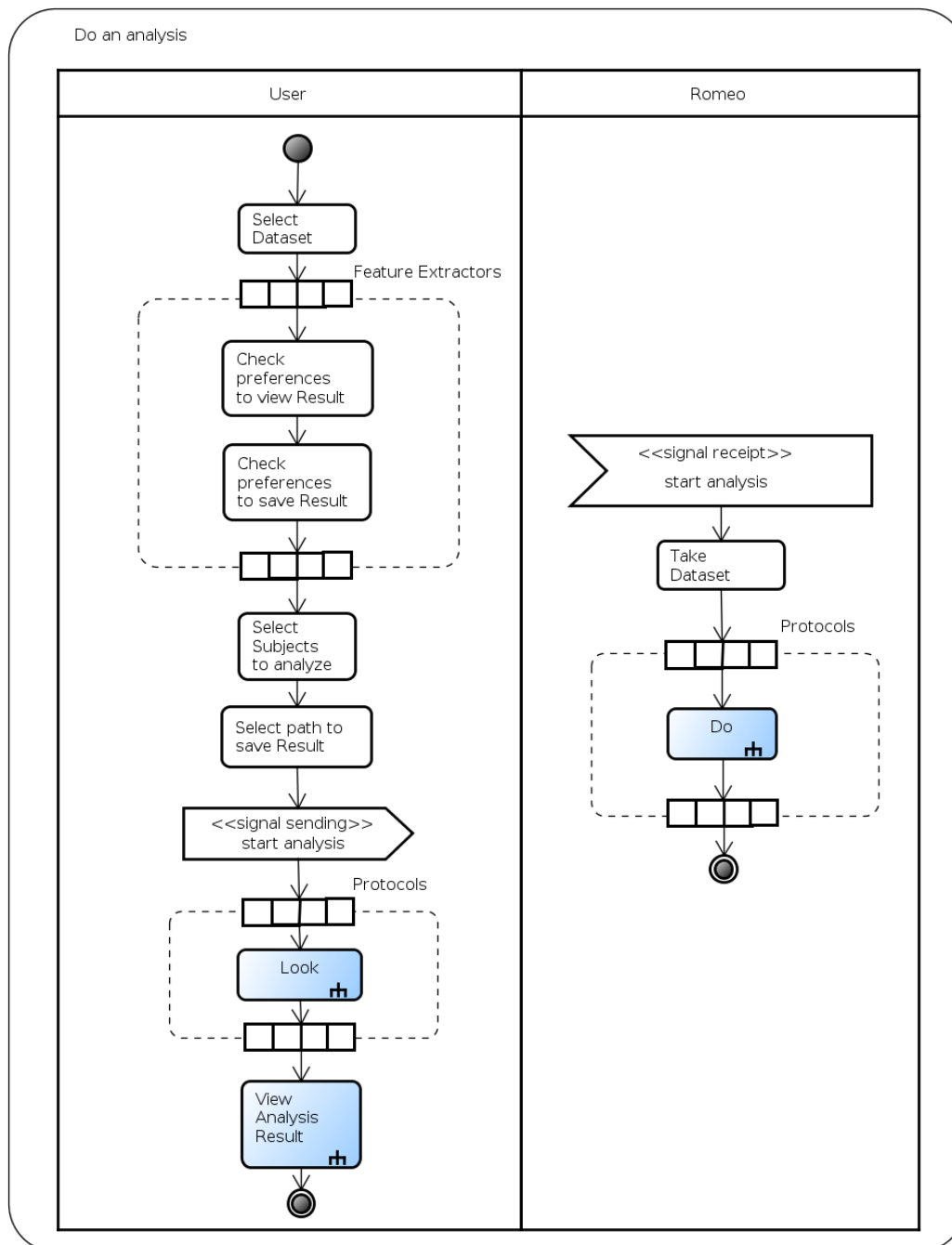


Figura 41: Diagramma Attività - Avvio di un'analisi

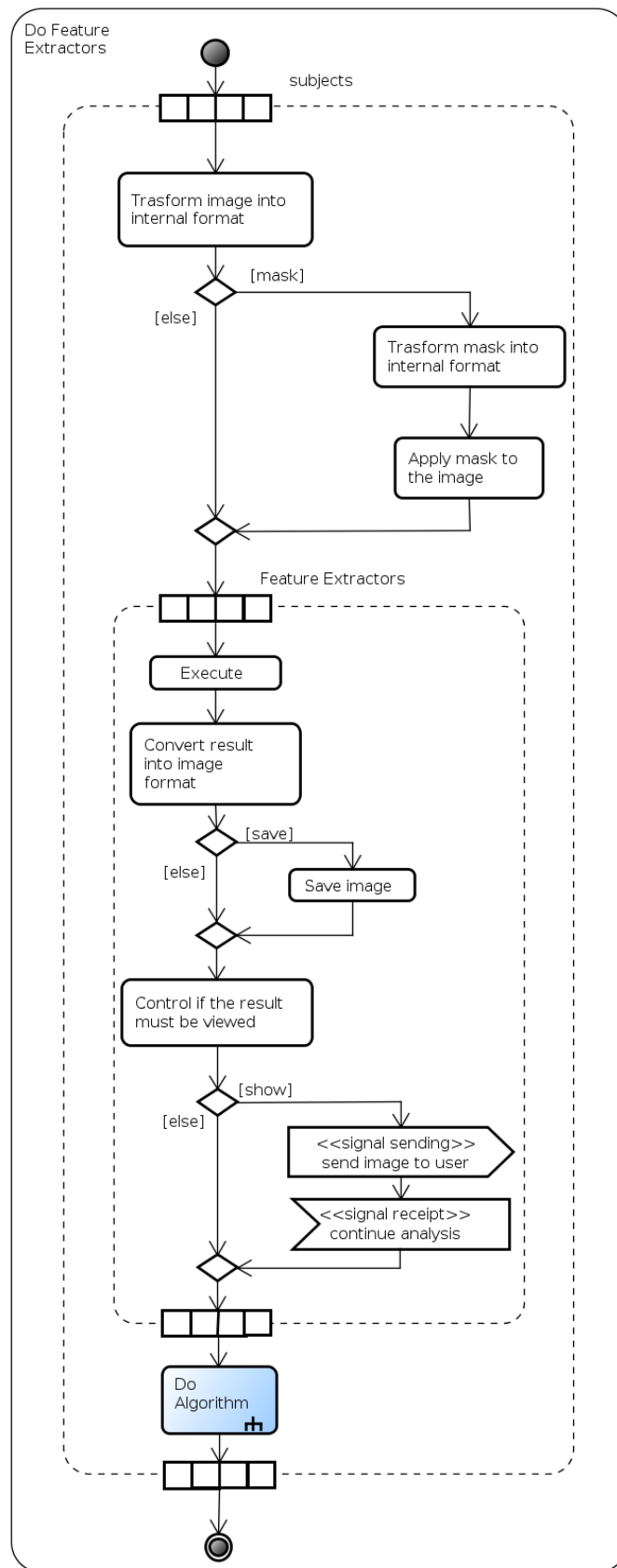


Figura 42: Diagramma Attività - Esecuzione analisi per ogni Protocol

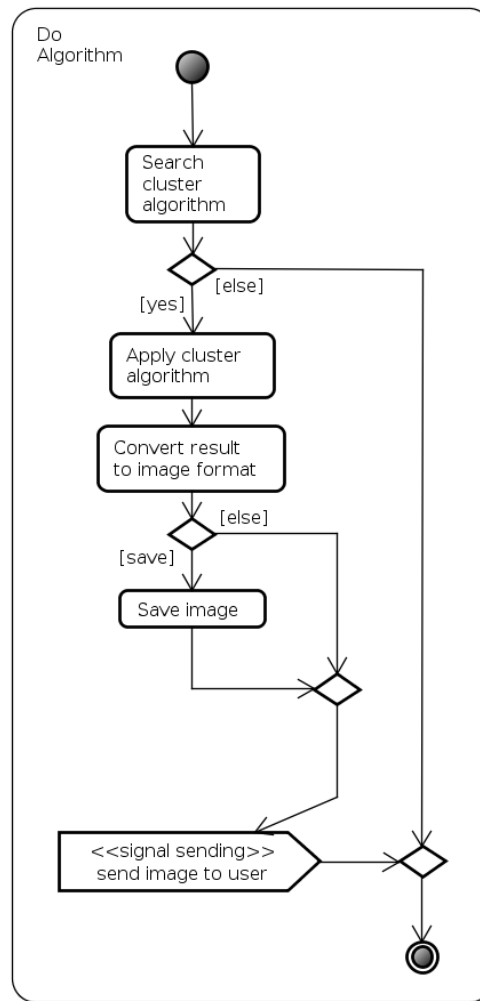


Figura 43: Diagramma Attività - Esecuzione analisi per ogni Protocol

Descrizione

L'attività di esecuzione di un'analisi (fig. 41), 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_G su cui vuole eseguire l'analisi. Successivamente, per ogni feature extractor_G presente nei vari Protocol_G del Dataset_G, dovrà decidere se visualizzare e se salvare il risultato dell'estrazione, subito dopo il suo calcolo. L'utente dovrà poi selezionare i Subject_G 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_G presente nel Dataset_G:

- Trasforma l'immagine e l'eventuale maschera di un Subject_G nel formato interno al software;
- Applica la maschera all'immagine (se non è presente, questo passo verrà saltato);
- Per ogni feature extractor_G presente nel Protocol_G:
