Don't Panic

3DMob: Grafica 3D su device mobili

Specifica Tecnica

Informazioni sul documento

Versione	4.2.0
Redazione	Basaglia Mattia Lain Daniele Pezzutti Marco
Verifica	Cesarato Fabio Busato Luca
Responsabile	Rampazzo Federico
Uso	Esterno
Lista di distribuzione	Don't Panic Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Specifica tecnica e architettura dell'applicazione 3DMob

Diario delle modifiche

Descrizione modifica	Autore	Ruolo	Data	Versione
Approvazione documento	Rampazzo Federico	Responsabile	2013-02-13	4.2.0
Verifica documento	Cesarato Fabio	Verificatore	2013-02-12	4.1.1
Verifica documento	Busato Luca	Verificatore	2013-02-11	4.1.0
Modifica librerie esterne	Pezzutti Marco	Progettista	2013-02-10	4.0.3
Miglioramento qualità immagini	Lain Daniele	Progettista	2013-02-09	4.0.2
Correzione diagrammi di attività	Basaglia Mattia	Progettista	2013-02-08	4.0.1
Revisione correttiva struttura e posizione contenuti basata su segnalazione del committente	Lain Daniele	Progettista	2013-02-07	4.0.0
Approvazione documento	Sciarrone Riccardo	Responsabile	2013-01-30	3.2.0
Verifica del documento	Basaglia Mattia	Verificatore	2013-01-26	3.1.0
Stesura ed importazione tracciamento	Pezzutti Marco	Progettista	2013-01-25	3.0.7
Stesura stime di fattibilità e di bisogno di risorse	Lain Daniele	Responsabile	2013-01-23	3.0.6
Stesura descrizione design pattern$_G$	Rampazzo Federico	Progettista	2013-01-22	3.0.5
Stesura descrizione dei singoli componenti	Busato Luca	Progettista	2013-01-21	3.0.4
Stesura della architettura generale	Cesarato Fabio	Progettista	2013-01-18	3.0.3
Stesura del metodo e formalismo di specifica	Rampazzo Federico	Progettista	2013-01-18	3.0.2
Stesura tecnologie utilizzate	Lain Daniele	Progettista	2013-01-17	3.0.1
Creazione scheletro del documento e stesura introduzione	Pezzutti Marco	Analista	2013-01-16	3.0.0

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello del progetto 3DMob.
Verrà presentata l'architettura generale secondo la quale saranno organizzate le varie componenti software e saranno descritti i Design Pattern$_G$ utilizzati.

1.2 Scopo del Prodotto

Lo scopo del progetto è la realizzazione di un applicazione in grado di convertire file prodotti da programmi di grafica 3D in file in formato JSON$_G$ in grado di essere visualizzati su dispositivi mobile senza perdita di informazione. L'obiettivo è quello di semplificare il workflow attuale necessario a rendere compatibili i file.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v4.2.0*.
Ogni occorrenza di vocaboli presenti nel *Glossario* è marcata da una "G" maiuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- **Analisi dei Requisiti**: *Analisi dei Requisiti v4.2.0*;
- **Norme di Progetto**: *Norme di Progetto v4.2.0*.

1.4.2 Informativi

- **Documentazione Qt$_G$ per Segnali e Slot**
  http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html;
- **Descrizione dei Design Pattern$_G$**
  http://sourcemaking.com/design_patterns;
- **Design Patterns: Elementi per il riuso di software a oggetti - E. Gamma, R. Helm, R. Johnson, J. Vlissides - 1ª Edizione (2002)**;
- **Documentazione di Assimp**
  http://assimp.sourceforge.net/lib_html/index.html;
- **Documentazione Boost.PropertyTree**
  http://www.boost.org/doc/libs/1_52_0/doc/html/property_tree.html.

2 Tecnologie Utilizzate

In questa sezione vengono descritte le tecnologie su cui si basa lo sviluppo del progetto e la motivazione del loro utilizzo.

2.1 C++

Si è deciso di utilizzare il C++ poiché offre i seguenti vantaggi:

- **Librerie**: il C++ mette a disposizione numerose librerie, descritte successivamente, che permettono di semplificare la realizzazione di alcune funzionalità richieste come il parsing dei file e la realizzazione di interfacce 3D;
- **Conoscenza**: il gruppo ha già avuto modo di utilizzare il linguaggio C++, sia per progetti di altri corsi universitari sia per esperienze personali, permettendo di raggiungere un buon grado di conoscenza;
- **Strumenti di analisi**: la maggior parte dei componenti del gruppo ha utilizzato gli strumenti di analisi che permettono di verificare dinamicamente il codice prodotto.

2.2 Qt

Si è deciso di scegliere il framework$_G$ Qt$_G$ perché offre i seguenti vantaggi:

- **Funzionalità 3D:** tra le numerose librerie disponibili sono presenti alcune funzionalità che semplificheranno lo sviluppo del software;
- **Qt$_G$ Creator:** IDE$_G$ ottimizzato per lo sviluppo con Qt$_G$;
- **Qt$_G$ Designer:** editor grafico per disegnare l'interfaccia grafica;
- **Qt$_G$ Linguist:** tool in grado di semplificare il supporto e la traduzione per numerose lingue;
- **Crossplatform:** Qt$_G$ è un framework$_G$ crossplatform e tra le piattaforme supportate ufficialmente sono presenti Windows$_G$, OS X e molte distribuzioni GNU/Linux$_G$;
- **Esperienza del team:** tutti i membri del team hanno già lavorato con Qt$_G$ durante l'insegnamento di Programmazione ad Oggetti;
- **Licenza GNU LGPL$_G$ v2.1:** Qt$_G$ è disponibile con licenza GNU LGPL$_G$ v2.1 e può essere quindi utilizzato nel nostro progetto senza che sia necessario rilasciare il codice prodotto.

2.2.1 Segnali e Slot

Il framework$_G$ Qt$_G$ offre un sistema di comunicazione tra oggetti tramite l'uso di Signal$_G$. Qt$_G$ permette di connettere un Signal$_G$ ad un puntatore di funzione o ad un oggetto di funzione. Quando un oggetto emette un Signal$_G$ tutti i funtori$_G$ connessi ad esso vengono eseguiti nell'ordine in cui questi sono stati connessi al Signal$_G$.

2.2.1.1 Descrizione

Un Signal$_G$ è un metodo pubblico la cui implementazione viene generata dal meta-object compiler$_G$ di Qt$_G$.
Essendo parte del sistema di meta-informazioni è possibile definire Signal$_G$ solo in classi che derivano da `QObject` che contengono la macro `Q_OBJECT` e che sono stati preprocessati dal meta-object compiler$_G$.
Storicamente era possibile connettere ai Signal$_G$ solo degli Slot, anch'essi parte delle meta-informazioni. Con Qt$_G$ 5 questa limitazione è stata rimossa ed è possibile connettere ad un Signal$_G$ funzioni globali, metodi e funtori$_G$ generici.
Quando un Signal$_G$ viene emesso, la coda di funtori$_G$ connessi è eseguita in ordine passando eventuali argomenti forniti in fase di emissione.

2.2.1.2 Vantaggi

I Signal$_G$ permettono di far comunicare un oggetto con un numero illimitato di ascoltatori senza che questo aumenti le dipendenze per la classe istanziata da questo oggetto.
Il meccanismo di Signal$_G$ è parte integrante del framework$_G$ Qt$_G$ ed è usato dalla maggior parte delle classi fornite. Il suo utilizzo è quindi indispensabile per integrare Qt$_G$ all'applicativo che lo usa.

2.2.2 QMake

Qmake è il sistema di build$_G$ di Qt$_G$. Automatizza la compilazione generando un Makefile in base alle informazioni presenti sul file di progetto datogli in input. Il makefile generato da qmake si occupa dei seguenti passi:

- **User Interface Compiler**: genera codice a partire dai file del designer di interfacce grafiche;
- **Resource Compiler**: genera codice a partire dai file di risorse, per poter includere file di dati nell'eseguibile;
- **Meta-object Compiler$_G$**: genera il codice necessario per utilizzare le estensioni fornite da QObject;
- **Compilatore C++** genera codice oggetto a partire dai file sorgente e da quelli generati ai passi precedenti;
- **Linker** unisce tutti i file oggetto e inserisce informazioni riguardanti le librerie utilizzate producendo un file eseguibile.

2.2.2.1 Vantaggi

- il Makefile viene generato in base al sistema richiesto;
- automatizza tutti i processi necessari ad usare le estensioni di Qt$_G$;
- è integrato in Qt$_G$ Creator, permettendo di avere un ambiente di sviluppo rapido.

2.2.2.2 Svantaggi

- alcuni dei target del Makefile generato non sono ottimali;
- è difficile da integrare con altri sistemi di build$_G$.

2.3 Boost.PropertyTree

Si è deciso di usare Boost PropertyTree per effettuare il parsing dei file.
Vantaggi:

- **Estensibilità:** è possibile aggiungere supporto ad altri tipi di file in modo molto semplice;
- **Crossplatform:** Boost supporta tutti i sistemi più diffusi;
- **Header only:** La libreria non richiede dipendenze a file binari;
- **Esperienza del team:** alcuni membri del team hanno già lavorato con Boost;
- **Boost Software License Version 1.0:** Boost è disponibile con licenza Boost Software License 1.0 e può essere quindi utilizzato nel nostro progetto senza che sia necessario rilasciare il codice prodotto.

2.4 OpenGL

OpenGL$_G$ è una specifica che definisce una API per più linguaggi e per più piattaforme per scrivere applicazioni che utilizzino grafica 2D e 3D.
L'interfaccia consiste di diverse chiamate di funzione che vengono utilizzate per disegnare scene tridimensionali a partire da semplici primitive.

In particolare viene utilizzata la libreria software **Mesa 3D**.
Quest'ultima è una libreria grafica 3D$_G$ open source che fornisce un'implementazione generica di OpenGL$_G$ per la resa grafica tridimensionale.

Vantaggi:

- La versione attuale di Mesa 3D è disponibile e compilabile per tutte le piattaforme moderne;
- Viene costantemente aggiornata per mantenere allineate le API con l'ultima versione degli standard OpenGL$_G$;
- Supporta diversi acceleratori grafici;
- Può anche essere compilato come un render di tipo software.

2.5 Assimp

Assimp è l'abbreviazione di Open Asset Import Library. È una libreria Open Source scritta in C++ in grado di importare in modo uniforme molti formati per modelli 3D. L'importazione del modello avviene in una struttura dati semplice che può essere successivamente processata.
I formati supportati per l'importazione sono la maggior parte dei formati compatibili con:

- Blender 3D;
- $3ds_G$ Max $3DS_G$.

Tale libreria permette inoltre di esportare i modelli in diversi formati 3D. Può essere quindi utilizzata per creare un convertitore generale.

2.6 ubjson-cpp

È stata utilizzata la libreria ubjson-cpp, disponibile all'indirizzo https://github.com/dcoded/ubjson-cpp, per scrivere il codice necessario all'esportazione nel formato UBJSON.

3 Descrizione architettura

3.1 Metodo e formalismo di specifica

Nell'esposizione dell'architettura dell'applicazione si procederà con un approccio top-down$_G$, descrivendo l'architettura iniziando dal generale ed andando al particolare.
Si procederà quindi alla descrizione dei package$_G$ e dei componenti, per poi descrivere nel dettaglio le singole classi, specificando per ognuna il tipo, l'obiettivo, la funzione e le relazioni in ingresso ed in uscita.
Successivamente si illustreranno degli esempi di uso dei Design Pattern$_G$ nell'architettura del sistema, rimandando la spiegazione generale degli stessi all'appendice A.1.
I diagrammi di package$_G$ e di classe utilizzano il formalismo UML$_G$ 2.0. I diagrammi di attività e di sequenza utilizzano il formalismo UML$_G$ 2.4. Tali adozioni derivano dall'utilizzo di editor UML$_G$ diversi, come descritto nelle *Norme di Progetto v4.2.0*.
Nel riportare i diagrammi di package$_G$ e di classe si farà uso, dove appropriato, dei colori, per aiutare la distinzione tra componenti diversi. Si noti in particolare che le classi di colore verde appartengono a Qt$_G$ e sono quindi da considerarsi fuori dai package$_G$ e componenti definiti nell'architettura, anche se riportate all'interno in alcuni diagrammi per maggior chiarezza.
Nel trattare i componenti, si chiarisce che sono da intendersi come package$_G$ e i due termini verranno quindi usati come sinonimi.
Le classi astratte potrebbero non contenere metodi astratti in quanto questi verranno specificati durante la **Progettazione di Dettaglio**.

3.2 Architettura generale

L'architettura del software segue il Design Pattern$_G$ MVC ed è quindi suddivisa in: Model, View e Controller.
Il diagramma seguente presenta l'architettura ad alto livello dell'applicazione indicandone i package$_G$ e le relazioni tra questi. Un visione ancor più ad alto livello si trova nel componente .

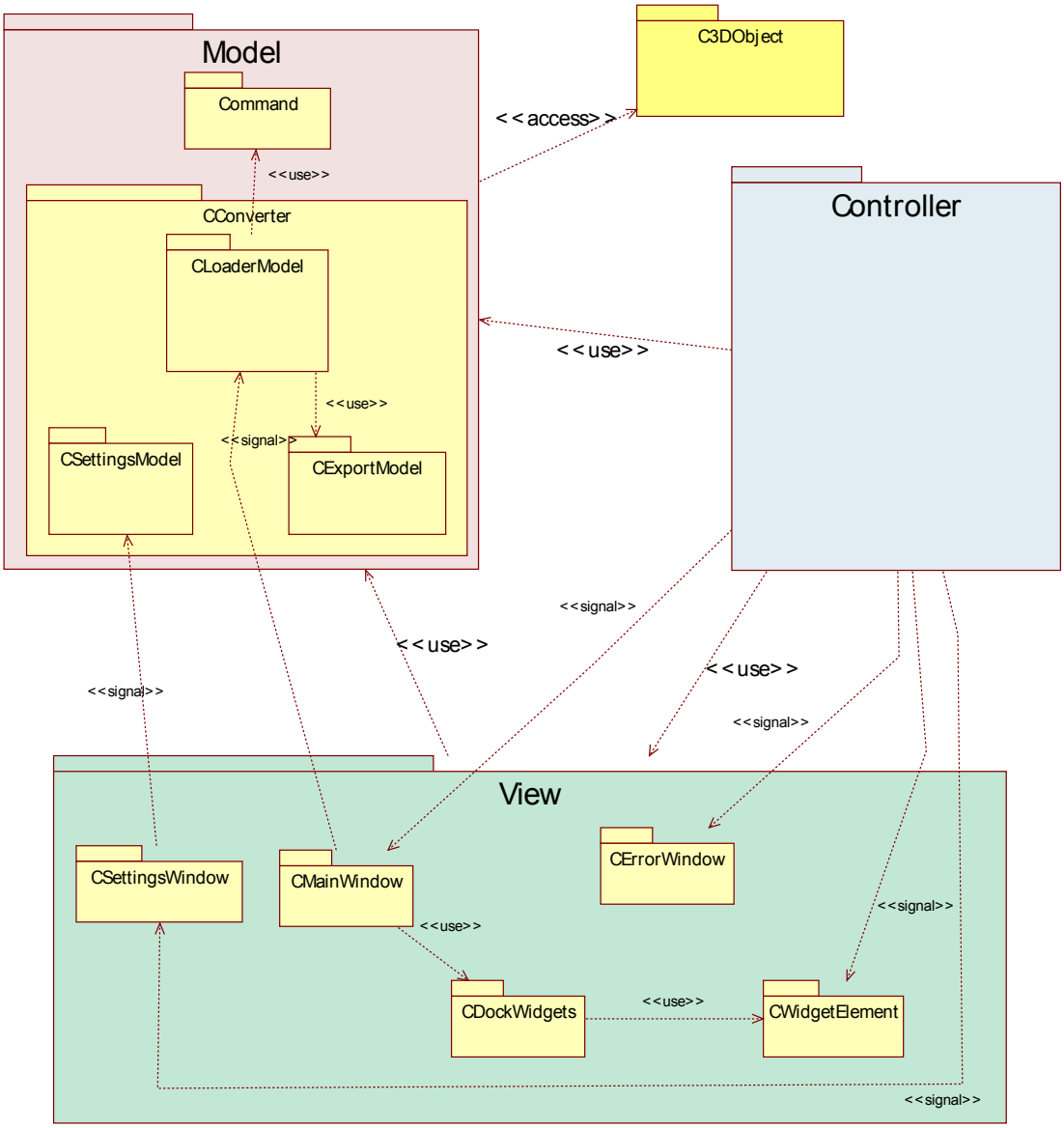

Figura 1: Architettura generale dell'applicazione - vista package

Nel diagramma 1 sono presentate le relazioni tra i package$_G$ Model, View e Controller che evidenziano le relazioni del Design Pattern$_G$ MVC.
Vengono inoltre presentati tutti i sotto-package$_G$ così da facilitare la comprensione dell'intero sistema.

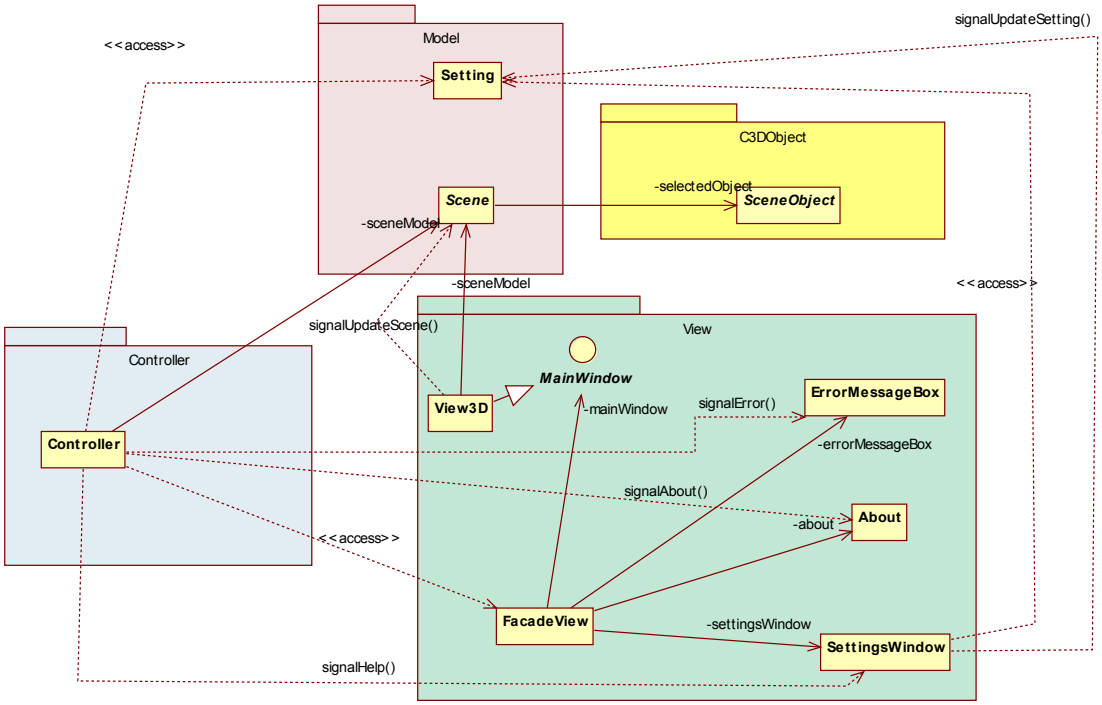

Figura 2: Architettura generale dell'applicazione

Nel diagramma 2 presenta l'architettura ad alto livello dell'applicazione e vengono indicate le classi fondamentali per rappresentare le relazioni del pattern MVC.
Il Design Pattern$_G$ MVC è decritto ampiamente nella sezione A.1.1. I diagrammi di sequenza relativi allo scambio di segnali, lo scopo ed il contesto di utilizzo sono presenti nella sezione 6.1.1.
I segnali vengono indicati nei diagrammi indicando gli eventuali parametri passati.