 - Esegue l'estrazione della feature_G 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. 42);
- Lascia decidere all'utente se continuare a visualizzare i risultati delle features extractors_G oppure continuare l'estrazione senza più mostrarli;
- Viene eseguito l'algoritmo di clustering_G eventualmente presente nel Protocol_G;
- Il risultato verrà riconvertito in immagine;
- L'immagine verrà salvata (condizione precedentemente decisa dall'utente);

I passi sopra elencati, verranno eseguiti per ogni Subject_G presente nel gruppo di Subject_G del Dataset_G.

Una volta terminata l'analisi, l'utente potrà visualizzare tutti risultati dell'analisi (vedi 7.10.1).

7.10.1 Visualizzare i risultati dell'analisi

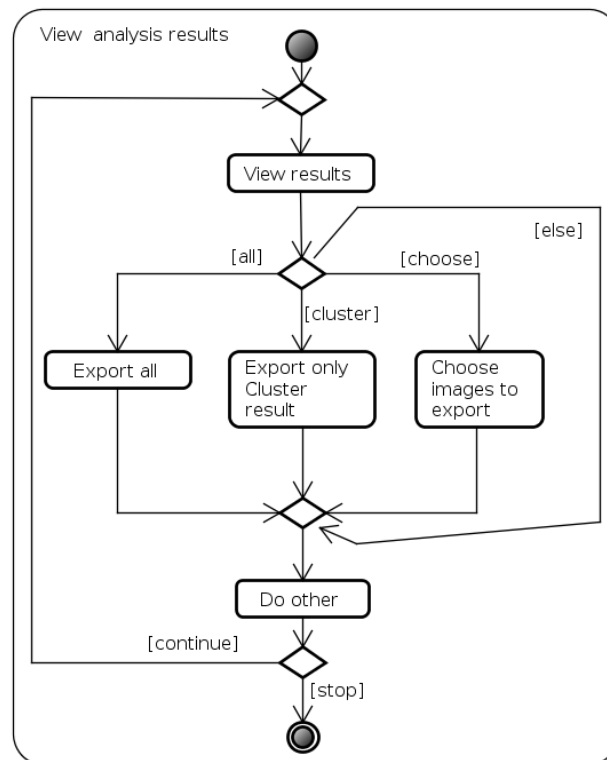


Figura 44: Diagramma Attività - Visualizzazione dei risultati dell'analisi effettuata

Descrizione

Quest'attività, conseguente alla fine dell'analisi del Dataset_G, 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_G oppure decidere manualmente quali immagini esportare.