3.2.1 Model

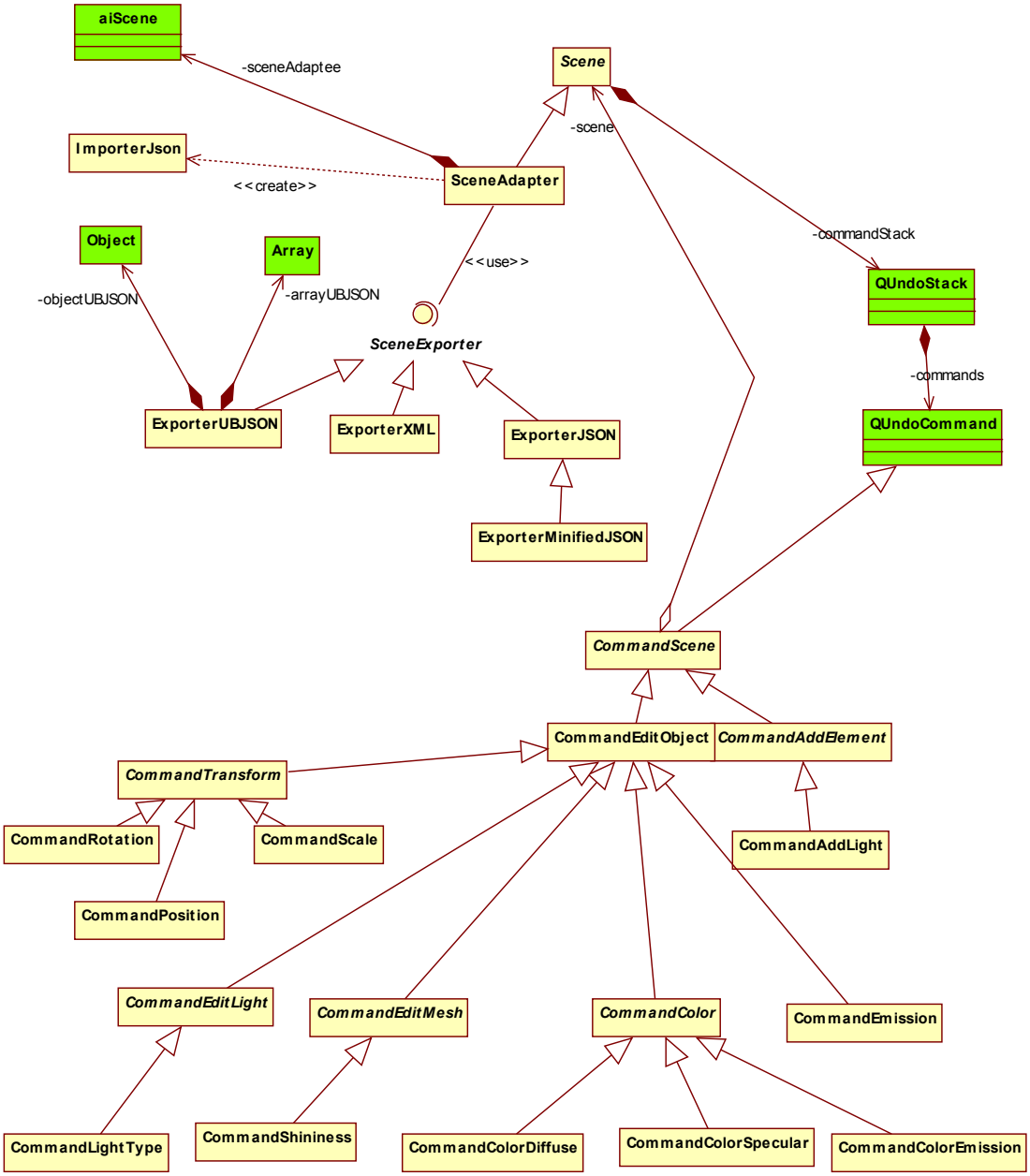

Figura 3: Diagramma delle classi del Model

Nel Model sono presenti gli oggetti che rappresentano:

- La scena$_G$ 3D caricata in memoria;
- La lista dei comandi eseguiti sulla scena$_G$;
- I dati sulle impostazioni del programma;
- I contenuti del sistema di aiuto.

Mette inoltre a disposizione tutti i metodi per modificare gli oggetti in esso contenuti.
Manda segnali alla View quando vi è un aggiornamento dei dati in esso contenuti.
In caso si verifichino errori, questi vengono gestiti dal Model e segnalati alla View.

3.2.2 View

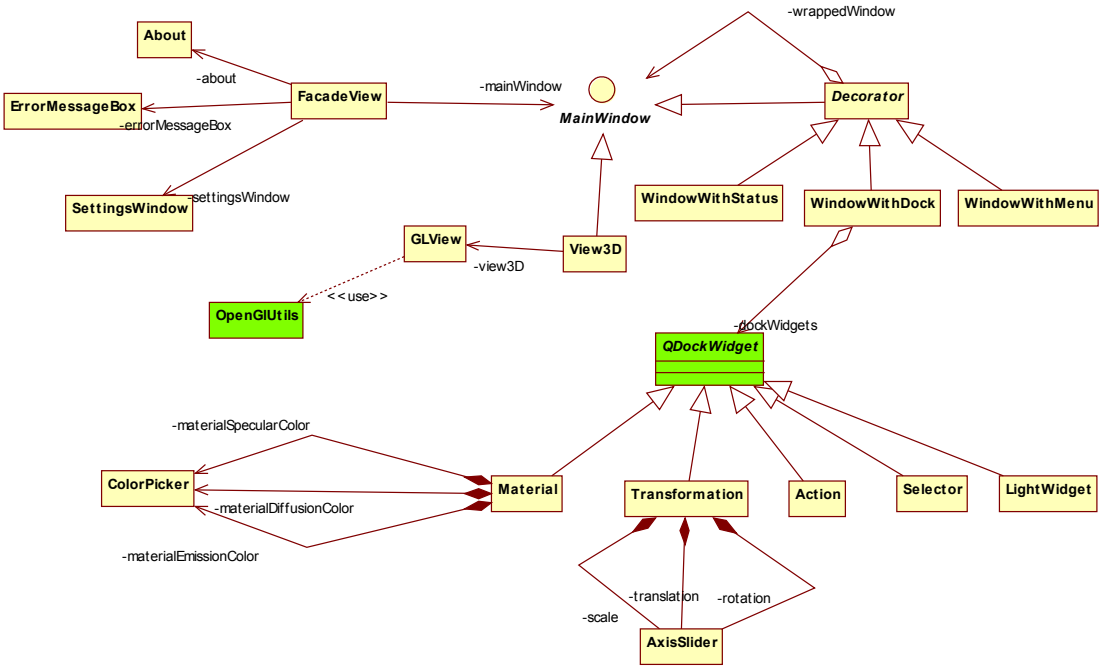

Figura 4: Diagramma delle classi del View

Nella View sono presenti le finestre che permettono all'utente di interagire con il programma.
Tutti i componenti il cui scopo è modificare dati nel Model mandano segnali al Controller. Sarà questo ad invocare i metodi del Model per effettuare l'aggiornamento o la richiesta dei dati.
I componenti delle View che devono essere aggiornati dopo la modifica ricevono un segnale dal Model e da questo vanno a prendere i dati aggiornati.

3.2.3 Controller

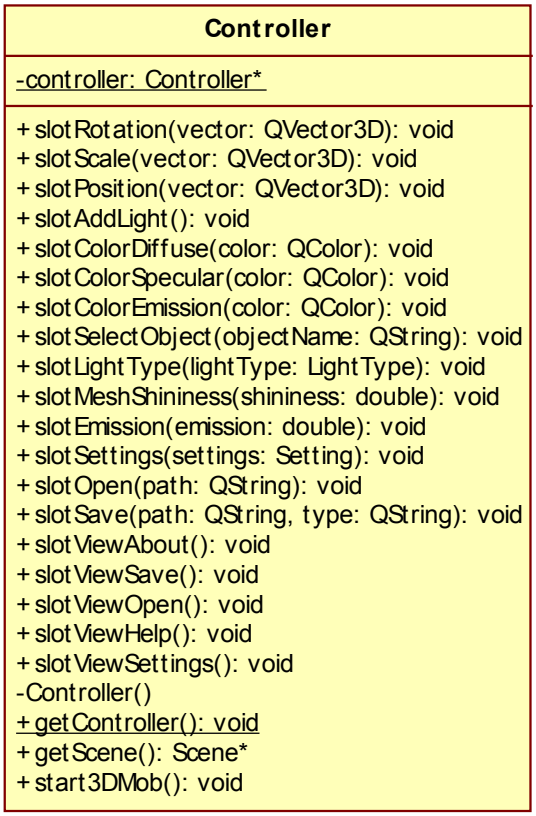

Figura 5: Diagramma delle classi del Controller

Nel Controller è presente la classe che riceve tutti i Signal_G emessi dalla View e che decide che azioni intraprendere:

- **View:** il Controller può agire sulla View e aprire nuove finestre;
- **Model:** il Controller può creare un nuovo comando e inviarlo al Model.

3.2.4 C3DObject

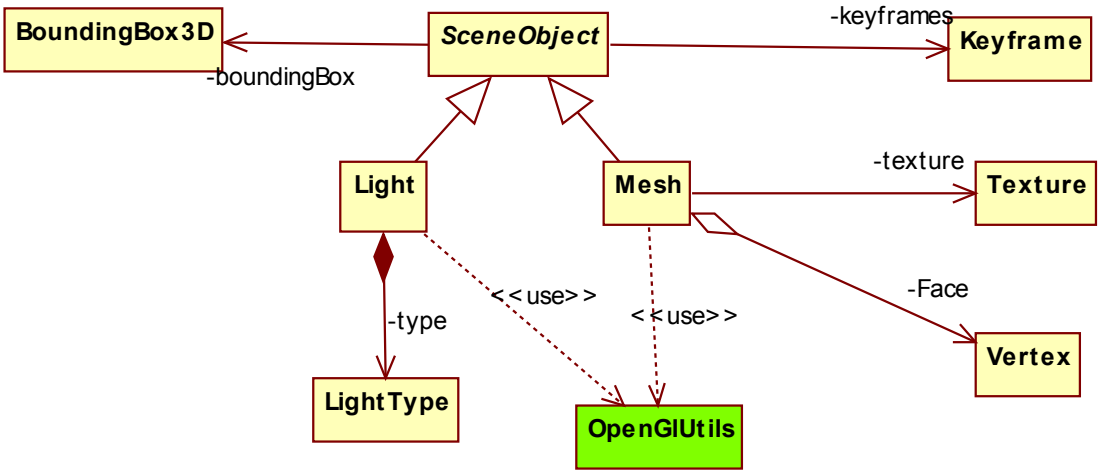

Figura 6: Diagramma delle classi di C3DObject

C3DObject è un package$_G$ di utilità usato dal Model per rappresentare i suoi oggetti.

4 Componenti e Classi

4.1 DDDMob

4.1.1 Informazioni sul package

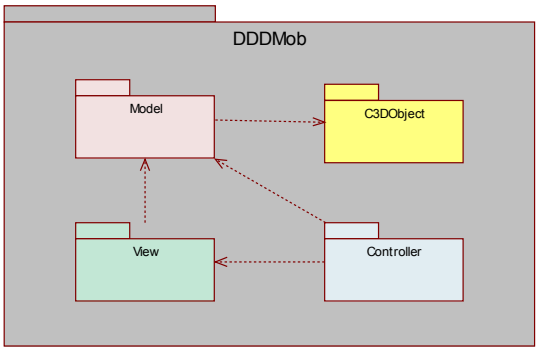

Figura 7: Componente DDDMob

4.1.1.1 Descrizione

Namespace$_G$ globale per il progetto. Le relazioni tra i package$_G$ Model, View e Controller identificano le relazioni tipiche che intercorrono tra le componenti del Design Pattern$_G$ MVC.

4.1.1.2 Package contenuti

- DDDMob::Model;
- DDDMob::Controller;
- DDDMob::View;
- DDDMob::C3DObject.

4.2 DDDMob::Model

4.2.1 Informazioni sul package

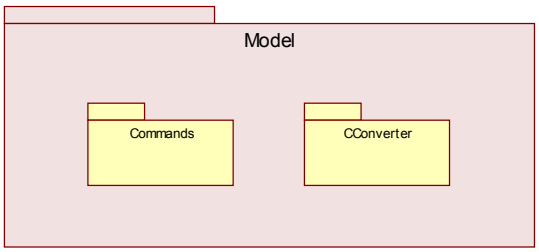

Figura 8: Componente DDDMob::Model

4.2.1.1 Descrizione

Package$_G$ per il componente Model dell'architettura MVC.

4.2.1.2 Package contenuti

- DDDMob::Model::Commands;
- DDDMob::Model::CConverter.

4.3 DDDMob::Model::Commands

4.3.1 Informazioni sul package

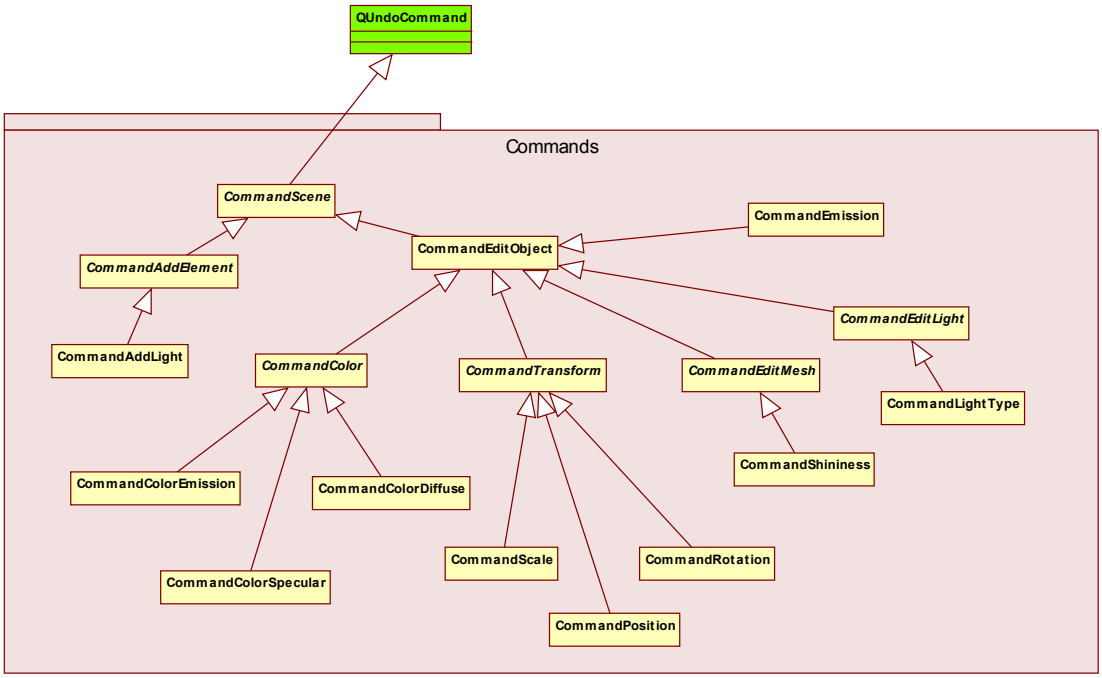

Figura 9: Componente DDDMob::Model::Commands

4.3.1.1 Descrizione

Componente parte del Model per la gestione dei comandi.

4.3.1.2 Interazioni con altri componenti

- DDDMob::Model::CConverter::CLoaderModel;
- DDDMob::C3DObject;
- Qt_G.

4.3.2 Classi

4.3.2.1 DDDMob :: Model :: Commands :: CommandTransform

Descrizione

Classe base per i comandi di trasformazione;

Utilizzo

Viene utilizzata per applicare un generico parametro di trasformazione ad un oggetto della scena$_G$ 3D, specificato poi nelle classi che ereditano da questa;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Classi figlie

- DDDMob :: Model :: Commands :: CommandPosition;
- DDDMob :: Model :: Commands :: CommandScale;
- DDDMob :: Model :: Commands :: CommandRotation.

4.3.2.2 DDDMob :: Model :: Commands :: CommandAddLight

Descrizione

Classe che rappresenta il comando di aggiunta di una luce con i parametri di default alla scena$_G$ 3D;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la creazione di una luce ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandAddElement.

4.3.2.3 DDDMob :: Model :: Commands :: CommandEditMesh

Descrizione

Classe che rappresenta il comando di modifica di una mesh;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica di una mesh ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Classi figlie

- DDDMob :: Model :: Commands :: CommandShininess.

4.3.2.4 DDDMob :: Model :: Commands :: CommandEditLight

Descrizione

Classe che rappresenta il comando di modifica della luce selezionata;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della luce selezionata ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Classi figlie

- DDDMob :: Model :: Commands :: CommandLightType.

4.3.2.5 DDDMob :: Model :: Commands :: CommandScene

Descrizione

Classe che rappresenta un generico comando da eseguire sulla scena$_G$. È il padre di tutti i possibili componenti concrete command del Design Pattern$_G$ Command;

Utilizzo

Viene utilizzata per applicare un generico comando da eseguire sulla scena$_G$ 3D, specificato poi nelle classi che ereditano da questa;

Classi ereditate

- QUndoCommand.

Classi figlie

- DDDMob :: Model :: Commands :: CommandAddElement;
- DDDMob :: Model :: Commands :: CommandEditObject.

Relazioni con altre classi

DDDMob::Model::CConverter::CLoaderModel::Scene

Relazione uscente, riferimento alla scena$_G$ del modello.

4.3.2.6 DDDMob :: Model :: Commands :: CommandAddElement

Descrizione

Classe che rappresenta un comando di aggiunta di un elemento alla scena$_G$ 3D;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti l'aggiunta di un elemento ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandScene.

Classi figlie

- DDDMob :: Model :: Commands :: CommandAddLight.

Relazioni con altre classi

DDDMob::C3DObject::SceneObject

Relazione uscente, oggetto della scena$_G$.

4.3.2.7 DDDMob :: Model :: Commands :: CommandPosition

Descrizione

Classe che rappresenta un comando per cambiare la posizione dell'oggetto della scena$_G$ 3D selezionato;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica della posizione di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

4.3.2.8 DDDMob :: Model :: Commands :: CommandScale

Descrizione

Classe che rappresenta un comando per cambiare la dimensione dell'oggetto della scena$_G$ 3D selezionato;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la dimensione di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

4.3.2.9 DDDMob :: Model :: Commands :: CommandRotation

Descrizione

Classe che rappresenta un comando per cambiare la rotazione del modello;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandTransform.

4.3.2.10 DDDMob :: Model :: Commands :: CommandLightType

Descrizione

Classe che rappresenta un comando per cambiare il tipo della luce selezionata;

Utilizzo

Viene utilizzata per gestire i Signal_G riguardanti la modifica della tipologia di una luce ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditLight.

Relazioni con altre classi

DDDMob::C3DObject::LightType

Relazione uscente, nuovo tipo di luce;

DDDMob::C3DObject::LightType

Relazione uscente, vecchio tipo di luce.

4.3.2.11 DDDMob :: Model :: Commands :: CommandShininess

Descrizione

Classe che rappresenta il comando che cambia il valore di lucentezza della mesh;

Utilizzo

Viene utilizzata per gestire i Signal_G riguardanti la modifica della lucentezza di una mesh ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditMesh.

4.3.2.12 DDDMob :: Model :: Commands :: CommandEmission

Descrizione

Classe che rappresenta il comando per cambiare il valore della quantità di emissione del materiale di un oggetto;

Utilizzo

Viene utilizzata per gestire i Signal_G riguardanti la quantità di emissione del materiale di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

4.3.2.13 DDDMob :: Model :: Commands :: CommandColor

Descrizione

Classe che rappresenta il comando per cambiare le caratteristiche legate al colore dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandEditObject.

Classi figlie

- DDDMob :: Model :: Commands :: CommandColorEmission;
- DDDMob :: Model :: Commands :: CommandColorSpecular;
- DDDMob :: Model :: Commands :: CommandColorDiffuse.

Relazioni con altre classi

QColor

Relazione uscente, colore vecchio dell'oggetto;

QColor

Relazione uscente, rappresenta il colore nuovo dell'oggetto.