7.11 Visualizzare le analisi effettuate

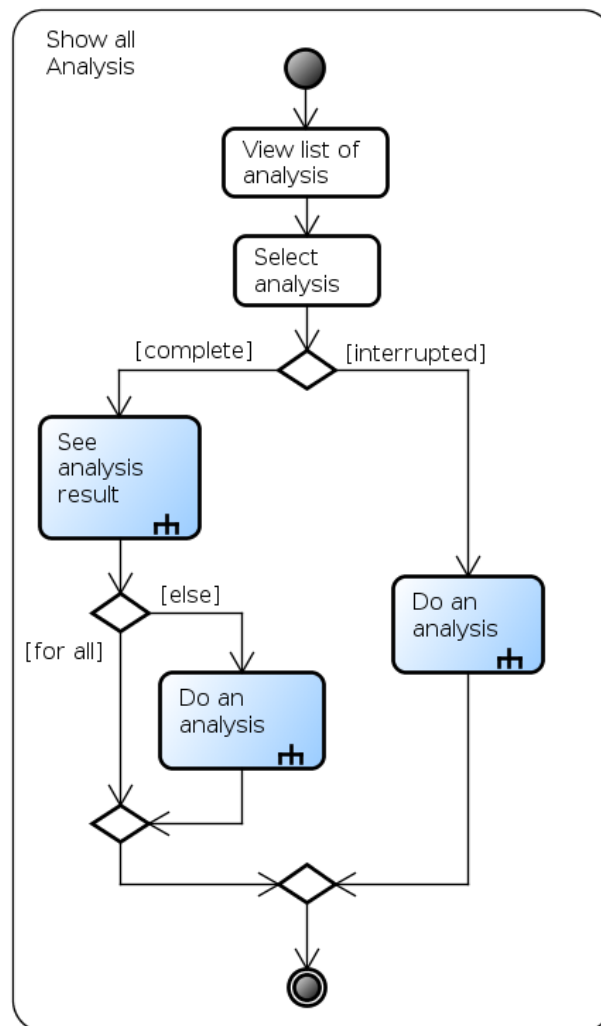


Figura 45: Diagramma Attività - Visualizzazione di tutte le analisi effettuate

Descrizione

Quest'attività permette all'utente di visualizzare lo storico delle analisi effettuate sui Dataset_G presenti nel software. Si possono presentare tre casi:

- L'analisi è stata completata su tutti gli elementi del gruppo di Subject_G. Sarà quindi possibile visualizzarne i risultati;
- L'analisi è stata completata solo per un sottoinsieme di Subject_G 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. 41).

7.12 Aprire la guida

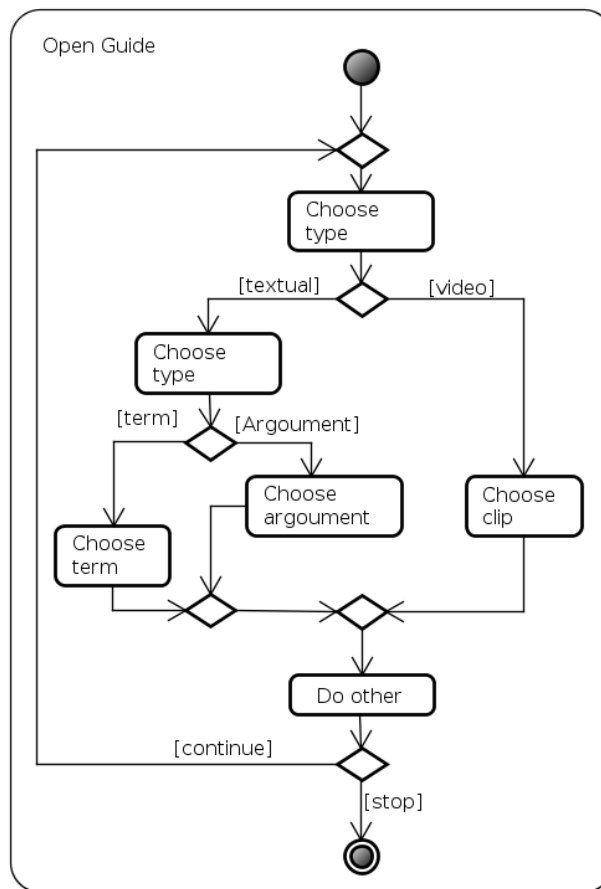


Figura 46: Diagramma Attività - Apertura della guida

Descrizione

L'attività di apertura della guida (fig. 7.12), 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.

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 _G	Core
R0F1.2.1.2	Il sistema deve accettare in input file di formato JPG _G	Core
R0F1.2.1.3	Il sistema deve accettare in input file di formato BMP _G	Core
R0F1.2.1.4	Il sistema deve accettare in input file di formato AVI _G	Core
R0F1.2.1.5	Il sistema deve accettare in input file di formato NIfTI _G	Core

R0F1.2.1.6	Il sistema deve accettare in input file di formato Analyze7.5 _G	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 _G 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 _G come maschera di un immagine 3D o 3D time dependent	Core
R0F1.3.5	L'utente può caricare un file di formato Analyze _G 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 _G 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 _G 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 _G 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 _G 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 _G	Algorithms
R0F5.4.2.4.1	Il valore di default per la soglia di probabilità di Fuzzy C _G 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 _G 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_G) 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.

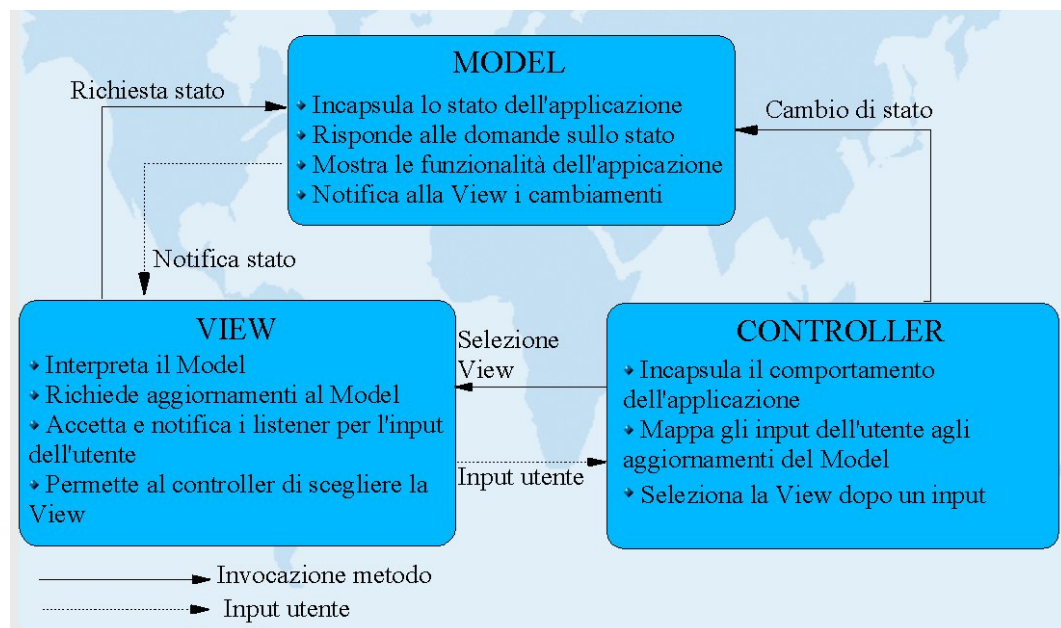


Figura 47: Diagramma del design pattern MVC

Scopo

Disaccoppiare le tre componenti.

Partecipanti e responsabilità

- **Model:** Analizzando la figura 47, 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_G), che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema. Ogni GUI_G 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_G MVC_G 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

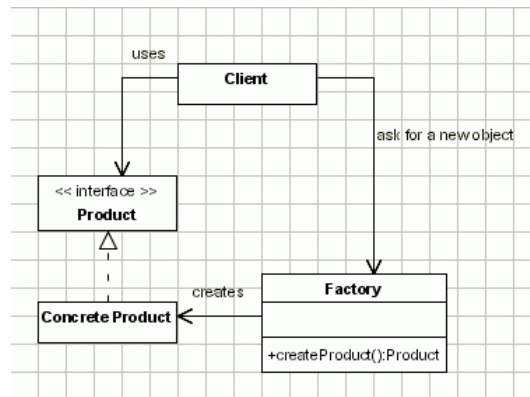


Figura 48: Diagramma del design pattern Factory

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 49: 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

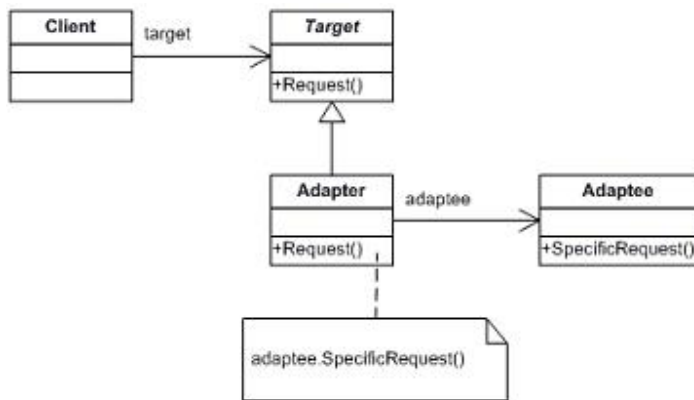


Figura 50: Diagramma del design pattern Adapter

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 **G** 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)

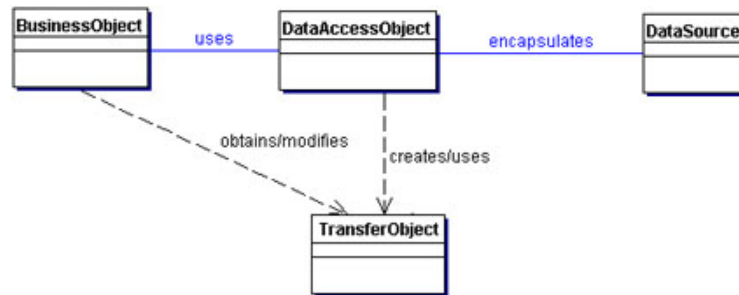


Figura 51: Diagramma del design pattern DAO

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_G 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_G, rendendo il sistema manutenibile.

A.3.3 Proxy

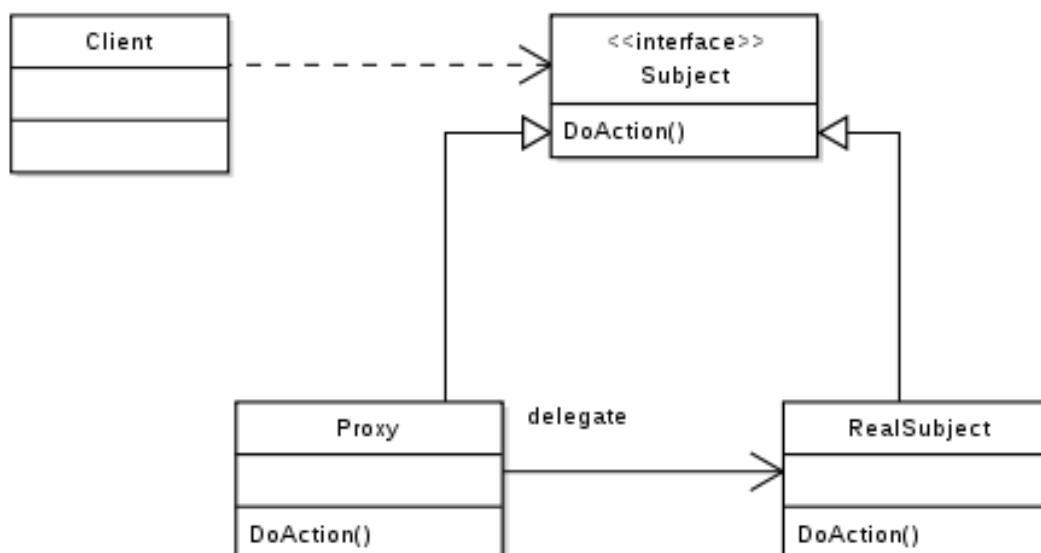


Figura 52: Diagramma del design pattern Proxy

Scopo

Fornire un surrogato o un segnaposto di un altro oggetto per controllare l'accesso a tale oggetto.

Motivazione

Una ragione per effettuare un controllo sull'accesso a un oggetto può essere quella di rinviare il costo della sua creazione e inizializzazione fino a quando l'oggetto non è effettivamente necessario.

Applicabilità

Il design pattern è applicabile nelle seguenti situazioni:

- Un proxy remoto fornisce un rappresentante local per un oggetto in diverso spazio di indirizzamento;
- Un proxy virtuale gestisce la creazione su richiesta di oggetti “costosi”;
- Un proxy di protezione controlla l'accesso a un oggetto. Questo tipo di proxy si rivela utile quando possono essere definiti diritti di accesso diversi per gli oggetti;
- Un riferimento intelligente sostituisce un puntatore puro ad un oggetto, consentendo l'esecuzione di attività aggiuntive quando si accede all'oggetto referenziato.