4.3.2.14 DDDMob :: Model :: Commands :: CommandColorEmission

Descrizione

Classe che rappresenta il comando per cambiare il colore di emissione dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di emissione dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

4.3.2.15 DDDMob :: Model :: Commands :: CommandColorSpecular

Descrizione

Classe che rappresenta il comando per cambiare il colore di diffusione$_G$ dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di diffusione$_G$ dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

4.3.2.16 DDDMob :: Model :: Commands :: CommandColorDiffuse

Descrizione

Comando per cambiare il colore di diffusione$_G$ dell'oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti il colore di diffusione$_G$ dell'oggetto;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandColor.

4.3.2.17 DDDMob :: Model :: Commands :: CommandEditObject

Descrizione

Classe che rappresenta il comando di modifica di un oggetto;

Utilizzo

Viene utilizzata per gestire i Signal$_G$ riguardanti la modifica di un oggetto ed invocare i corretti metodi del Model;

Classi ereditate

- DDDMob :: Model :: Commands :: CommandScene.

Classi figlie

- DDDMob :: Model :: Commands :: CommandTransform;
- DDDMob :: Model :: Commands :: CommandEditMesh;
- DDDMob :: Model :: Commands :: CommandEditLight;
- DDDMob :: Model :: Commands :: CommandEmission;
- DDDMob :: Model :: Commands :: CommandColor.

Relazioni con altre classi

DDDMob::C3DObject::SceneObject

Relazione uscente, riferimento all'oggetto della scena$_G$.

4.4 DDDMob::Model::CConverter

4.4.1 Informazioni sul package

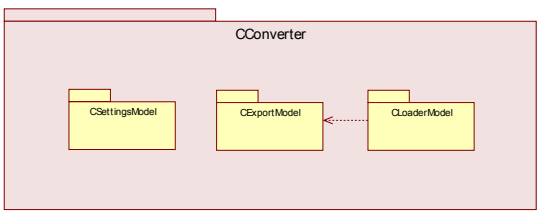

Figura 10: Componente DDDMob::Model::CConverter

4.4.1.1 Descrizione

Macro componente per l'integrazione delle funzionalità di importazione, applicazione dei limiti impostati ed esportazione a formare il convertitore di base.

4.4.1.2 Package contenuti

- DDDMob::Model::CConverter::CSettingsModel;
- DDDMob::Model::CConverter::CExportModel;
- DDDMob::Model::CConverter::CLoaderModel.

4.5 DDDMob::Model::CConverter::CSettingsModel

4.5.1 Informazioni sul package

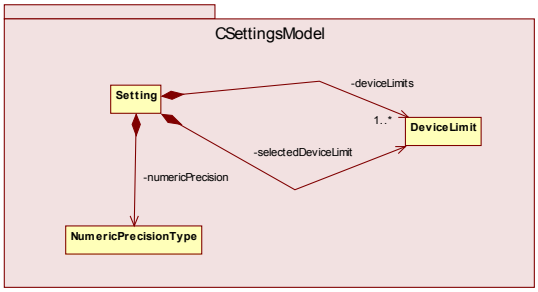

Figura 11: Componente DDDMob::Model::CConverter::CSettingsModel

4.5.1.1 Descrizione

Componente parte del Model per le funzionalità di impostazioni e configurazione.

4.5.2 Classi

4.5.2.1 DDDMob :: Model :: CConverter :: CSettingsModel :: Setting

Descrizione

Classe implementata con il Design Pattern$_G$ Singleton che contiene tutte le impostazioni del programma;

Utilizzo

Viene utilizzata per modificare e recuperare i limiti di importazione, precisione di esportazione e colore di sfondo. Riceve richieste di modifica di alcuni elementi o di restituzione dei dati, e notifica alla vista quando gli stessi sono disponibili o aggiornati;

Relazioni con altre classi

DDDMob::Model::CConverter::CSettingsModel::DeviceLimit
Relazione uscente, limiti del dispositivo selezionato;

DDDMob::Model::CConverter::CSettingsModel::DeviceLimit
Relazione uscente, limiti dei dispositivi predefiniti;

...Model::CConverter::CSettingsModel::NumericPrecisionType
Relazione uscente, precisione numerica dell'oggetto esportato.

4.5.2.2 DDDMob :: Model :: CConverter :: CSettingsModel :: DeviceLimit

Descrizione
Classe che rappresenta i limiti di un dispositivo;

Utilizzo
Viene utilizzata per salvare i dati che rappresentano i limiti di esportazione relativi a un dispositivo;

Relazioni con altre classi

DDDMob::Model::CConverter::CSettingsModel::Setting
Relazione entrante, limiti del dispositivo selezionato;

DDDMob::Model::CConverter::CSettingsModel::Setting
Relazione entrante, limiti dei dispositivi predefiniti.

4.5.2.3 DDDMob :: Model :: CConverter :: CSettingsModel :: NumericPrecisionType

Descrizione
Classe che contiene le diverse tipologie di precisione dei numeri;

Utilizzo
Viene utilizzata come `Enumerate`;

Relazioni con altre classi

DDDMob::Model::CConverter::CSettingsModel::Setting
Relazione entrante, precisione numerica dell'oggetto esportato.

4.6 DDDMob::Model::CConverter::CExportModel

4.6.1 Informazioni sul package

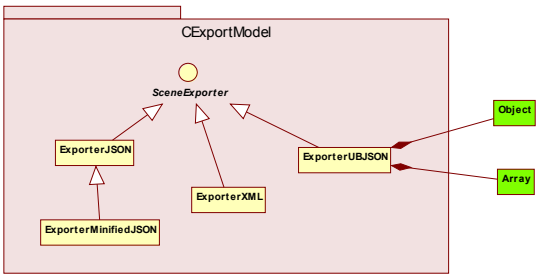

Figura 12: Componente DDDMob::Model::CConverter::CExportModel

4.6.1.1 Descrizione

Componente parte del Model per le funzionalità di esportazione.

4.6.1.2 Interazioni con altri componenti

- Qt_G.

4.6.2 Classi

4.6.2.1 DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter

Descrizione

Interfaccia per la componente strategy del Design Pattern$_G$ Strategy per la selezione dell'algoritmo di esportazione della scena$_G$ tra tutti i disponibili;

Utilizzo

Permette di selezionare dinamicamente ed in modo estensibile l'algoritmo di esportazione della scena$_G$ e di conseguenza il formato del file salvato in uscita;

Classi figlie

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterUBJSON;
- DDDMob :: Model :: CConverter :: CExportModel :: ExporterXML;
- DDDMob :: Model :: CConverter :: CExportModel :: ExporterJSON.

4.6.2.2 DDDMob :: Model :: CConverter :: CExportModel :: ExporterUBJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato UBJSON. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file UBJSON a partire dalla scena$_G$ 3D;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

Relazioni con altre classi

Array

Relazione uscente, array contenente gli elementi della scena$_G$ da serializzare;

Object

Relazione uscente, oggetto contenente le animazioni della scena$_G$ da serializzare.

4.6.2.3 DDDMob :: Model :: CConverter :: CExportModel :: ExporterXML

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato XML$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file XML$_G$ a partire dalla scena$_G$ 3D;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

4.6.2.4 DDDMob :: Model :: CConverter :: CExportModel :: ExporterMinifiedJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato JSON$_G$ che contiene JSON$_G$ minificato$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file JSON$_G$ che contiene JSON$_G$ minificato$_G$ a partire dalla scena$_G$ 3D;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterJSON.

4.6.2.5 DDDMob :: Model :: CConverter :: CExportModel :: ExporterJSON

Descrizione

Classe che rappresenta un algoritmo di esportazione della scena$_G$ in un file di formato JSON$_G$. È uno dei componenti concrete component del Design Pattern$_G$ Strategy;

Utilizzo

Viene utilizzata per creare un file JSON$_G$ a partire dalla scena$_G$ 3D;

Classi ereditate

- DDDMob :: Model :: CConverter :: CExportModel :: SceneExporter.

Classi figlie

- DDDMob :: Model :: CConverter :: CExportModel :: ExporterMinifiedJSON.

4.7 DDDMob::Model::CConverter::CLoaderModel

4.7.1 Informazioni sul package

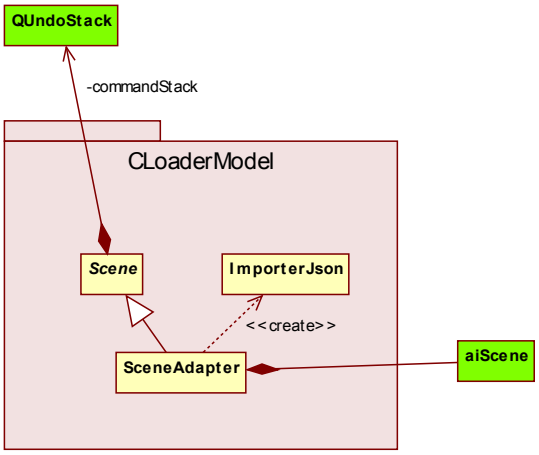

Figura 13: Componente DDDMob::Model::CConverter::CLoaderModel

4.7.1.1 Descrizione

Componente parte del Model per le funzionalità di caricamento del file nella scena$_G$.

4.7.1.2 Interazioni con altri componenti

- DDDMob::Controller;
- Qt$_G$;
- DDDMob::C3DObject;
- DDDMob::Model::Commands;
- DDDMob::View::CMainWindow.

4.7.2 Classi

4.7.2.1 DDDMob :: Model :: CConverter :: CLoaderModel :: Scene

Descrizione

Classe che rappresenta la scena$_G$ 3D che contiene le mesh e le luci. È il componente receiver del Design Pattern$_G$ Command, componente context del Design Pattern$_G$ Strategy e componente target del Design Pattern$_G$ Adapter. Internamente contiene la lista di comandi eseguiti sulla scena$_G$ 3D;

Utilizzo

Interfaccia per gestire le proprietà degli oggetti 3D. Permette la selezione di un oggetto della scena$_G$ e la sua modifica. Contiene lo stato dei comandi eseguiti sulla scena$_G$ così da poter annullare e ripristinare le modifiche effettuate. Permette l'esportazione in vari formati, selezionati grazie al Design Pattern$_G$ Strategy, e l'importazione nei vari formati permessi dalla libreria esterna utilizzata;

Classi figlie

- DDDMob :: Model :: CConverter :: CLoaderModel :: SceneAdapter.

Relazioni con altre classi

DDDMob::Model::Commands::CommandScene
Relazione entrante, riferimento alla scena$_G$ del modello;

DDDMob::Controller::Controller
Relazione entrante, riferimento alla scena$_G$ del modello;

QUndoStack
Relazione uscente, lista delle azioni effettuate;

DDDMob::View::CMainWindow::View3D
Relazione entrante, riferimento alla scena$_G$ del Modello;

DDDMob::C3DObject::SceneObject
Relazione uscente, l'oggetto 3D attualmente selezionato.

4.7.2.2 DDDMob :: Model :: CConverter :: CLoaderModel :: SceneAdapter

Descrizione
Classe che viene utilizzata come adattatore per la libreria esterna Assimp. Rappresenta il componente adapter del Design Pattern$_G$ Adapter;

Utilizzo
Viene utilizzata come adattatore tra quanto esposto dalla libreria esterna Assimp e la classe Scene;

Classi ereditate

- DDDMob :: Model :: CConverter :: CLoaderModel :: Scene.

Relazioni con altre classi

DDDMob::C3DObject::SceneObject
Relazione uscente, mappa tra i nomi e i riferimenti agli oggetti contenuti nella scena$_G$;

aiScene
Relazione uscente, riferimento alla struttura dati di Assimp che rappresenta la scena$_G$ da adattare.

4.7.2.3 DDDMob :: Model :: CConverter :: CLoaderModel :: ImporterJson

Descrizione
Classe che rappresenta un algoritmo di importazione della scena$_G$ in un file di formato JSON$_G$;

Utilizzo
Viene utilizzata per creare tutti gli oggetti della scena$_G$ a partire da un file JSON$_G$.

4.8 DDDMob::Controller

4.8.1 Informazioni sul package

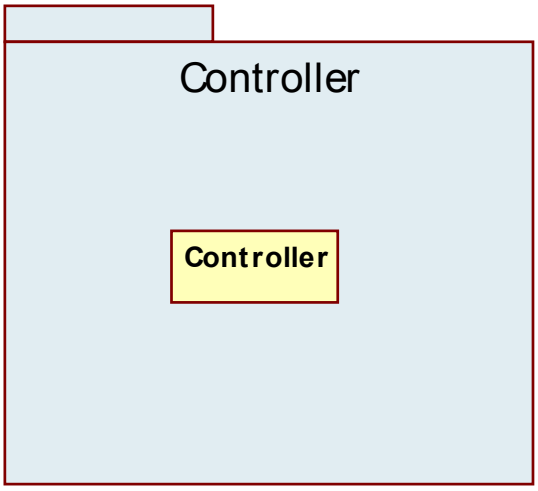

Figura 14: Componente DDDMob::Controller

4.8.1.1 Descrizione

Package$_G$ per il componente Controller dell'architettura MVC.

4.8.1.2 Interazioni con altri componenti

- DDDMob::Model::CConverter::CLoaderModel.

4.8.2 Classi

4.8.2.1 DDDMob :: Controller :: Controller

Descrizione

Classe che rappresenta il componente controller del Design Pattern$_G$ MVC ed il componente client del Design Pattern$_G$ Command;

Utilizzo

Gestisce i Signal$_G$ inviati dalla View ed agisce nel modo corretto. Genera i comandi da eseguire sulla scena$_G$ 3D e memorizza tali comandi nel componente Scene del Model. Concorre nell'applicare le modifiche all'oggetto selezionato della scena$_G$ 3D;

Relazioni con altre classi

DDDMob::Model::CConverter::CLoaderModel::Scene

Relazione uscente, riferimento alla scena$_G$ del modello.

4.9 DDDMob::View

4.9.1 Informazioni sul package

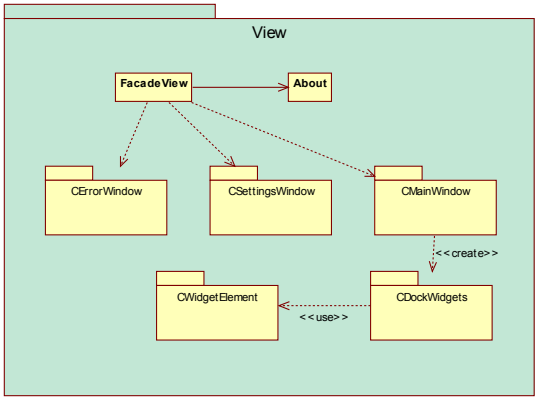

Figura 15: Componente DDDMob::View

4.9.1.1 Descrizione

Package$_G$ per il componente View dell'architettura MVC.

4.9.1.2 Package contenuti

- DDDMob::View::CDockWidgets;
- DDDMob::View::CMainWindow;
- DDDMob::View::CSettingsWindow;
- DDDMob::View::CErrorWindow;
- DDDMob::View::CWidgetElement.

4.9.1.3 Interazioni con altri componenti

- Qt$_G$.

4.9.2 Classi

4.9.2.1 DDDMob :: View :: FacadeView

Descrizione

Classe che rappresenta la Vista. È la componente facade del Design Pattern$_G$ Facade, ed è implementata con il Design Pattern$_G$ Singleton;

Utilizzo

Viene utilizzata per accedere alla vista in modo protetto, in quanto espone i metodi necessari per l'interazione dall'esterno, nascondendo l'implementazione e la struttura;

Relazioni con altre classi

DDDMob::View::CMainWindow::MainWindow

Relazione uscente, riferimento alla finestra principale;

DDDMob::View::CSettingsWindow::SettingsWindow
Relazione uscente, riferimento all'interfaccia per le impostazioni del programma;

DDDMob::View::CErrorWindow::ErrorMessageBox
Relazione uscente, riferimento protetto alla finestra di informazione di errore;

DDDMob::View::About
Relazione uscente, riferimento alla finestra di dialogo con le informazioni sull'applicazione.

4.9.2.2 DDDMob :: View :: About

Descrizione
Classe che rappresenta la finestra contenente le informazioni sul programma e sui suoi creatori;

Utilizzo
Viene utilizzata per visualizzare le informazioni sul programma e sui suoi creatori, oltre che la versione del prodotto e delle librerie utilizzate;

Classi ereditate
- QDialog.

Relazioni con altre classi

DDDMob::View::FacadeView
Relazione entrante, riferimento alla finestra di dialogo con le informazioni sull'applicazione.

4.10 DDDMob::View::CDockWidgets

4.10.1 Informazioni sul package

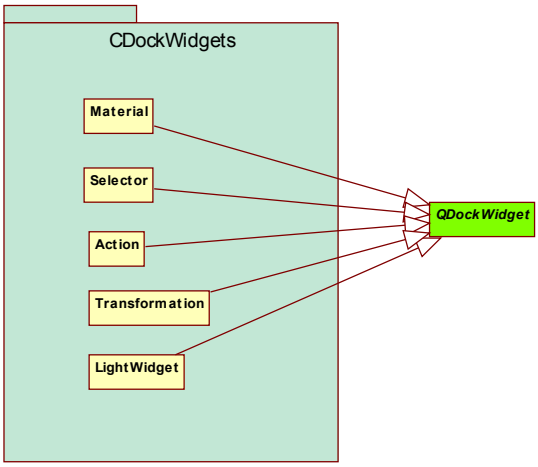

Figura 16: Componente DDDMob::View::CDockWidgets

4.10.1.1 Descrizione

Componente che contiene tutti i tipi di dockwidget disponibili nel sistema.

4.10.1.2 Interazioni con altri componenti

- DDDMob::View::CWidgetElement;
- Qt_G.

4.10.2 Classi

4.10.2.1 DDDMob :: View :: CDockWidgets :: Material

Descrizione

Classe che rappresenta il widget contenente le informazioni sul materiale;

Utilizzo

Viene utilizzata per permettere di cambiare le caratteristiche del materiale dell'oggetto. Emette un $Signal_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Relazioni con altre classi

DDDMob::View::CWidgetElement::ColorPicker

Relazione uscente, elemento grafico che permette di modificare il colore di $diffusione_G$ del materiale;

DDDMob::View::CWidgetElement::ColorPicker

Relazione uscente, elemento grafico che permette di modificare il $colore\ speculare_G$ del materiale;

DDDMob::View::CWidgetElement::ColorPicker

Relazione uscente, elemento grafico che permette di modificare il colore di emissione del materiale.

4.10.2.2 DDDMob :: View :: CDockWidgets :: Selector

Descrizione

Classe che rappresenta il widget contenente il selettore di oggetti;

Utilizzo

Viene utilizzata per selezionare un oggetto della $scena_G$. Emette un $Signal_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Relazioni con altre classi

QListWidget

Relazione uscente, elemento grafico che permette di selezionare un oggetto.

4.10.2.3 DDDMob :: View :: CDockWidgets :: LightWidget

Descrizione

Classe che rappresenta il widget che permette di interagire con le luci della scena$_G$ e di aggiungerne di nuove;

Utilizzo

Viene utilizzata per interagire con le luci della scena$_G$ o aggiungerne una nuova. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

4.10.2.4 DDDMob :: View :: CDockWidgets :: Action

Descrizione

Classe che rappresenta il widget che consente di visualizzare le azioni effettuate, di annullarle e ripristinarle;

Utilizzo

Viene utilizzata per visualizzare le azioni effettuate, annullarle e ripristinarle. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Relazioni con altre classi

QUndoView

Relazione uscente, elemento grafico che permette di visualizzare le operazioni effettuate.

4.10.2.5 DDDMob :: View :: CDockWidgets :: Transformation

Descrizione

Classe che rappresenta un widget contenente gli slider per modificare alcune caratteristiche dell'oggetto;

Utilizzo

Viene utilizzata per permette di applicare una trasformazione alla scena$_G$. Emette un Signal$_G$ in seguito all'interazione dell'utente con l'elemento dell'interfaccia grafica;

Classi ereditate

- QDockWidget.

Relazioni con altre classi

DDDMob::View::CWidgetElement::AxisSlider
Relazione uscente, elemento grafico che permette di modificare la rotazione dell'elemento;

DDDMob::View::CWidgetElement::AxisSlider
Relazione uscente, elemento grafico che permette di modificare la dimensione;

DDDMob::View::CWidgetElement::AxisSlider
Relazione uscente, elemento grafico che permette di modificare la traslazione dell'elemento.