A.4 Design pattern comportamentali

A.4.1 Strategy

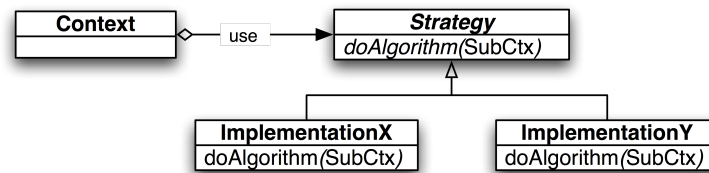


Figura 53: Diagramma del design pattern Strategy

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 **G** 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_G.

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.

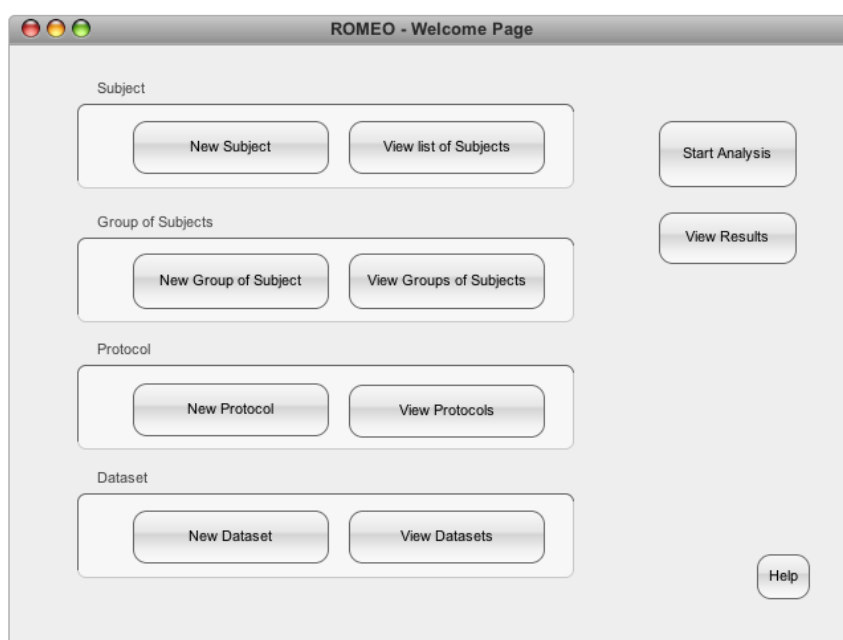


Figura 54: Romeo: Mock-up della pagina di benvenuto

B.2 Pagine di creazione

B.2.1 Creazione di un nuovo Subject

In questa pagina sarà possibile inserire un nuovo subject_G, specificandone il *nome*, caricando il file dell'immagine o del video e aggiungendo un eventuale maschera_G.

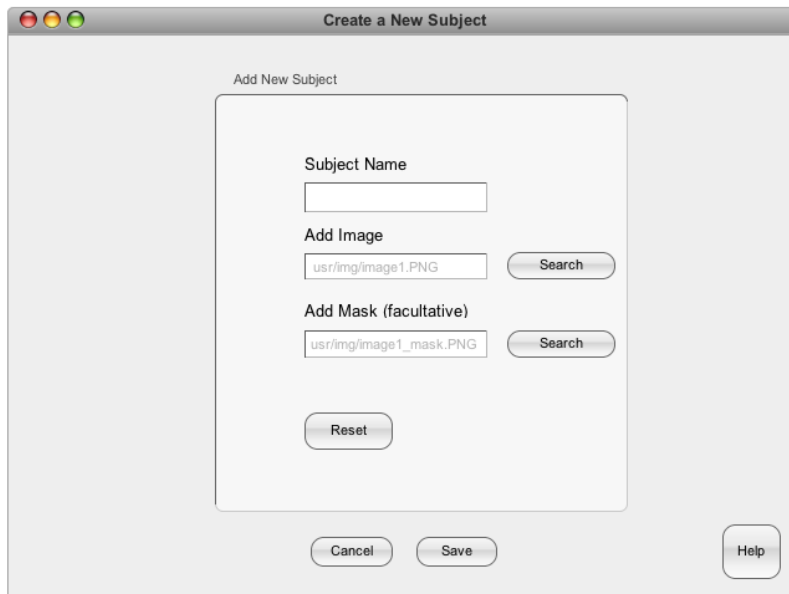


Figura 55: Mock-up della pagina di creazione di un nuovo Subject

B.2.2 Creazione di un gruppo di Subject

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

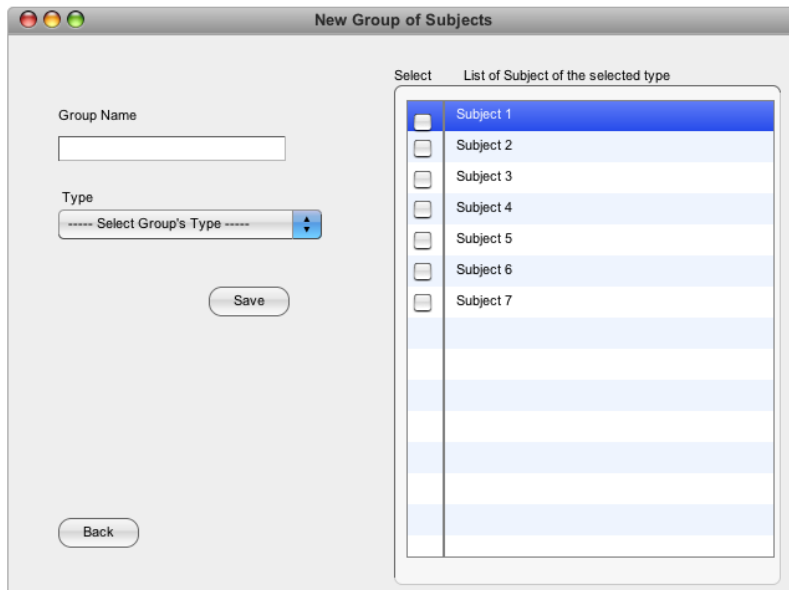


Figura 56: Mock-up della pagina di creazione di un gruppo di Subject

B.2.3 Creazione di un Protocol

In questa pagina sarà possibile creare un nuovo protocol_G, inserendo il nome del protocol_G, il tipo, una o più feature_G e un algoritmo di cluster_G da applicare per l'analisi.