4.11 DDDMob::View::CMainWindow

4.11.1 Informazioni sul package

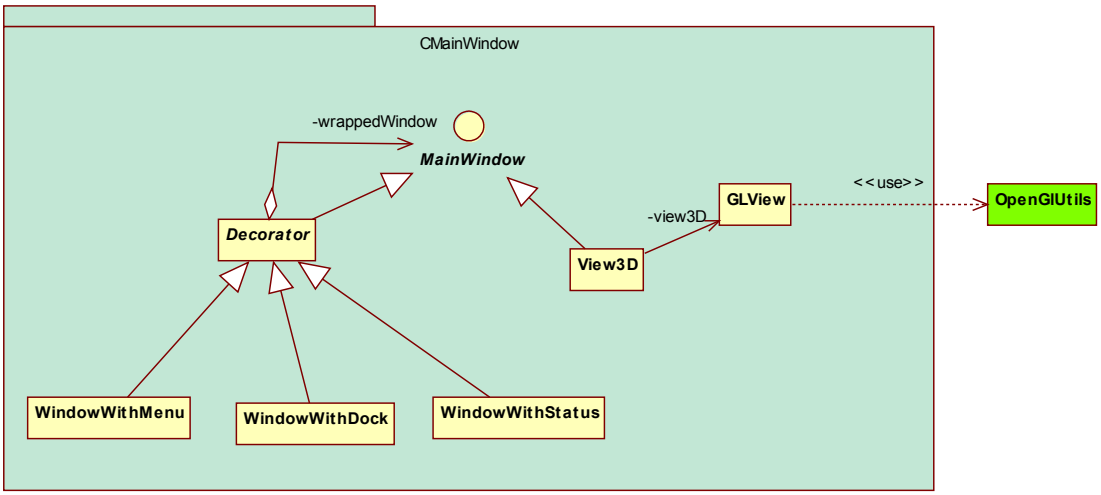

Figura 17: Componente DDDMob::View::CMainWindow

4.11.1.1 Descrizione

Componente parte di View per la finestra principale dell'applicazione.

4.11.1.2 Interazioni con altri componenti

- DDDMob::View;
- Qt_G;
- DDDMob::Model::CConverter::CLoaderModel.

4.11.2 Classi

4.11.2.1 DDDMob :: View :: CMainWindow :: WindowWithMenu

Descrizione
Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern_G Decorator;

Utilizzo
Viene utilizzata per aggiungere un menù alla finestra;

Classi ereditate
- QMenuBar;
- DDDMob :: View :: CMainWindow :: Decorator.

Relazioni con altre classi

QMenu
Relazione uscente, menù nella barra dei menù.

4.11.2.2 DDDMob :: View :: CMainWindow :: WindowWithDock

Descrizione
Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern$_G$ Decorator;

Utilizzo
Decoratore che aggiunge elementi ancorabili alla finestra;

Classi ereditate
- DDDMob :: View :: CMainWindow :: Decorator.

Relazioni con altre classi

QDockWidget
Relazione uscente, rappresenta i dockWidgets presenti nella finestra.

4.11.2.3 DDDMob :: View :: CMainWindow :: WindowWithStatus

Descrizione
Classe che decora la finestra principale. Rappresenta uno dei componenti concrete decorator del Design Pattern$_G$ Decorator;

Utilizzo
Viene utilizzata per aggiungere ad una finestra la barra di stato;

Classi ereditate
- QStatusBar;
- DDDMob :: View :: CMainWindow :: Decorator.

4.11.2.4 DDDMob :: View :: CMainWindow :: Decorator

Descrizione

Classe astratta che rappresenta le varie decorazioni applicabili all'interfaccia utente. Rappresenta il componente decorator del Design Pattern$_G$ Decorator;

Utilizzo

Classe che rappresenta le possibili decorazioni applicabili all'interfaccia utente, specificate poi dalle classi che ereditano da questa;

Classi ereditate

- DDDMob :: View :: CMainWindow :: MainWindow.

Classi figlie

- DDDMob :: View :: CMainWindow :: WindowWithMenu;
- DDDMob :: View :: CMainWindow :: WindowWithDock;
- DDDMob :: View :: CMainWindow :: WindowWithStatus.

Relazioni con altre classi

DDDMob::View::CMainWindow::MainWindow

Relazione uscente, puntatore alla composizione di finestre che si vuole decorare con la classe.

4.11.2.5 DDDMob :: View :: CMainWindow :: MainWindow

Descrizione

Interfaccia che rappresenta la finestra principale del programma, ottenuta dalla vista 3D decorata con le decorazioni offerte dalla classe Decorator. Rappresenta il componente component del Design Pattern$_G$ Decorator;

Classi figlie

- DDDMob :: View :: CMainWindow :: Decorator;
- DDDMob :: View :: CMainWindow :: View3D.

Relazioni con altre classi

DDDMob::View::CMainWindow::Decorator

Relazione entrante, puntatore alla composizione di finestre che si vuole decorare con la classe;

DDDMob::View::FacadeView

Relazione entrante, riferimento alla finestra principale.

4.11.2.6 DDDMob :: View :: CMainWindow :: View3D

Descrizione

Classe che rappresenta la vista contenente la scena$_G$ 3D. È il componente concrete component del Design Pattern$_G$ Decorator;

Utilizzo

Viene utilizzata per visualizzare l'anteprima della scena$_G$ 3D del Model. Manda Signal$_G$ di richiesta di aggiornamento dei dati di visualizzazione della scena$_G$ 3D al Model. Può leggere i dati aggiornati direttamente dal Model dopo aver ricevuto un Signal$_G$ di aggiornamento;

Classi ereditate

- QMainWindow;
- DDDMob :: View :: CMainWindow :: MainWindow.

Relazioni con altre classi

DDDMob::Model::CConverter::CLoaderModel::Scene

Relazione uscente, riferimento alla scena$_G$ del Modello;

DDDMob::View::CMainWindow::GLView

Relazione uscente, finestra di visualizzazione della scena$_G$ 3D.

4.11.2.7 DDDMob :: View :: CMainWindow :: GLView

Descrizione

Classe che esegue il rendering$_G$ della scena$_G$;

Relazioni con altre classi

DDDMob::View::CMainWindow::View3D

Relazione entrante, finestra di visualizzazione della scena$_G$ 3D.

4.12 DDDMob::View::CSettingsWindow

4.12.1 Informazioni sul package

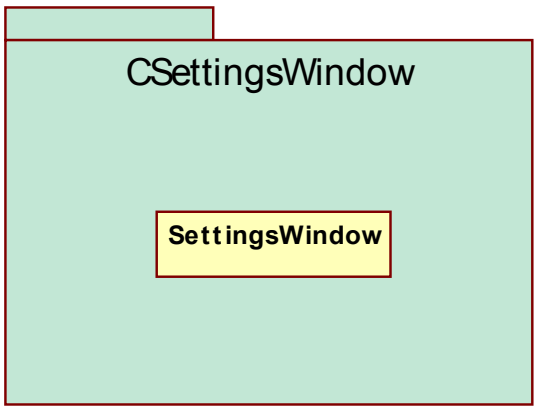

Figura 18: Componente DDDMob::View::CSettingsWindow

4.12.1.1 Descrizione

Componente parte di View per la finestra di configurazione delle impostazioni del programma.

4.12.1.2 Interazioni con altri componenti

- DDDMob::View;
- Qt_G.

4.12.2 Classi

4.12.2.1 DDDMob :: View :: CSettingsWindow :: SettingsWindow

Descrizione

Classe che rappresenta la finestra per modificare le impostazioni del programma;

Utilizzo

Viene utilizzata per visualizzare le impostazioni del programma. La sua apertura deriva dalla ricezione di un $Signal_G$ da parte del Controller. Manda $Signal_G$ di richiesta di aggiornamento dei dati di configurazione al Model. Può leggere i dati aggiornati direttamente dal Model dopo aver ricevuto un $Signal_G$ di aggiornamento;

Classi ereditate

- QWindow.

Relazioni con altre classi

DDDMob::View::FacadeView

Relazione entrante, riferimento all'interfaccia per le impostazioni del programma.

4.13 DDDMob::View::CErrorWindow

4.13.1 Informazioni sul package

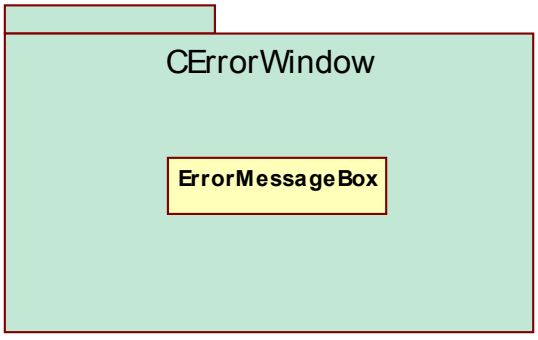

Figura 19: Componente DDDMob::View::CErrorWindow

4.13.1.1 Descrizione

Componente parte di View per la finestra di segnalazione degli errori.

4.13.1.2 Interazioni con altri componenti

- DDDMob::View.

4.13.2 Classi

4.13.2.1 DDDMob :: View :: CErrorWindow :: ErrorMessageBox

Descrizione

Classe che rappresenta una finestra che riporta messaggi d'errore per l'utente;

Utilizzo

La sua apertura deriva dalla ricezione di un Signal$_G$ da parte del Model, generato nel caso in cui si verifichi una inconsistenza durante l'azione del Model. Il Signal$_G$ include un codice di errore e visualizza il relativo messaggio di errore. Il messaggio di errore verrà recuperato mediante QtLinguist;

Relazioni con altre classi

DDDMob::View::FacadeView

Relazione entrante, riferimento protetto alla finestra di informazione di errore.

4.14 DDDMob::View::CWidgetElement

4.14.1 Informazioni sul package

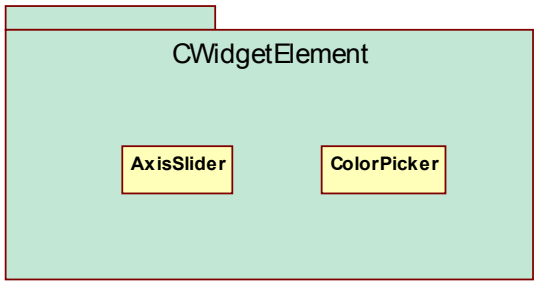

Figura 20: Componente DDDMob::View::CWidgetElement

4.14.1.1 Descrizione

Componente per elementi che sono utilizzati dai widget di View.

4.14.1.2 Interazioni con altri componenti

- DDDMob::View::CDockWidgets;
- Qt$_G$.

4.14.2 Classi

4.14.2.1 DDDMob :: View :: CWidgetElement :: AxisSlider

Descrizione

Classe che rappresenta il widget per modificare dei valori sugli assi x, y, z;

Utilizzo

Viene utilizzata per inviare il signal$_G$ opportuno al controller per la modifica dei valori;

Classi ereditate

- QGroupBox.

Relazioni con altre classi

DDDMob::View::CDockWidgets::Transformation
Relazione entrante, elemento grafico che permette di modificare la rotazione dell'elemento;

DDDMob::View::CDockWidgets::Transformation
Relazione entrante, elemento grafico che permette di modificare la dimensione;

DDDMob::View::CDockWidgets::Transformation
Relazione entrante, elemento grafico che permette di modificare la traslazione dell'elemento.

4.14.2.2 DDDMob :: View :: CWidgetElement :: ColorPicker

Descrizione

Classe che rappresenta il widget che consente di selezionare un colore;

Utilizzo

Viene utilizzata per permette di modificare il colore selezionato. Modifiche al colore selezionato provocano un signal$_G$ appropriato;

Classi ereditate

- QPushButton.

Relazioni con altre classi

DDDMob::View::CDockWidgets::Material
Relazione entrante, elemento grafico che permette di modificare il colore di diffusione$_G$ del materiale;

DDDMob::View::CDockWidgets::Material
Relazione entrante, elemento grafico che permette di modificare il colore speculare$_G$ del materiale;

DDDMob::View::CDockWidgets::Material
Relazione entrante, elemento grafico che permette di modificare il colore di emissione del materiale.

4.15 DDDMob::C3DObject

4.15.1 Informazioni sul package

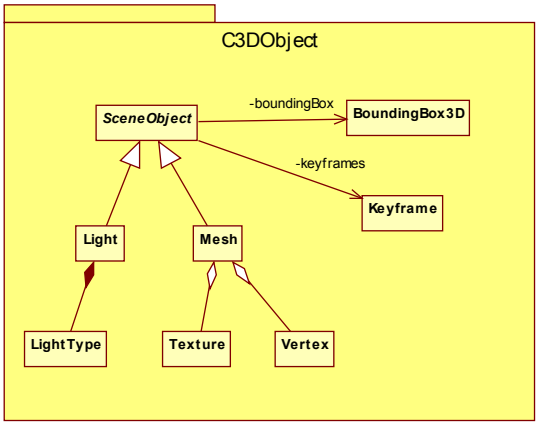

Figura 21: Componente DDDMob::C3DObject

4.15.1.1 Descrizione

Componente condiviso tra Modello e Controller che contiene la definizione dell'oggetto e mette a disposizione le operazioni di modifica.

4.15.1.2 Interazioni con altri componenti

- DDDMob::Model::CConverter::CLoaderModel;
- Qt_G;
- DDDMob::Model::Commands.

4.15.2 Classi

4.15.2.1 DDDMob :: C3DObject :: SceneObject

Descrizione

Classe astratta che rappresenta gli oggetti che è possibile trovare nella scena_G 3D;

Utilizzo

Mette a disposizione i metodi per la modifica degli attributi e per le trasformazioni all'oggetto;

Classi figlie

- DDDMob :: C3DObject :: Light;
- DDDMob :: C3DObject :: Mesh.

Relazioni con altre classi

DDDMob::Model::CConverter::CLoaderModel::Scene

Relazione entrante, l'oggetto 3D attualmente selezionato;

QVector3D

Relazione uscente, posizione;

QVector3D
Relazione uscente, rotazione;

QVector3D
Relazione uscente, ridimensionamento;

QColor
Relazione uscente, colore speculare$_G$;

QColor
Relazione uscente, colore di diffusione$_G$;

QColor
Relazione uscente, colore di emissione;

DDDMob::Model::Commands::CommandAddElement
Relazione entrante, oggetto della scena$_G$;

DDDMob::Model::Commands::CommandEditObject
Relazione entrante, riferimento all'oggetto della scena$_G$;

DDDMob::C3DObject::BoundingBox3D
Relazione uscente, parallelepipedo di volume minimo che contiene completamente la parte visibile dell'oggetto;

DDDMob::C3DObject::Keyframe$_G$
Relazione uscente, sequenza di elementi di animazione;

DDDMob::Model::CConverter::CLoaderModel::SceneAdapter
Relazione entrante, mappa tra i nomi e i riferimenti agli oggetti contenuti nella scena$_G$.

4.15.2.2 DDDMob :: C3DObject :: Light

Descrizione
Classe che rappresenta una luce presente nella scena$_G$ 3D;

Utilizzo
Mette a disposizione i metodi per la modifica degli attributi propri della luce;

Classi ereditate
- DDDMob :: C3DObject :: SceneObject.

Relazioni con altre classi

DDDMob::C3DObject::LightType
Relazione uscente, tipo della luce.

4.15.2.3 DDDMob :: C3DObject :: Mesh

Descrizione
Classe che rappresenta una mesh poligonale presente nella scena$_G$ 3D;

Utilizzo
Mette a disposizione i metodi per la modifica degli attributi propri delle mesh;

Classi ereditate

- DDDMob :: C3DObject :: SceneObject.

Relazioni con altre classi

DDDMob::C3DObject::Texture
Relazione uscente, texture;

DDDMob::C3DObject::Vertex
Relazione uscente, rappresenta una faccia della mesh.

4.15.2.4 DDDMob :: C3DObject :: LightType

Descrizione
Definisce il tipo di luce;

Utilizzo
Viene utilizzato come tipo di enumerazione;

Relazioni con altre classi

DDDMob::Model::Commands::CommandLightType
Relazione entrante, nuovo tipo di luce;

DDDMob::Model::Commands::CommandLightType
Relazione entrante, vecchio tipo di luce;

DDDMob::C3DObject::Light
Relazione entrante, tipo della luce.

4.15.2.5 DDDMob :: C3DObject :: Texture

Descrizione
Classe che rappresenta una texture per un oggetto della scena$_G$;

Utilizzo
Classe utilizzata per la rappresentazione di una texture per un oggetto della scena$_G$, definita con il percorso al file immagine e con una rappresentazione interna dell'immagine grazie alla classe QImage;

Relazioni con altre classi

DDDMob::C3DObject::Mesh
Relazione entrante, texture.

4.15.2.6 DDDMob :: C3DObject :: Vertex

Descrizione

Classe che rappresenta un vertice di un oggetto;

Utilizzo

Viene usata per fornire informazioni utili proprie dei vertici, quali la posizione, le normali e l'UV;

Relazioni con altre classi

DDDMob::C3DObject::Mesh

Relazione entrante, rappresenta una faccia della mesh.

4.15.2.7 DDDMob :: C3DObject :: Keyframe$_G$

Descrizione

Classe che rappresenta un keyframe$_G$ nella scena$_G$ 3D;

Utilizzo

Viene utilizzata per contenere le informazioni di un dato keyframe$_G$, tempo e posizione, per uno degli oggetti della scena$_G$;

Relazioni con altre classi

DDDMob::C3DObject::SceneObject

Relazione entrante, sequenza di elementi di animazione.

4.15.2.8 DDDMob :: C3DObject :: BoundingBox3D

Descrizione

Oggetto Bounding Box$_G$;

Relazioni con altre classi

DDDMob::C3DObject::SceneObject

Relazione entrante, parallelepipedo di volume minimo che contiene completamente la parte visibile dell'oggetto.

5 Diagrammi di attività

Vengono ora illustrati i diagrammi di attività che descrivono le interazioni dell'utente con l'applicativo 3DMob. È stato disegnato un diagramma ad alto livello che descrive le attività possibili, le quali vengono poi illustrate tramite dei sotto-diagrammi specifici. Per agevolare la lettura dei diagrammi si specifica che i riquadri che contengono testo in grassetto sono da considerarsi attività ad alto livello, descritte approfonditamente in sotto-attività tramite i relativi diagrammi. Mentre i riquadri in testo normale sono da considerarsi singole azioni.

5.1 Attività principali

L'utente, dopo aver effettuato l'accesso al sistema, ha la possibilità di *Caricare un file*, *Aprire il sistema di Aiuto contestuale*, *Impostare le Preferenze di sistema* e *Visualizzare le informazioni di sistema*. Queste sono le funzioni principali dell'applicazione e possono essere utilizzate parallelamente senza interrompere lo sviluppo delle singole attività (figura 22).

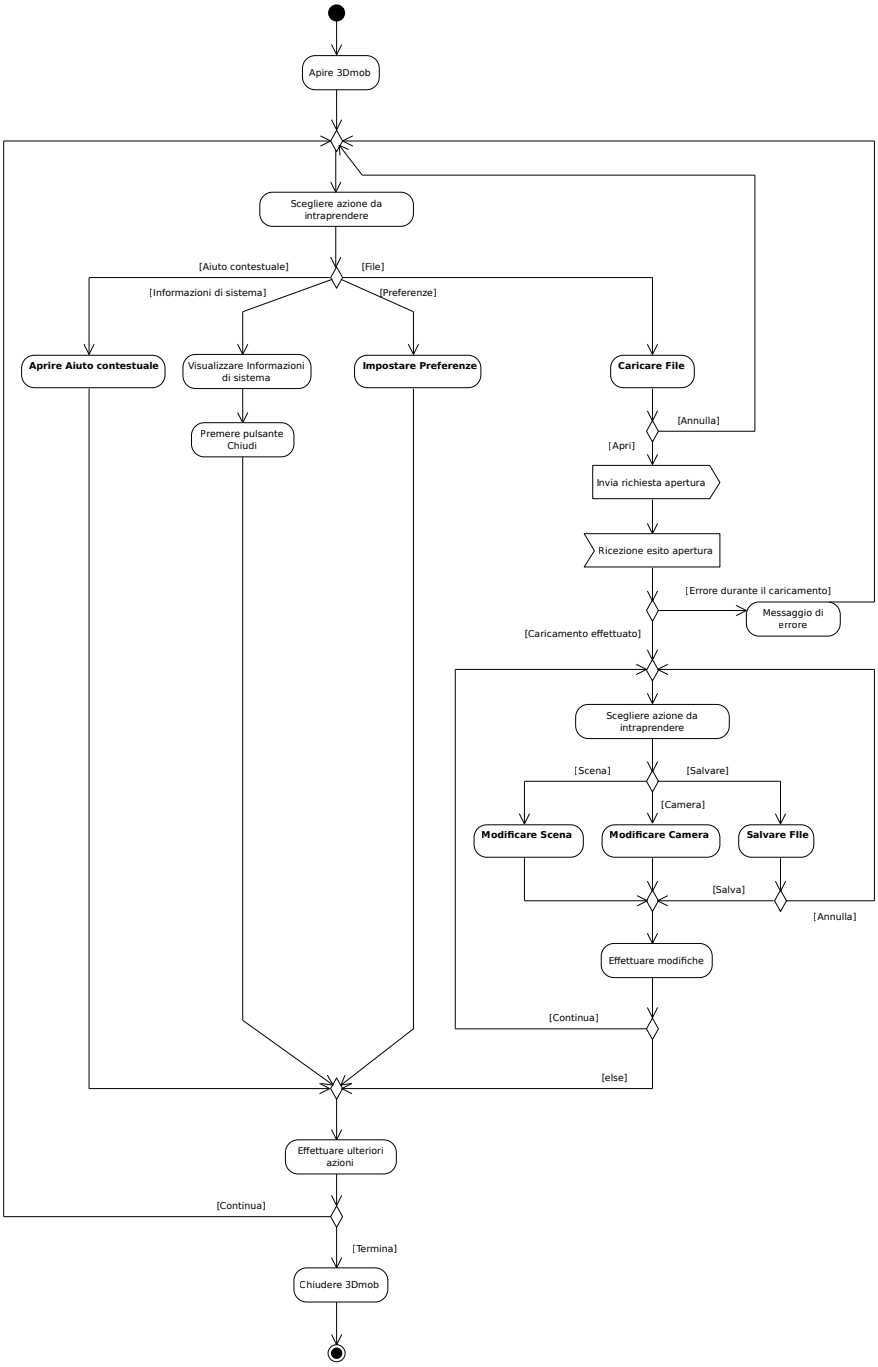

Figura 22: Diagramma attività - Attività principali dell'applicativo 3DMob

Nel momento in cui un file viene caricato correttamente l'utente è in grado di *Modificare la scena*$_G$, *Modificare la camera*$_G$ o *Caricare un altro file*. Una volta applicate le modifiche desiderate è consentito salvare la scena$_G$ ottenuta. È possibile esportare la scena$_G$ anche senza aver apportato modifiche. Dopo aver effettuato correttamente il salvataggio della scena$_G$ è possibile caricare un nuovo file da elaborare oppure uscire dal programma.

5.2 Caricare File

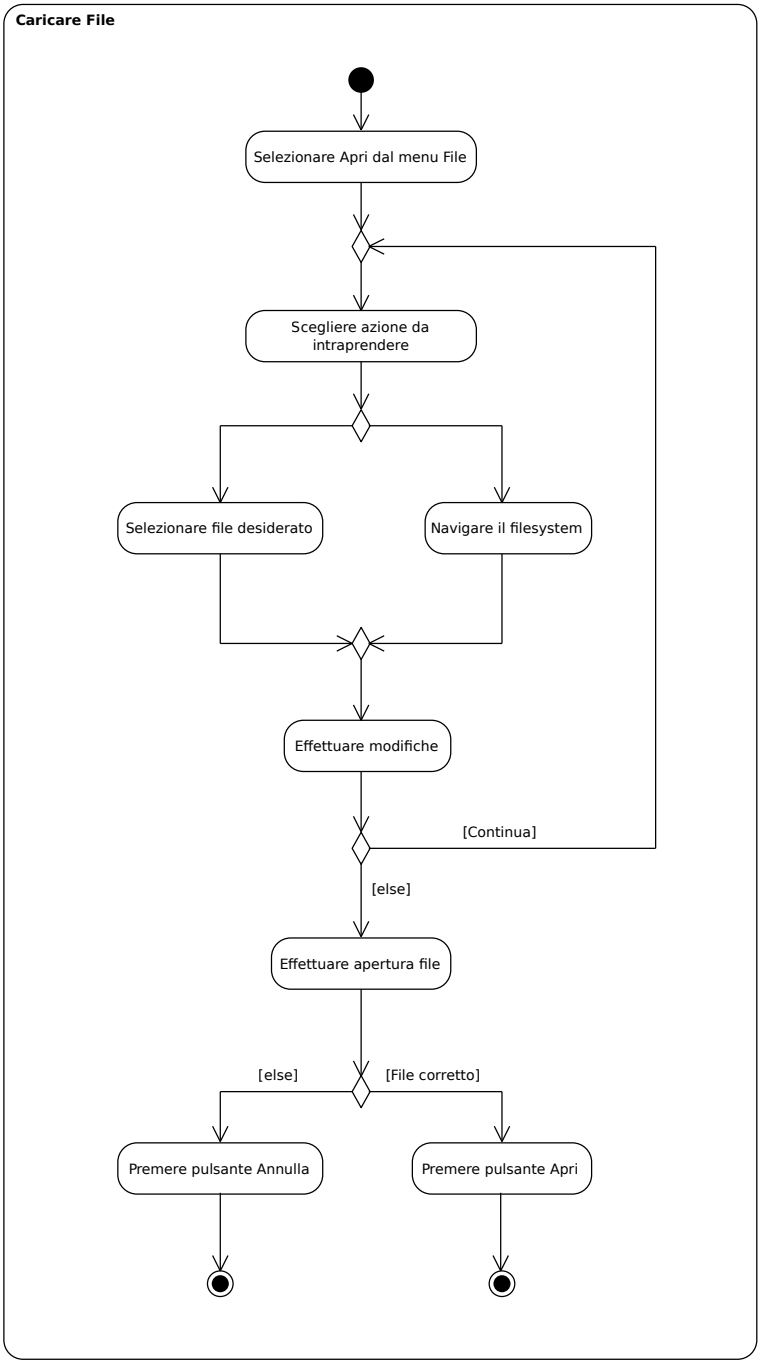

Figura 23: Diagramma attività - Caricamento di un file

L'attività di caricamento in figura 23 di un file comprende le azioni di navigazione del filesystem, di selezione del file scelto e della conferma di apertura. L'utente può opzionalmente annullare il caricamento in qualsiasi momento e di conseguenza ritornare alla finestra principale dell'applicazione.

5.3 Modificare Scena

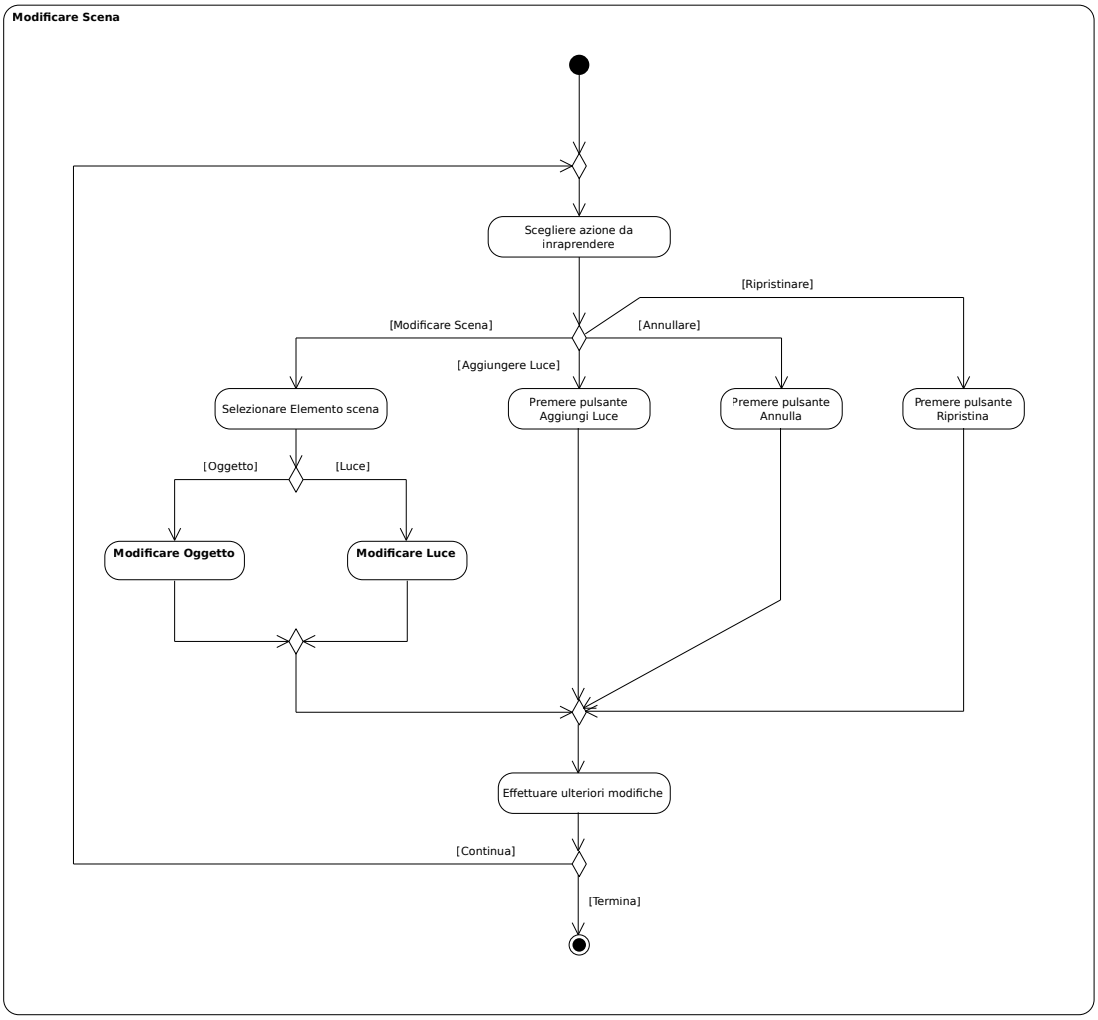

Figura 24: Diagramma attività - Modifica della scena

La modifica della scena$_G$ in figura 24 è l'attività più articolata che l'utente può effettuare tramite l'interfaccia grafica. Ha la possibilità di selezionare un elemento della scena$_G$ ed eseguire su di esso delle modifiche diverse a seconda che sia un oggetto oppure una luce. Inoltre è possibile aggiungere una fonte luminosa oppure agire sui pulsanti di *Annulla* e *Ripristina* per annullare azioni o ripristinare quelle annullate in ordine cronologico.

5.4 Modificare Oggetto

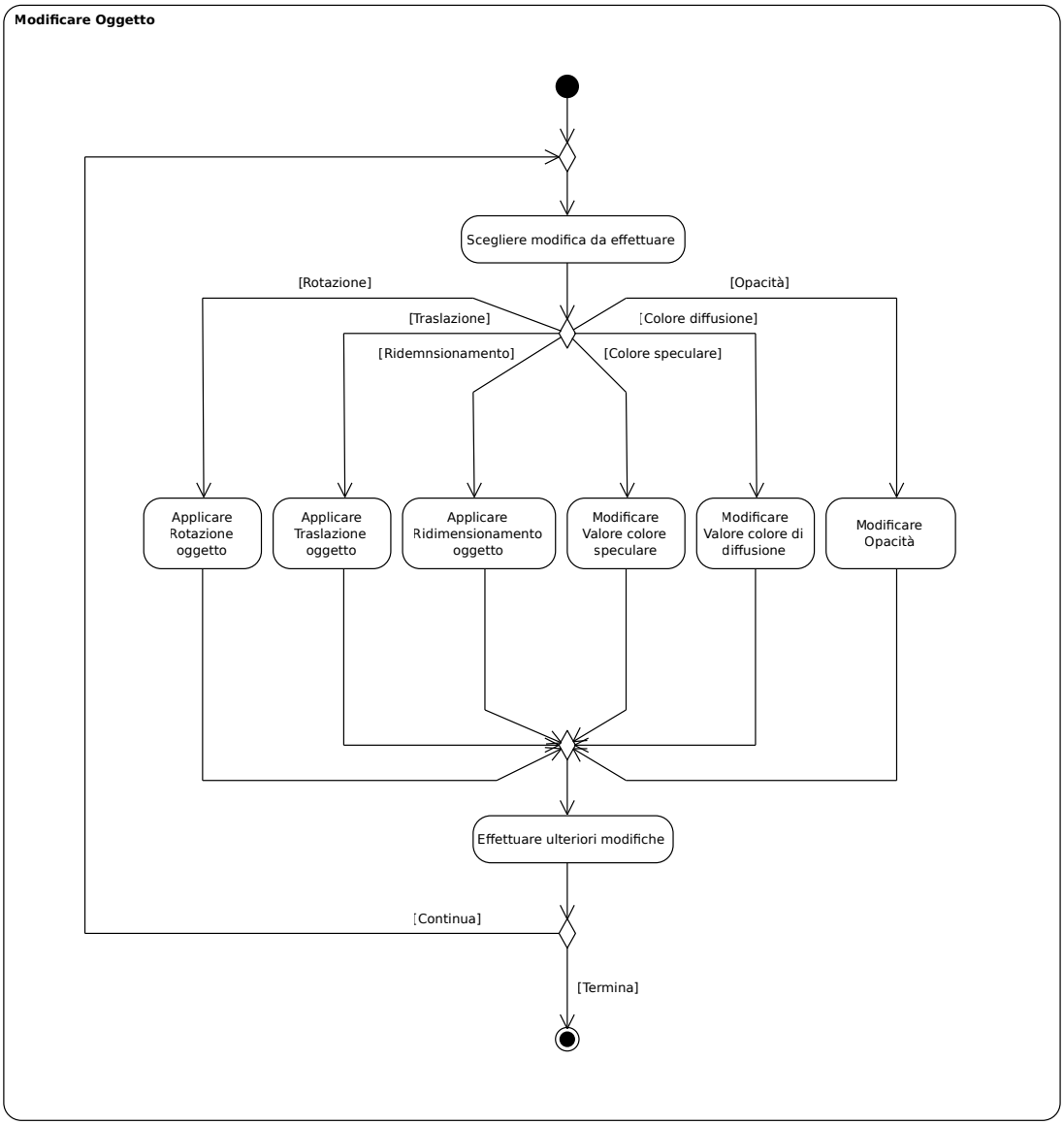

Figura 25: Diagramma attività - Modifica dell'oggetto

Un oggetto presenta diverse proprietà che possono essere modificate e tra queste troviamo la dimensione, il colore speculare$_G$, il colore di diffusione$_G$ e l'opacità; inoltre è possibile applicare ad un oggetto una rotazione oppure una traslazione lungo gli assi di riferimento all'interno della scena$_G$ caricata (figura 25).

5.5 Modificare Luce

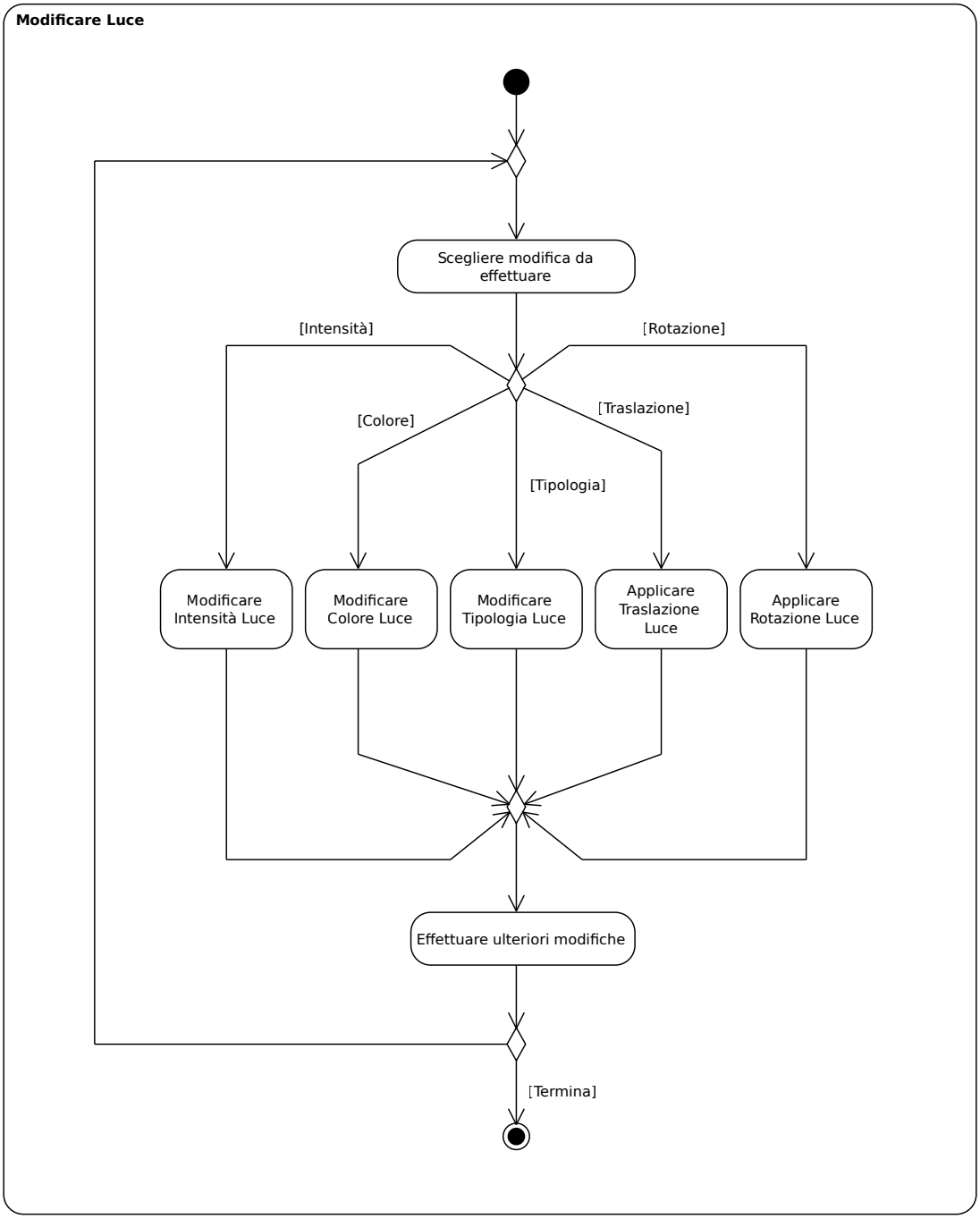

Figura 26: Diagramma attività - Modifica della luce

Una luce presenta diverse proprietà che possono essere modificate e tra queste troviamo l'intensità, il colore e la tipologia; inoltre è possibile applicare ad una luce una rotazione oppure una traslazione lungo gli assi di riferimento all'interno della scena$_G$ caricata (figura 26).