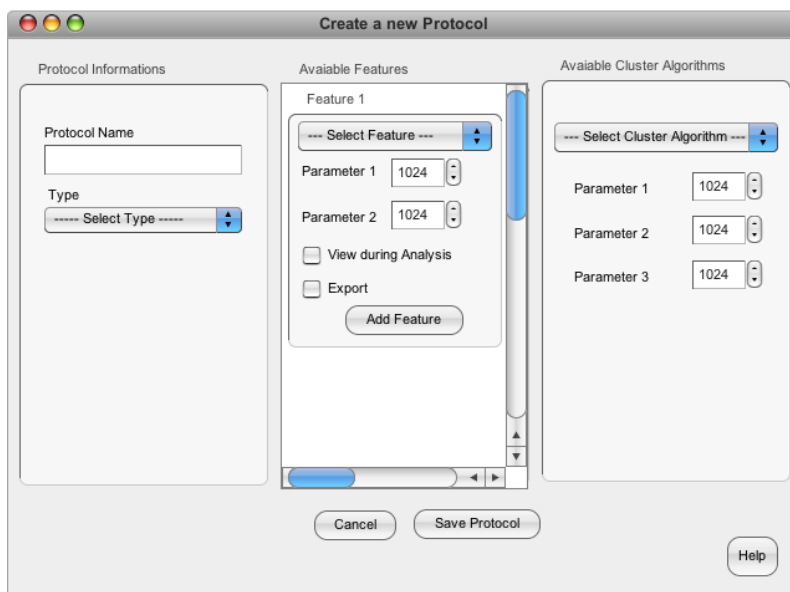


Figura 57: Mock-up della pagina di creazione di un Protocol

B.2.4 Creazione di un Dataset

In questa pagina sarà possibile creare un nuovo dataset_G, inserendo il nome, scegliendo uno dei gruppi di subject_G esistenti e uno o più protocol_G da applicare all'analisi.

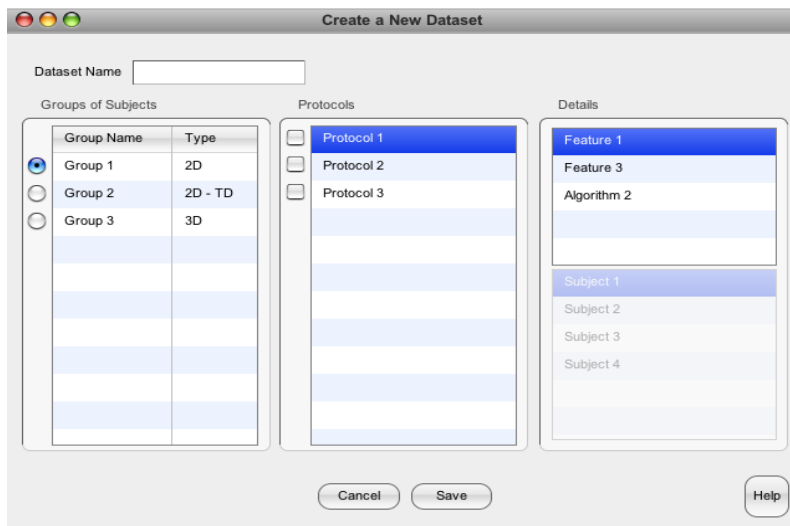


Figura 58: Mock-up della pagina di creazione di un Dataset

B.3 Pagine di visualizzazione e modifica

In queste pagine sarà possibile visualizzare e/o gestire⁵ le varie entità⁶ 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 subject_G presenti nel sistema. Per ogni subject_G verrà visualizzato un pannello contenente i dettagli dell'immagine.

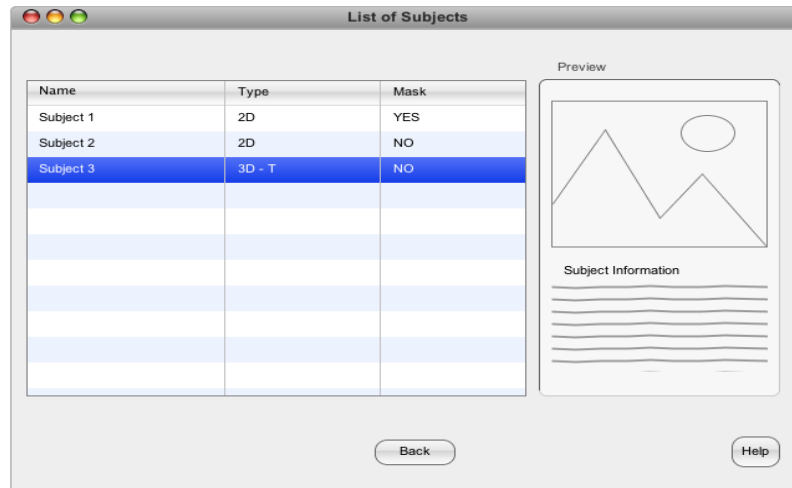


Figura 59: Mock-up della pagina di visualizzazione dei Subject

B.3.2 Visualizzazione dei gruppi di Subject

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

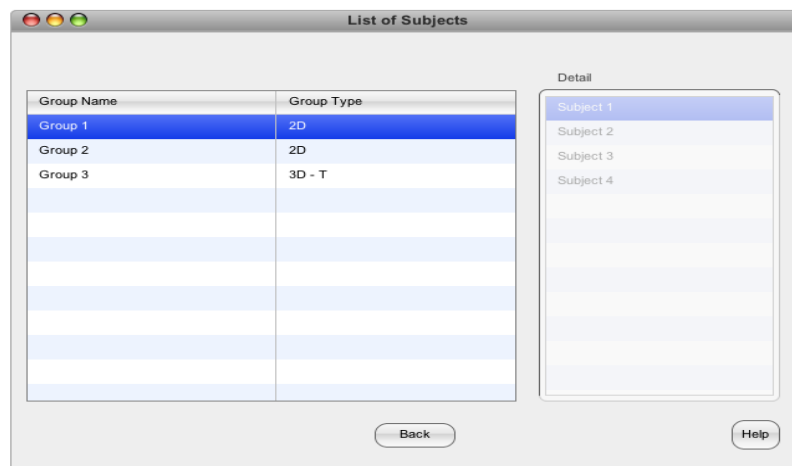


Figura 60: Mock-up della pagina di visualizzazione dei gruppi di Subject

⁵La gestione è prevista per tutte le entità tranne che per i subject_G

⁶Il termine è da intendersi come sostitutivo di: *subject_G, gruppi di subject_G, protocol_G e dataset_G*

B.3.3 Visualizzazione dei Protocol

In questa pagina verranno visualizzati tutti i protocol_G creati. Per ogni protocol_G selezionato, verrà visualizzata una lista delle feature_G che lo compongono. Sarà inoltre possibile selezionare uno o più protocol_G per l'eliminazione.