5.6 Modificare Camera

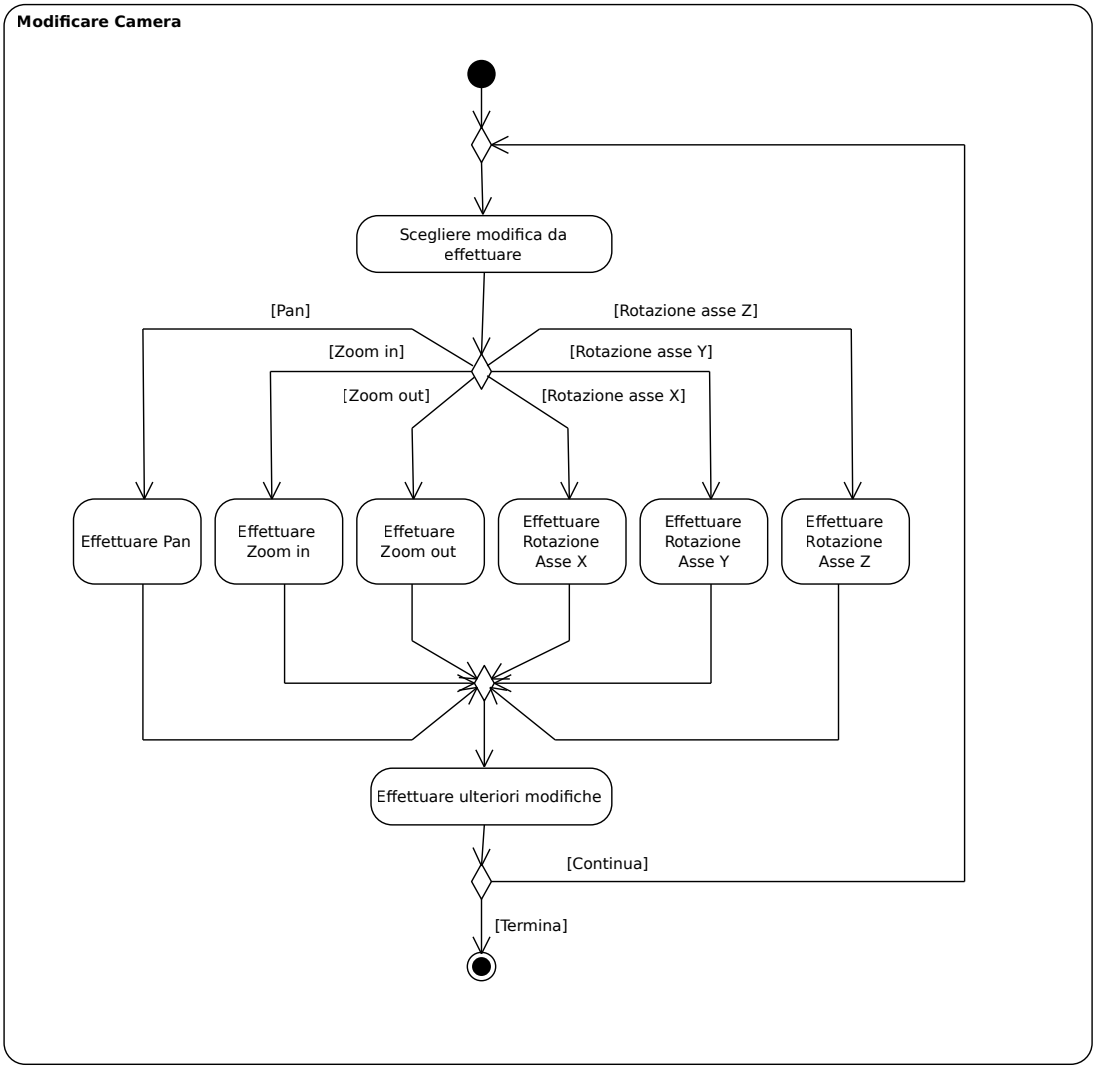

Figura 27: Diagramma attività - Modifica della camera

La modifica della camera$_G$ in figura 27 è un'attività eseguibile direttamente sull'anteprima della scena$_G$ e comporta dei cambiamenti nella visualizzazione del modello che non implicano modifiche reali alla scena$_G$ caricata ma permettono all'utente di focalizzare la propria attenzione sui particolari di maggior interesse.

5.7 Salvare File

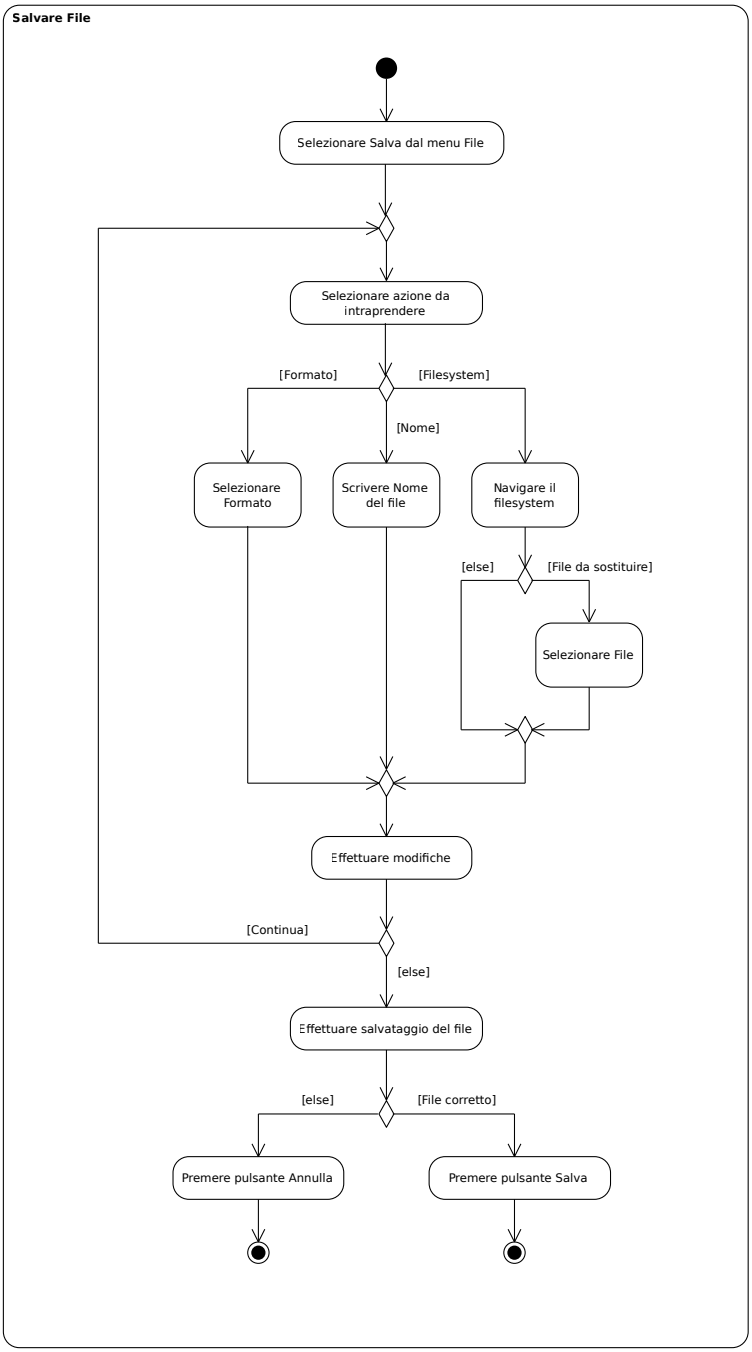

Figura 28: Diagramma attività - Salvataggio della scena

Il salvataggio di una scena$_G$ descritto in figura 28 viene svolto attraverso una finestra che permette di navigare il filesystem per selezionare l'esatta posizione in cui salvare oppure selezionare un file da sovrascrivere. È inoltre possibile scrivere il percorso del file e scegliere il formato del file in output.
La pressione del pulsante *Conferma* avvierà il processo di salvataggio mentre la pressione del *Annulla* riporterà l'utente alla finestra principale.

5.8 Impostare Preferenze

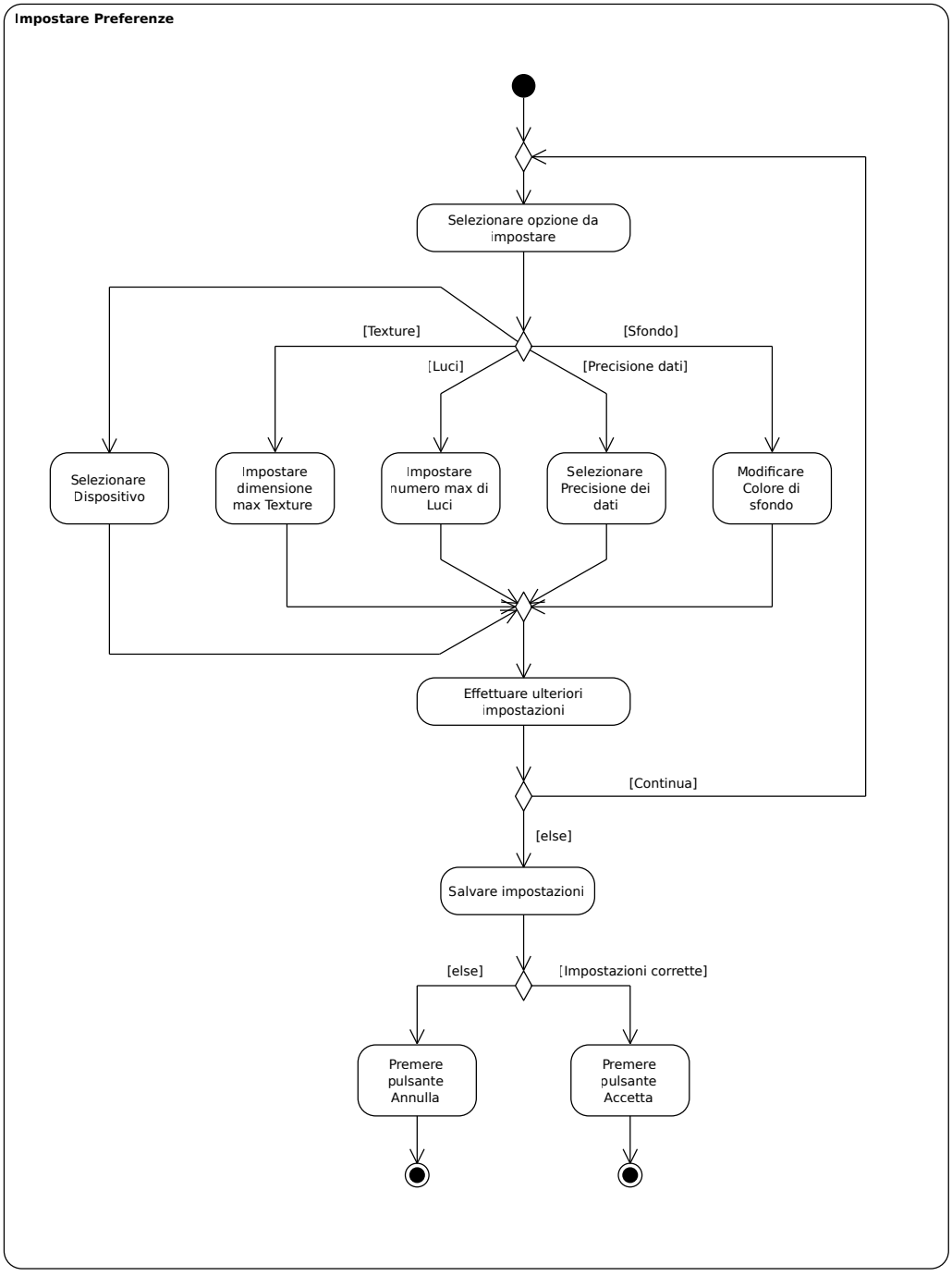

Figura 29: Diagramma attività - Impostazione delle preferenze del sistema

L'utente, come illustrato in figura 29, può visualizzare la finestra di impostazioni generali dell'applicazione nella quale può impostare la precisione dei dati della scena$_G$ e i limiti intrinseci dei dispositivi sui quali si vuole esportare il modello tridimensionale di partenza.

5.9 Aprire Aiuto contestuale

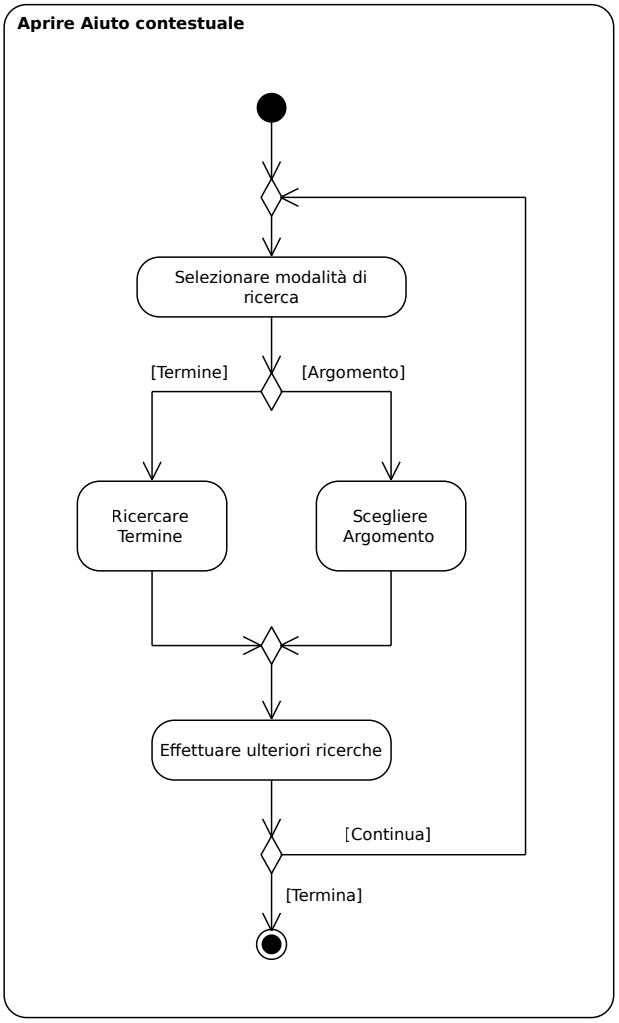

Figura 30: Diagramma attività - Apertura della finestra di aiuto contestuale

L'aiuto contestuale descritto in figura 30 consente di ricercare un termine e visualizzarne le relative informazioni, oppure di scegliere un particolare argomento del quale si vuole approfondire la conoscenza.

6 Design Pattern

I Design Pattern$_G$ sono soluzioni a problemi ricorrenti. Adottare i Design Pattern$_G$ semplifica l'attività di progettazione, favorisce il riutilizzo del codice e rende l'architettura più manutenibile.
I Design Pattern$_G$ possono essere suddivisi in:

- **Design Pattern$_G$ architetturali:** definiscono l'architettura dell'applicazione ad un livello più elevato;
- **Design Pattern$_G$ creazionali:** consentono di nascondere i costruttori delle classi, permettendo di creare oggetti senza conoscere la loro implementazione;
- **Design Pattern$_G$ strutturali:** consentono di riutilizzare classi pre-esistenti, fornendo un'interfaccia più adatta;
- **Design Pattern$_G$ comportamentali:** definiscono soluzioni per le interazioni tra oggetti.

Per una descrizione generale ed approfondita dei Design Pattern$_G$ utilizzati si veda l'Appendice A.
Nella realizzazione del progetto 3DMob si è deciso di implementare i seguenti Design Pattern$_G$.

6.1 Design Pattern architetturali

6.1.1 MVC

- **Scopo dell'utilizzo:** È stato scelto il pattern MVC per separare la logica dell'applicazione dalla rappresentazione grafica;
- **Contesto d'utilizzo:** Il pattern MVC viene utilizzato per l'architettura generale dell'applicazione. Viene utilizzato il sistema Signal$_G$ e Slot di Qt$_G$, descritto nella sezione 2.2.1, per far comunicare i componenti tra loro.

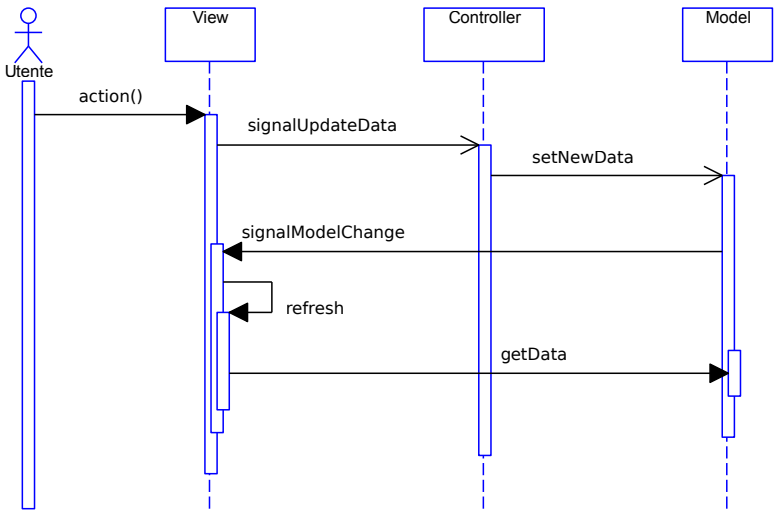

Figura 31: Diagramma di sequenza - Modifica della scena

Ogni modifica effettuata dall'utente sulla View viene inviata al Controller che invia un comando al Model. Il Model notifica la View di ogni cambiamento e la View va a recuperare i dati aggiornati dal Model.
Nel diagramma di sequenza seguente, quando nei segnali vengono passati dei dati, i dati passati vengono indicati dentro alle parantesi del metodo. I segnali in Qt_G sono sincroni. Per chiarire come Qt_G gestisce internamente i segnali, vengono indicati i ritorni dei segnali stessi. Tali ritorni sono nascosti al programmatore e quindi non verranno indicati in altri punti della *Specifica Tecnica v4.2.0* e nella *Definizione di Prodotto v4.2.0*.

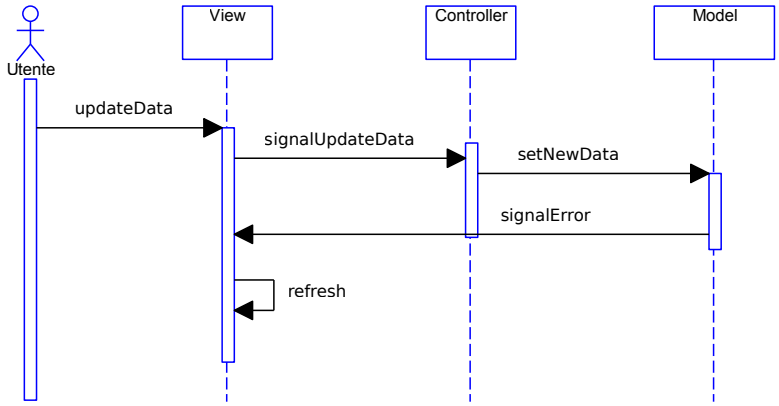

Figura 32: Diagramma di sequenza - Modifica della scena con errore

Nel caso si verifichi un errore il Model notifica la View dell'errore.

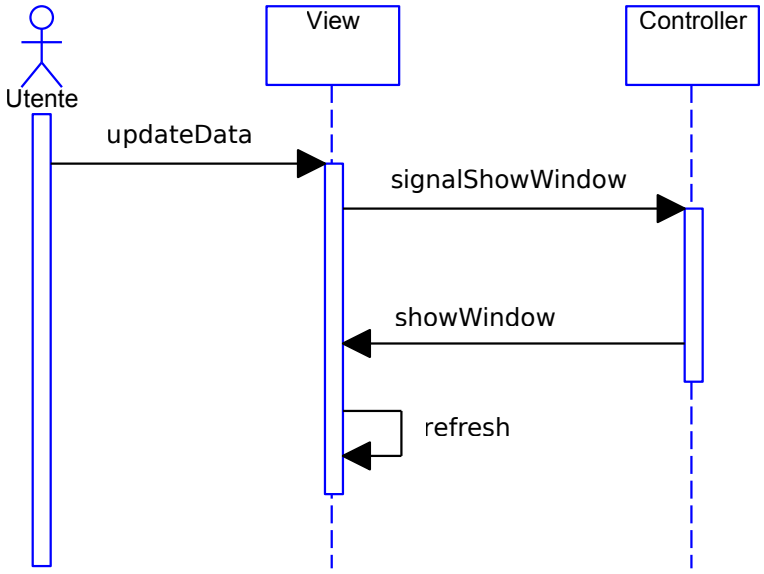

Figura 33: Diagramma di sequenza - Selezione ed apertura della vista

Nell'apertura delle finestre Settings e About, viene inviato un $Signal_G$ dalla View al Controller. Successivamente il Controller seleziona la nuova finestra e lo comunica alla View.