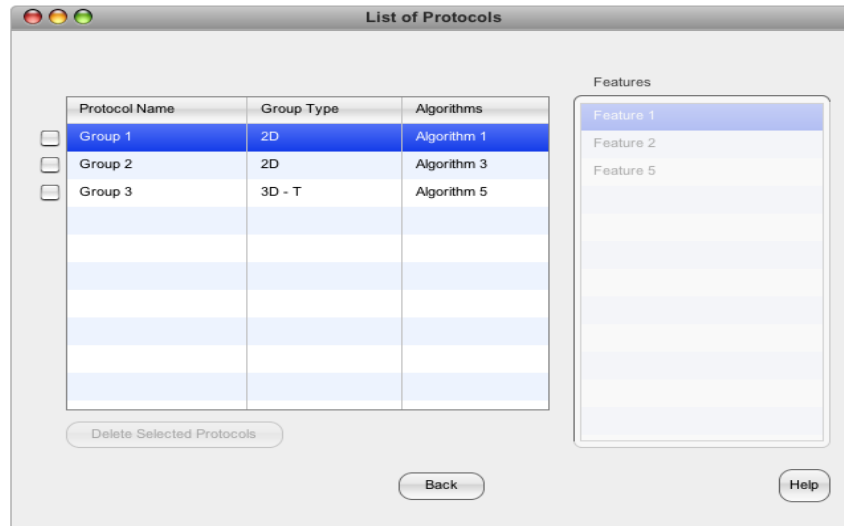


Figura 61: Mock-up della pagina di visualizzazione dei gruppi dei Protocol

B.3.4 Visualizzazione dei Dataset

In questa pagina verranno visualizzati i dataset_G presenti nel sistema. Per ogni dataset_G verranno visualizzati nel pannello dei dettagli, la lista dei protocol_G , l'algoritmo di cluster_G , i subject_G coinvolti nell'analisi ed eventuali informazioni aggiuntive. Sarà inoltre possibile selezionare uno o più dataset_G per l'eliminazione.

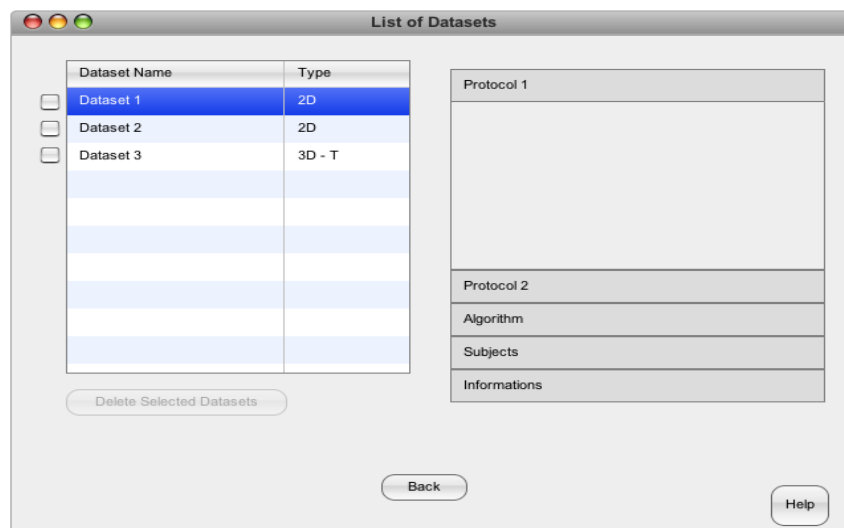


Figura 62: Mock-up della pagina di visualizzazione dei Dataset

B.4 Analisi e visualizzazione risultati

B.4.1 Avvio analisi

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

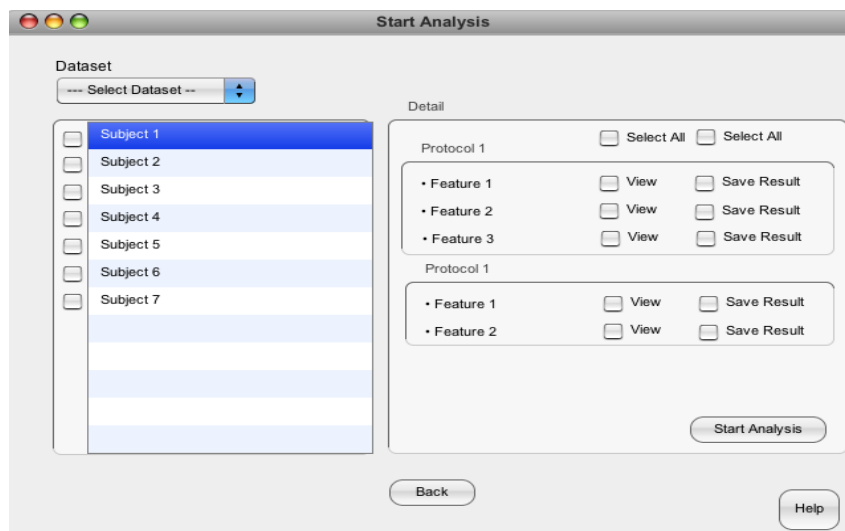


Figura 63: Mock-up della pagina di avvio analisi

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.

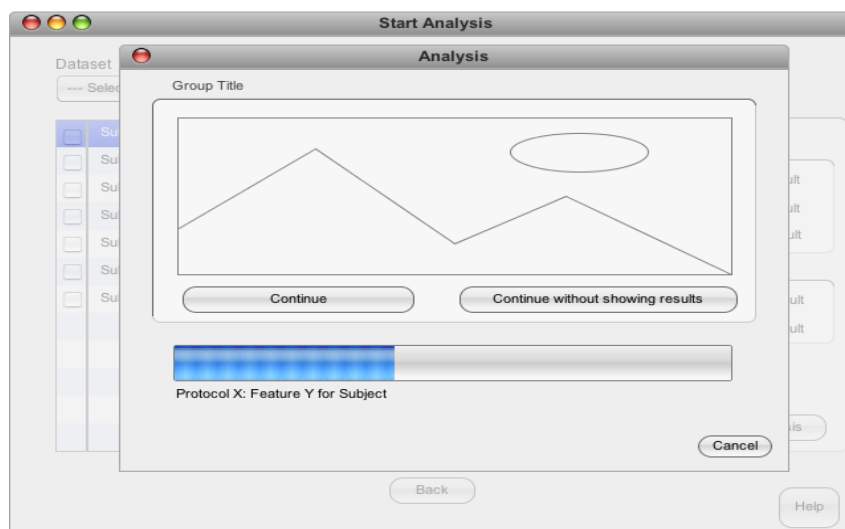


Figura 64: Mock-up della finestra di analisi

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_G 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.

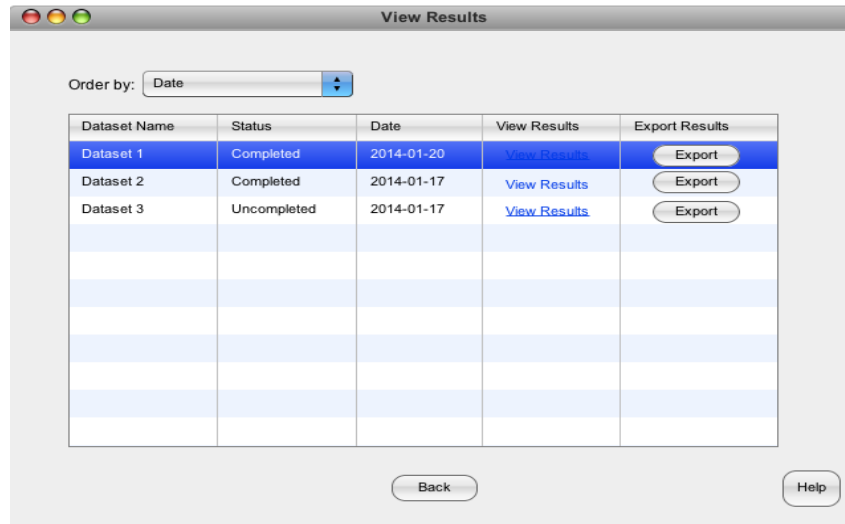


Figura 65: Mock-up della finestra dei risultati

B.4.4 Visualizzazione dettaglio risultati

In questa pagina sarà possibile visualizzare in dettaglio i risultati di un'analisi, filtrati per protocol_G (fig. 66) o per subject_G (fig.67). Sarà possibile quindi, visualizzare un'anteprima delle immagini prima dell'esportazione.

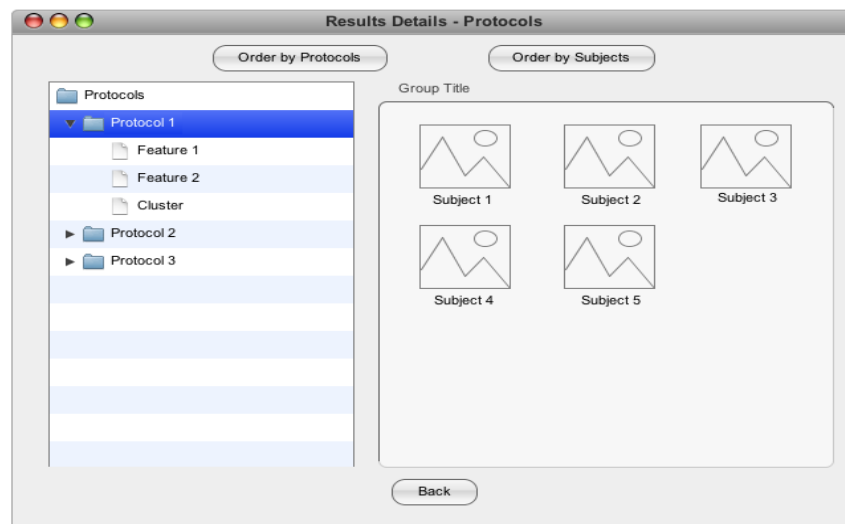


Figura 66: Mock-up della finestra di dettaglio dei risultati per Protocol

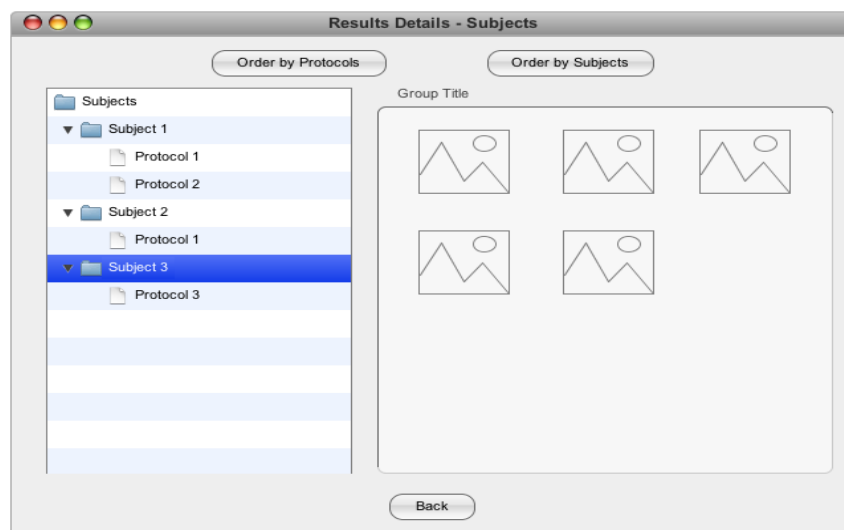


Figura 67: Mock-up della finestra di dettaglio dei risultati per Subject