6.2 Design Pattern creazionali

6.2.1 Singleton

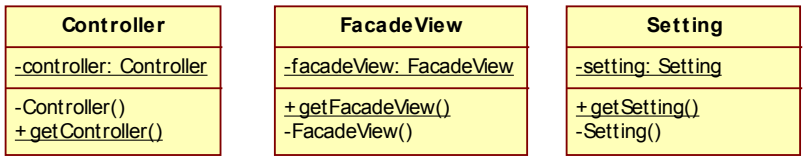

Figura 34: Diagramma del Design Pattern Singleton in 3DMob

- **Scopo dell'utilizzo:** Viene usato il pattern Singleton per le classi che devono avere un'unica istanza durante l'esecuzione dell'applicazione;
- **Contesto d'utilizzo:** Le classi che devono avere un'unica istanza sono:
  - `Settings`;
  - `FacadeView`;
  - `Controller`.

6.3 Design Pattern strutturali

6.3.1 Adapter

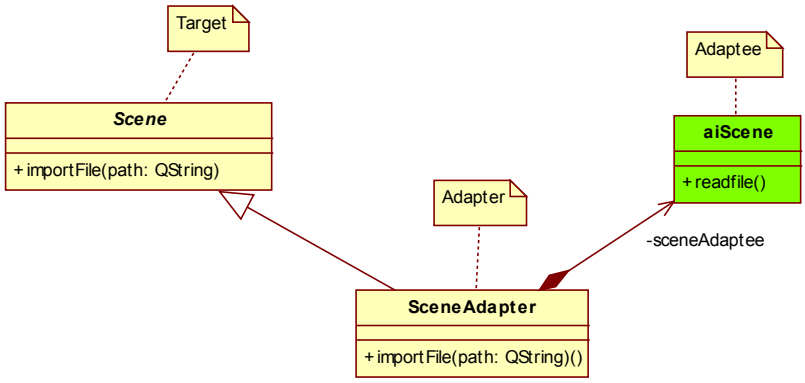

Figura 35: Applicazione di Adapter in 3DMob

- **Scopo dell'utilizzo:** Il pattern Adapter viene utilizzato per adattare una classe riutilizzando un oggetto già esistente. Questo semplifica l'eventuale processo di sostituzione dell'oggetto esistente, creando un'interfaccia stabile per il resto dell'applicazione;
- **Contesto d'utilizzo:** È stato usato per adattare la classe che rappresenta la scena$_G$ in Assimp. `SceneAdapter` adatta `aiScene`.

6.3.2 Facade

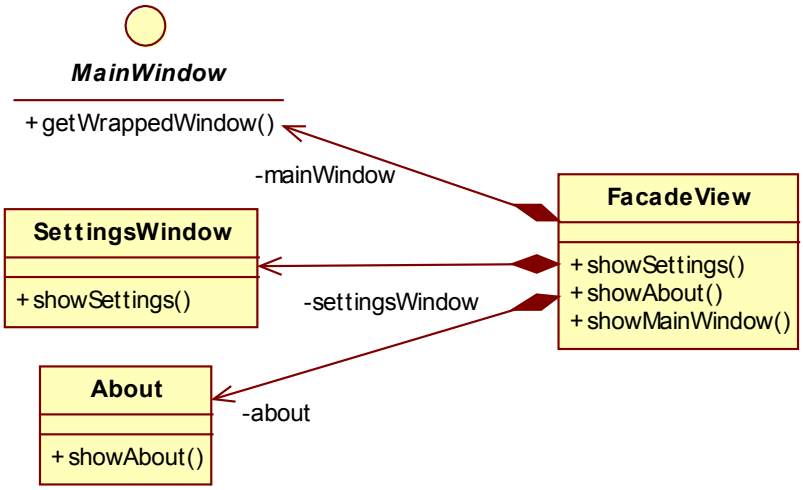

Figura 36: Applicazione di Facade in 3DMob

- **Scopo dell'utilizzo:** Il pattern Facade viene usato per fornire un'interfaccia unica a più classi;
- **Contesto d'utilizzo:** `FacadeView` è una Facade che presenta un'interfaccia per tutti gli oggetti che compongono la View:
  - `MainWindow`;
  - `SettingsWindow`;
  - `About`.

6.3.3 Decorator

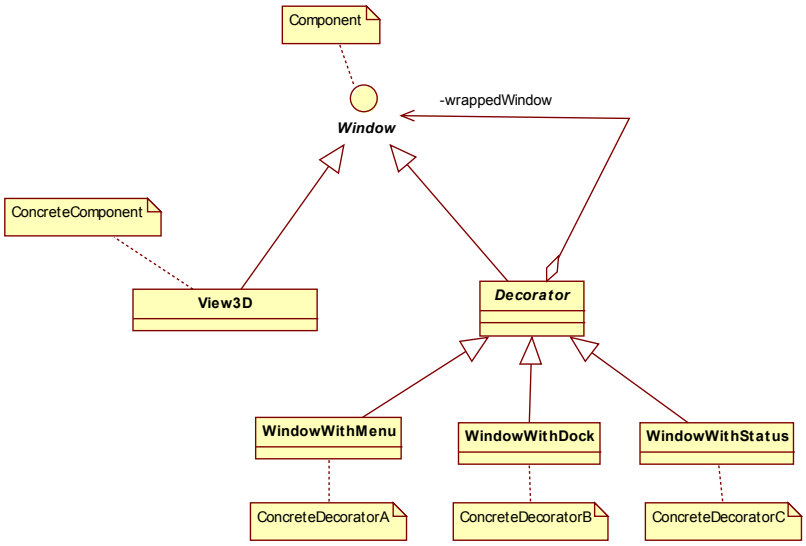

Figura 37: Applicazione di Decorator in 3DMob

- **Scopo dell'utilizzo:** Il pattern Decorator viene usato per aggiungere delle caratteristiche minori ad un oggetto di base, rendendolo facilmente estensibile;
- **Contesto d'utilizzo:** Viene usato il pattern Decorator per estendere `View3D`, la classe che rappresenta la vista 3D con interfaccia minimale. `View3D` viene esteso con le seguenti classi ConcreteDecorator:
  - `WindowWithMenu`;
  - `WindowWithDock`;
  - `WindowWithStatus`.

6.4 Design Pattern comportamentali

6.4.1 Command

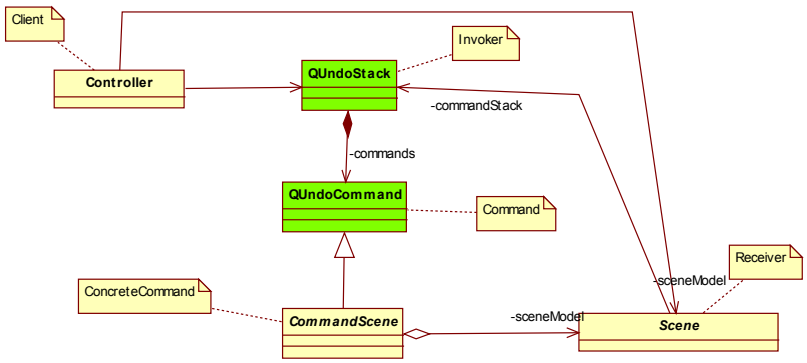

Figura 38: Applicazione di Command in 3DMob

- **Scopo dell'utilizzo:** Il pattern Command viene utilizzato per separare il codice di un'azione dal codice che richiede l'esecuzione dello stesso;
- **Contesto d'utilizzo:** Viene usato per la gestione dei comandi che modificano la scena$_G$. Le classi che implementano i comandi sono:
  - `CommandAddLight`;
  - `CommandPosition`;
  - `CommandScale`;
  - `CommandRotation`;
  - `CommandLightType`;
  - `CommandShininess`;
  - `CommandEmission`;
  - `CommandColor`;
  - `CommandColorEmission`;
  - `CommandColorSpecular`;
  - `CommandColorDiffuse`.

  La classe incaricata di eseguire i comandi è `QUndoStack`. Il `Controller` è il Client del Command e riceve un Signal$_G$ dalla View con cui interagisce l'utente.

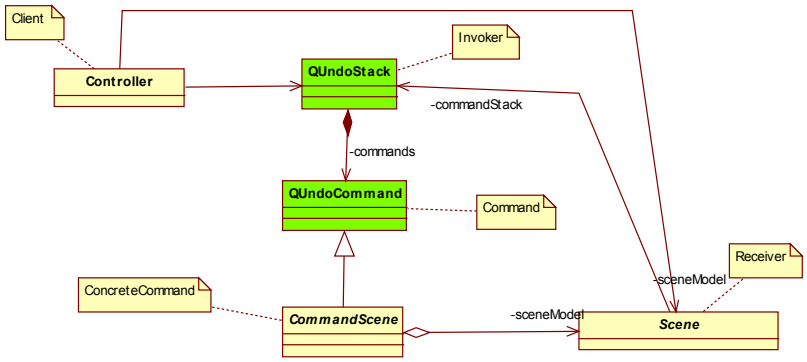

Figura 39: Diagramma di sequenza - Gestore di comandi della scena

Dopo che la View ha inviato un Signal_G al Controller, viene creato un `CommandScene` che viene inserito nella `QUndoStack` presente nel Model. `QUndoStack` invoca il comando e modifica la `Scene`.

6.4.2 Strategy

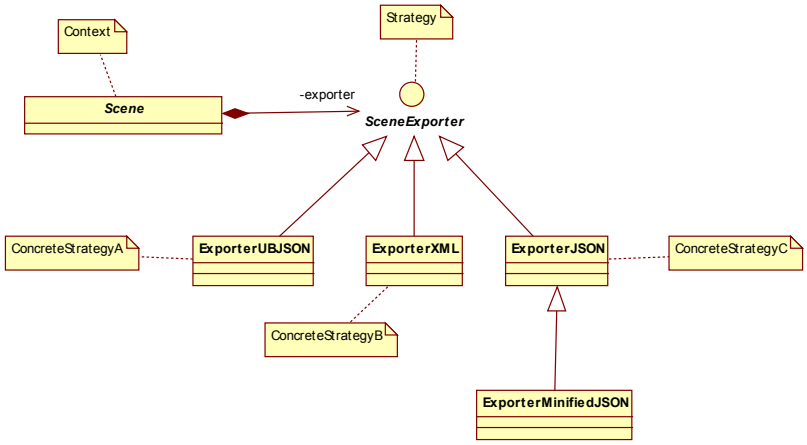

Figura 40: Applicazione di Strategy in 3DMob

- **Scopo dell'utilizzo:** Il pattern Strategy viene usato per isolare più algoritmi che svolgono la stessa funzione dal codice che esegue la funzione;
- **Contesto d'utilizzo:** Viene usato per gestire gli algoritmi di esportazione in `SceneExporter`. Dopo che l'utente seleziona il tipo di file da esportare, viene selezionato l'algoritmo di conversione corrispondente tra una delle ConcreteStrategy:
  - `ExporterUBJSON`;
  - `ExporterXML`;
  - `ExporterJSON`;
  - `ExporterMinifiedJSON`.

7 Stime di fattibilità e di bisogno di risorse

L'architettura definita precedentemente ha raggiunto un livello di dettaglio sufficiente per fornire una stima sulla fattibilità e di bisogno di risorse.
L'analisi dell'architettura progettata ha permesso di constatare che le tecnologie che si è scelto di adottare risultano sufficientemente adeguate per la realizzazione del prodotto e riescono a ricoprire le esigenze progettuali.
Gli strumenti scelti sono conosciuti dalla maggioranza dei componenti del gruppo che si impegneranno comunque ad approfondire le loro conoscenze inerenti alla grafica 3D$_G$ e le librerie utilizzate.
Gli strumenti utilizzati sono:

- Qt$_G$ framework$_G$ per la stesura del codice;
- Assimp, libreria per importare file 3D;
- Boost PropertyTree, per effettuare il parsing dei file;
- Qmake per la compilazione automatica del codice e la produzione del Makefile.

Tali strumenti potranno garantire una realizzazione efficace di tutti gli aspetti architetturali.

8 Tracciamento

Seguono le tabelle di tracciamento tra componenti e requisiti. Per semplicità di lettura, requisiti associati a figli di un componente non sono riportati anche nel padre.
I componenti riguardanti il Controller risultano senza requisiti tracciati per la natura di semplice gestore del flusso di informazioni tra View e Model.
I componenti DDDMob::View::CDockWidgets e DDDMob::View::CWidgetElement hanno il solo scopo di contenere classi generiche per l'input utente da interfaccia grafica. Ciò significa che questi non abbiano requisiti associati ma vengono utilizzati da altri componenti della View per soddisfare i propri requisiti.

8.1 Tracciamento componenti - requisiti

Componenti	Requisiti
DDDMob	
⟶ DDDMob::Model	
⟶ DDDMob::Model::Commands	
⟶ DDDMob::Model::CConverter	
↪ DDDMob::Model::CConverter::CSettingsModel	R0F1.1 R0F1.2 R0V2 R1F7.1.1.1 R0V2.1 R0V2.2 R2F1.6
↪ DDDMob::Model::CConverter::CExportModel	R0F1 R0F1.3.1 R0F1.3.2 R0F1.3.3 R0F1.3.4 R0Q4 R1Q5 R1F1.3.5 R0F1.3
↪ DDDMob::Model::CConverter::CLoaderModel	R0F8.1 R2F8.2 R1F8.4.1 R0F8.3.3 R0V2.1 R0V2.2
⟶ DDDMob::Controller	
⟶ DDDMob::View	R0F8 R0F1.4 R2F1.5 R2F1.7
⟶ DDDMob::View::CDockWidgets	

↪ DDDMob::View::CMainWindow	R2F7.1 R2F7.1.2.1 R2F7.1.2.2 R2F7.1.2.3 R1F8.3 R1F8.3.1.1.1 R1F8.3.1.1.2 R1F8.3.1.1.3 R1F8.3.1.2.1 R1F8.3.1.1.4 R1F8.4 R1F8.3.1.1.5 R1F8.3.1.1.5.1 R2F8.3.1 R2F7.1.1 R2F8.3.2 R0F8.3.3 R1F8.3.4 R2F7.1.2 R1F8.3.1.2.2 R1F8.3.5 R1F8.3.1.2.3 R2F8.3.1.1 R2F8.3.1.2 R0F8.3.1.2.4 R2F8.3.1.2.5 R1F8.3.1.1.4.1 R1F8.3.1.1.4.2 R2F8.3.4.1
↪ DDDMob::View::CSettingsWindow	R0F1.1 R0F1.2 R1F7.1.1.1 R2F1.6 R1F1.6.3
↪ DDDMob::View::CErrorWindow	R2F7.2 R2F7.2.1 R2F7.2.2 R2F7.2.3
↪ DDDMob::View::CWidgetElement	
↪ DDDMob::C3DObject	R1F8.3 R1F8.3.1.2.1 R2F8.3.1 R1F8.3.1.2.2 R1F8.3.1.2.3 R2F8.3.1.1 R2F8.3.1.2 R0F8.3.1.2.4 R2F8.3.1.2.5

Tabella 2: Tabella componenti / requisiti

8.2 Tracciamento requisiti - componenti

Requisito	Descrizione	Componenti
R0F1	Il programma deve essere in grado di salvare la scena$_G$ in un file secondo uno dei formati permessi	DDDMob::Model::-CConverter::-CExportModel
⟶ R0F1.1	L'utente deve poter decidere se salvare la scena$_G$ in formato leggibile o in formato minificato$_G$	DDDMob::Model::-CConverter::-CSettingsModel DDDMob::View::-CSettingsWindow
⟶ R0F1.2	L'utente deve poter decidere se salvare la scena$_G$ utilizzando dati in virgola mobile con precisione singola o precisione doppia	DDDMob::Model::-CConverter::-CSettingsModel DDDMob::View::-CSettingsWindow
⟶ R0F1.3	Il programma deve mantenere nell'esportazione le caratteristiche del solido	DDDMob::Model::-CConverter::-CExportModel
⟶ R0F1.3.1	Il programma dovrà esportare file che mantengano le normali e i vertici del solido presenti nell'oggetto importato	DDDMob::Model::-CConverter::-CExportModel
⟶ R0F1.3.2	Il programma dovrà esportare file che mantengano le caratteristiche delle texture presenti nell'oggetto importato, ovvero l'istogramma della texture importata dovrà essere uguale a quello della texture esportata	DDDMob::Model::-CConverter::-CExportModel
⟶ R0F1.3.3	Il programma dovrà esportare file che rispettino i parametri dei materiali (colore e emissività) presenti nel modello importato, ovvero siano gli stessi a meno di modifiche utente	DDDMob::Model::-CConverter::-CExportModel
⟶ R0F1.3.4	Il programma dovrà esportare file che rispettino le sorgenti di luce presenti nel modello importato, ovvero mantengano le fonti di luce non in soprannumero rispetto al massimo configurato	DDDMob::Model::-CConverter::-CExportModel
⟶ R1F1.3.5	Il programma deve mantenere le animazioni presenti nel file importato durante la fase di esportazione	DDDMob::Model::-CConverter::-CExportModel

⟶ R0F1.4	L'utente deve poter scegliere il formato JSON$_G$ per l'esportazione	DDDMob::View
⟶ R2F1.5	L'utente deve poter scegliere il formato XML$_G$ per l'esportazione	DDDMob::View
⟶ R2F1.6	L'utente deve poter configurare i limiti di importazione della scena$_G$	DDDMob::Model::-CConverter::-CSettingsModel DDDMob::View::-CSettingsWindow
⟶ R1F1.6.3	L'utente deve poter selezionare i limiti di importazione da una lista di valori predefiniti	DDDMob::View::-CSettingsWindow
⟶ R2F1.7	L'utente deve poter scegliere il formato binario$_G$ Universal Binary JSON$_G$ Specification (http://ubjson.org/) per l'esportazione	DDDMob::View
R0V2	Il programma, di default, deve esportare file che rispettino i limiti dell'iPhone 4S	DDDMob::Model::-CConverter::-CSettingsModel
⟶ R0V2.1	Non devono essere presenti più di 8 luci nella scena$_G$ esportata	DDDMob::Model::-CConverter::-CLoaderModel DDDMob::Model::-CConverter::-CSettingsModel
⟶ R0V2.2	Non devono essere presenti texture di larghezza e altezza superiore a 4096 pixel nella scena$_G$ esportata	DDDMob::Model::-CConverter::-CLoaderModel DDDMob::Model::-CConverter::-CSettingsModel
R0Q4	Il programma deve esportare i dati in un formato che segua il workflow$_G$ necessario al rendering$_G$ della scena$_G$ in OpenGL ES$_G$ 2.0	DDDMob::Model::-CConverter::-CExportModel
R1Q5	Il programma deve esportare i dati in un formato che segua il workflow$_G$ necessario al rendering$_G$ della scena$_G$ in OpenGL ES$_G$ 3.0	DDDMob::Model::-CConverter::-CExportModel
⟶ R2F7.1	Il programma deve mostrare un'anteprima della scena$_G$ che verrà esportata	DDDMob::View::-CMainWindow

↪ R2F7.1.1	L'anteprima della scena$_G$ da esportare deve prevedere uno sfondo	DDDMob::View::-CMainWindow
↪ R1F7.1.1.1	L'utente deve poter modificare il colore dello sfondo dell'anteprima	DDDMob::Model::-CConverter::-CSettingsModel DDDMob::View::-CSettingsWindow
↪ R2F7.1.2	L'anteprima della scena$_G$ da esportare deve essere modificabile	DDDMob::View::-CMainWindow
↪ R2F7.1.2.1	L'utente deve poter ruotare liberamente l'anteprima	DDDMob::View::-CMainWindow
↪ R2F7.1.2.2	L'utente deve poter modificare lo zoom dell'anteprima	DDDMob::View::-CMainWindow
↪ R2F7.1.2.3	L'utente deve poter spostare la camera$_G$ e navigare nell'anteprima	DDDMob::View::-CMainWindow
↪ R2F7.2	Deve essere previsto un sistema di notifica degli errori e delle informazioni	DDDMob::View::-CErrorWindow
↪ R2F7.2.1	Il sistema di notifica deve poter segnalare gli errori nell'apertura del file	DDDMob::View::-CErrorWindow
↪ R2F7.2.2	Il sistema di notifica deve poter segnalare i malfunzionamenti del sistema	DDDMob::View::-CErrorWindow
↪ R2F7.2.3	Il sistema di notifica deve poter segnalare le perdite di informazione nella conversione	DDDMob::View::-CErrorWindow
R0F8	L'utente deve poter scegliere il file da aprire o importare	DDDMob::View
↪ R0F8.1	Deve essere possibile importare un file 3ds$_G$	DDDMob::Model::-CConverter::-CLoaderModel
↪ R2F8.2	Il programma deve essere in grado di leggere i file in JSON$_G$ che crea	DDDMob::Model::-CConverter::-CLoaderModel
↪ R1F8.3	L'utente deve poter modificare la scena$_G$ importata	DDDMob::C3DObject DDDMob::View::-CMainWindow
↪ R2F8.3.1	L'utente deve poter modificare un oggetto od una fonte di luce	DDDMob::C3DObject DDDMob::View::-CMainWindow
↪ R2F8.3.1.1	L'utente deve poter modificare un oggetto	DDDMob::C3DObject DDDMob::View::-CMainWindow

↳ R1F8.3.1.1.1	L'utente deve poter ruotare l'oggetto selezionato negli assi X, Y e Z	DDDMob::View::-CMainWindow
↳ R1F8.3.1.1.2	L'utente deve poter traslare l'oggetto selezionato negli assi X, Y e Z	DDDMob::View::-CMainWindow
↳ R1F8.3.1.1.3	L'utente deve poter scalare le dimensioni dell'oggetto selezionato	DDDMob::View::-CMainWindow
↳ R1F8.3.1.1.4	Deve essere possibile modificare le caratteristiche dei materiali dell'oggetto selezionato	DDDMob::View::-CMainWindow
⅄ R1F8.3.1.1.4.1	Deve essere possibile modificare il colore speculare$_G$ dei materiali di un oggetto nella scena$_G$	DDDMob::View::-CMainWindow
⅄ R1F8.3.1.1.4.2	Deve essere possibile modificare il colore di diffusione$_G$ per i materiali dell'oggetto selezionato.	DDDMob::View::-CMainWindow
↳ R1F8.3.1.1.5	Deve essere possibile modificare l'opacità dell'oggetto selezionato	DDDMob::View::-CMainWindow
⅄ R1F8.3.1.1.5.1	Deve essere possibile modificare l'opacità di parte dell'oggetto selezionato	DDDMob::View::-CMainWindow
⟷ R2F8.3.1.2	L'utente deve poter modificare una fonte di luce	DDDMob::C3DObject DDDMob::View::-CMainWindow
↳ R1F8.3.1.2.1	L'utente deve poter modificare l'intensità della luce selezionata	DDDMob::C3DObject DDDMob::View::-CMainWindow
↳ R1F8.3.1.2.2	L'utente deve poter modificare il colore delle luci	DDDMob::C3DObject DDDMob::View::-CMainWindow
↳ R1F8.3.1.2.3	L'utente deve poter modificare la tipologia della fonte di luce	DDDMob::C3DObject DDDMob::View::-CMainWindow
↳ R0F8.3.1.2.4	L'utente deve poter ruotare la luce selezionata negli assi X, Y e Z	DDDMob::C3DObject DDDMob::View::-CMainWindow
↳ R2F8.3.1.2.5	L'utente deve poter traslare la luce selezionata negli assi X, Y e Z	DDDMob::C3DObject DDDMob::View::-CMainWindow
⟷ R2F8.3.2	L'utente deve poter annullare o ripristinare le azioni di modifica effettuate su oggetti o luci della scena$_G$	DDDMob::View::-CMainWindow

↪ R0F8.3.3	Tutte le modifiche effettuate devono riflettersi sull'anteprima	DDDMob::Model::-CConverter::-CLoaderModel DDDMob::View::-CMainWindow
↪ R1F8.3.4	L'utente deve poter aggiungere una fonte di luce alla scena_G con valori di materiale predefiniti e alla posizione (0,0,0)	DDDMob::View::-CMainWindow
↪ R2F8.3.4.1	L'utente deve poter scegliere il tipo di fonte luminosa da aggiungere tra omni e spotlight_G	DDDMob::View::-CMainWindow
↪ R1F8.3.5	L'utente deve poter selezionare l'oggetto o la fonte di luce che desidera modificare	DDDMob::View::-CMainWindow
↪ R1F8.4	Deve essere possibile importare un file nel formato Wavefront obj	DDDMob::View::-CMainWindow
↪ R1F8.4.1	IL sistema deve poter importare un file nel formato mtl riferito all'interno del file obj importato	DDDMob::Model::-CConverter::-CLoaderModel

Tabella 3: Tabella requisiti / componenti

A Descrizione Design Pattern

A.1 Design Pattern architetturali

A.1.1 MVC

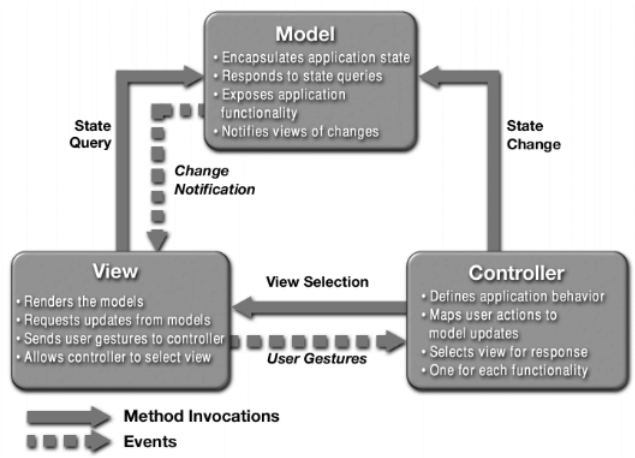

Figura 41: Diagramma del Design Pattern MVC

- **Scopo:** Disaccoppiare le tre componenti seguenti:
  - Model: dati di business e regole di accesso;
  - View: rappresentazione grafica;
  - Controller: reazioni della UI agli input utente.
- **Motivazione:** Lo scopo di molte applicazioni è quello di recuperare dati e visualizzarli in maniera opportuna a seconda delle esigenze degli utenti. Poiché il flusso chiave di informazione avviene tra il dispositivo su cui sono memorizzati i dati e l'interfaccia utente, si è portati a legare insieme queste due parti per ridurre la quantità di codice e migliorare le performance dell'applicazione. Questo approccio, apparentemente naturale, presenta alcuni problemi significativi; uno di questi è che l'interfaccia utente tende a cambiare più in fretta rispetto al sistema di memorizzazione dei dati. Un altro problema, che si ha nel mettere insieme i dati e l'interfaccia utente, è che le applicazioni aziendali tendono ad incorporare logica di business che va al di là della semplice trasmissione di dati. C'è la necessità, quindi, di rendere modulari le funzionalità dell'interfaccia utente in maniera tale da poter facilmente modificare le singole parti. La soluzione a tutto ciò è costituita dal pattern Model-View-Controller (MVC) che separa la modellazione del dominio, la presentazione, e le azioni basate sugli input degli utenti all'interno di tre classi separate;
- **Applicabilità:** Il pattern MVC può essere utilizzato nei seguenti casi:
  - Quando si vuole trattare un gruppo di oggetti come un oggetto singolo;
  - Quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro;
  - Quando si vogliono agganciare più View a un Model per fornire più rappresentazioni del Model stesso.

A.2 Design Pattern creazionali

A.2.1 Singleton

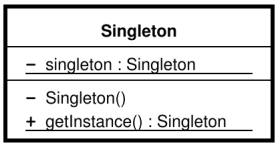

Figura 42: Diagramma del Design Pattern Singleton

- **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 far ciò 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 pattern 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

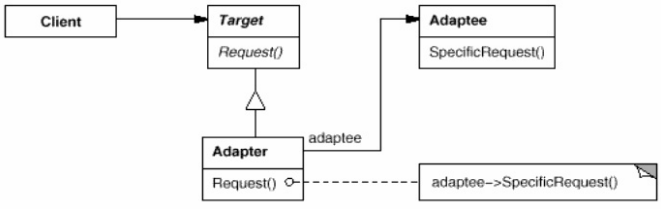

Figura 43: Diagramma del Design Pattern Adapter

- **Scopo:** Convertire l'interfaccia di una classe in un'altra interfaccia richiesta dal client. Consente a classi diverse di operare insieme quando ciò non sarebbe 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 pattern Adapter può essere utilizzato nei seguenti casi:
  - quando si vuole usare una classe esistente, ma la sua interfaccia non è compatibile con quella desiderata;
  - quando si vuole creare una classe riusabile in grado di cooperare con classi non correlate o impreviste, cioè con classi che non necessariamente hanno interfacce compatibili;
  - per gli oggetti adapter quando si devono utilizzare diverse sottoclassi esistenti, ma non è pratico adattare la loro interfaccia creando una sottoclasse per ciascuna di esse.

A.3.2 Facade

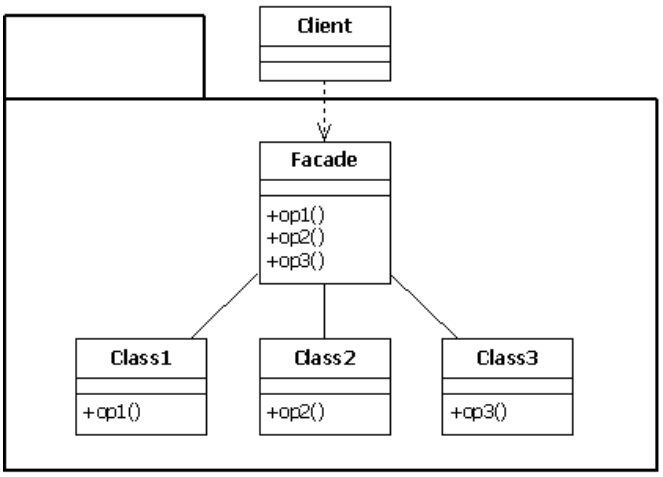

Figura 44: Diagramma del Design Pattern Facade

- **Scopo:** Fornire un'interfaccia unificata per un insieme di interfacce presenti in un sottosistema. Definisce un'interfaccia di livello più alto che rende il sottosistema più semplice da utilizzare;
- **Motivazione:** Suddividere un sistema in sottosistemi aiuta a ridurne la complessità. Un obiettivo comune di progettazione è la minimizzazione delle comunicazioni e delle dipendenza fra i diversi sottosistemi. Un modo per raggiungere questo obiettivo è introdurre un oggetto facade, che fornisce un'interfaccia unica e semplificata per accedere alle funzionalità offerte da un sottosistema;
- **Applicabilità:** Il pattern Facade può essere utilizzato nei seguenti casi:
  - Quando si vuole fornire un'interfaccia semplice a un sottosistema complesso poiché fornisce una vista semplice di base su un sottosistema che si rivela essere sufficiente per la maggior parte dei client;
  - Nei casi in cui ci sono molte dipendenze fra i client e le classi che implementano un'astrazione in quanto si disaccoppia il sottosistema dai client e dagli altri sistemi, promuovendo portabilità e indipendenza dei sottosistemi;
  - Quando si vogliono organizzare i sottosistemi in una struttura a livelli.

A.3.3 Decorator

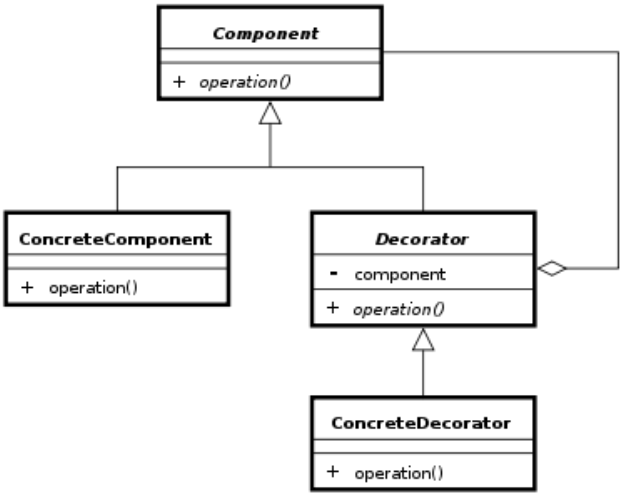

Figura 45: Diagramma del Design Pattern Decorator

Bridge

- **Scopo:** Aggiungere dinamicamente responsabilità a un oggetto. I Decorator forniscono un'alternativa flessibile alla definizione di sottoclassi come strumento per l'estensione delle funzionalità;
- **Motivazione:** Talvolta si vogliono aggiungere responsabilità a singoli oggetti e non a un'intera classe.

  Un modo per aggiungere responsabilità consiste nel racchiudere il componente da decorare in un altro. L'oggetto contenitore è chiamato Decorator. Il Decorator ha un interfaccia conforme a quella dell'elemento decorato, in modo da rendere trasparente la sua presenza ai client. Il decorator trasferisce le richieste al componente decorato e può svolgere azioni aggiuntive prima o dopo il trasferimento della richiesta;
- **Applicabilità:** Il pattern Decorator può essere utilizzato nei seguenti casi:
  - Si vuole poter aggiungere responsabilità a singoli oggetti dinamicamente ed in modo trasparente;
  - Si vuole poter togliere responsabilità agli oggetti;
  - si vuole definire un gran numero di estensioni indipendenti.

A.4 Design Pattern comportamentali

A.4.1 Command

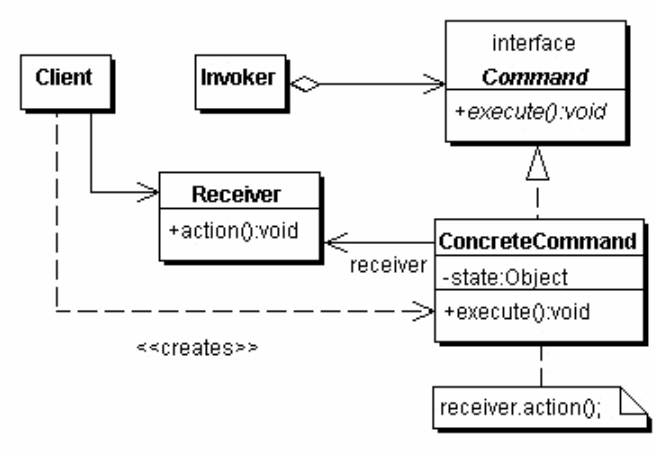

Figura 46: Diagramma del Design Pattern Command

- **Scopo:** Incapsula una richiesta in un oggetto, consentendo di parametrizzare i client con richieste diverse, accodare o mantenere uno storico delle richieste e gestire richieste cancellabili;
- **Motivazione:** Talvolta è necessario inoltrare richieste a oggetti senza conoscere nulla dell'operazione richiesta o del destinatario della richiesta. Il pattern Command permette agli oggetti dell'ambiente di inoltrare richieste a oggetti sconosciuti dell'applicazione trasformando la richiesta in un oggetto;
- **Applicabilità:** Il pattern Command può essere utilizzato nei seguenti casi:
  - Per parametrizzare gli oggetti rispetto a un'azione da compiere;
  - Per specificare, accordare ed eseguire le richieste in tempi diversi;
  - Per consentire l'annullamento di operazioni;
  - Per organizzare un sistema in operazioni d'alto livello a loro volta basate su operazioni primitive.

A.4.2 Strategy

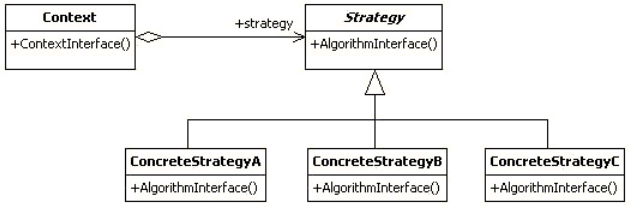

Figura 47: Diagramma del Design Pattern Strategy

- **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 per risolvere un problema. Codificare statisticamente ognuno di questi algoritmi nelle classi che ne fanno richiesta non é auspicabile per svariati motivi. Si possono evitare questi problemi definendo delle classi che incapsulano svariati algoritmi chiamati Strategy;
- **Applicabilità:** Il pattern Strategy può essere utilizzato nei seguenti casi:
  - Molte classi correlate differiscono fra loro solo per il comportamento;
  - Sono necessarie più varianti di un algoritmo;
  - Un algoritmo usa una struttura dati che non dovrebbe essere resa nota ai client;
  - Una classe definisce molti comportamenti che compaiono all'interno di scelte condizionali multiple.

B Mockup interfaccia grafica

B.1 Finestra principale

La finestra principale di 3DMob contiene il visualizzatore interattivo della scena$_G$ 3D (figura 48).
Le funzionalità di modifica della scena$_G$ sono raggruppate in ordine logico in dei `QDock Widget` che permettono all'utente di personalizzare l'interfaccia.
In particolare l'utente può:

- Aprire un nuovo file da importare;
- Selezionare un oggetto o una luce della scena$_G$;
- Modificare l'oggetto selezionato applicando rotazioni, traslazioni, ridimensionamenti;
- Modificare la luce selezionata applicando rotazioni o modifiche alle sue caratteristiche come lucentezza ed emissione;
- Aggiungere una nuova luce, scelta tra omnidirezionale e spotlight$_G$;
- Aggiungere un nuovo materiale all'oggetto selezionato;
- Navigare la scena$_G$.

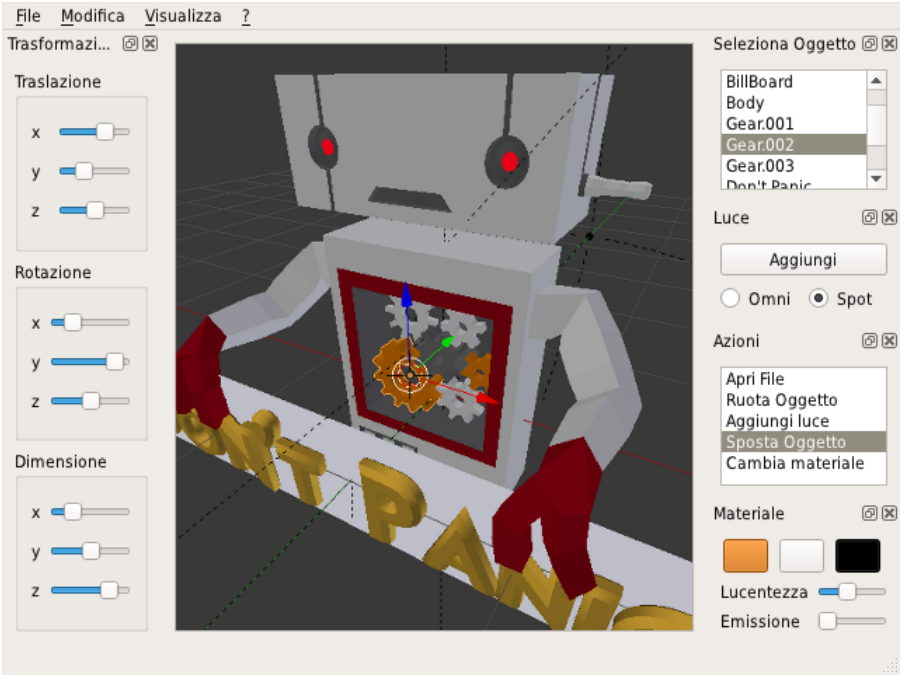

Figura 48: Mockup della finestra principale dell'applicazione 3DMob

Alcune funzionalità saranno raggiungibili anche da un menu a tendina presente nella finestra principale (figura 49).

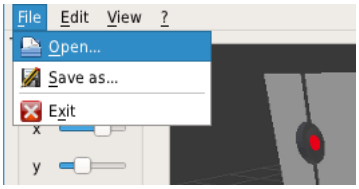

Figura 49: Mockup del menu file

B.2 Finestra di configurazione

Le impostazioni di sistema sono accessibili tramite l'apposita finestra (figura 50).
Tale finestra permette di configurare le impostazioni di sistema scegliendo se salvare la scena$_G$ esportando i dati in virgola mobile con precisione singola o doppia e impostando i limiti di esportazione che, di default, sono impostati su valori compatibili con iPhone 4S.

Figura 50: Mockup della finestra per la configurazione delle impostazioni

B.3 Finestra di selezione file da importare

Quando un utente decide di importare un file, viene visualizzata una finestra (figura 51) che permette di navigare il file system e di selezionare un file. L'utente può selezionare il file navigando nel file system e facendo un doppio click sul file desiderato o scrivendo per intero indirizzo e nome del file o solo nome del file se presente della directory corrente visualizzata nella finestra.

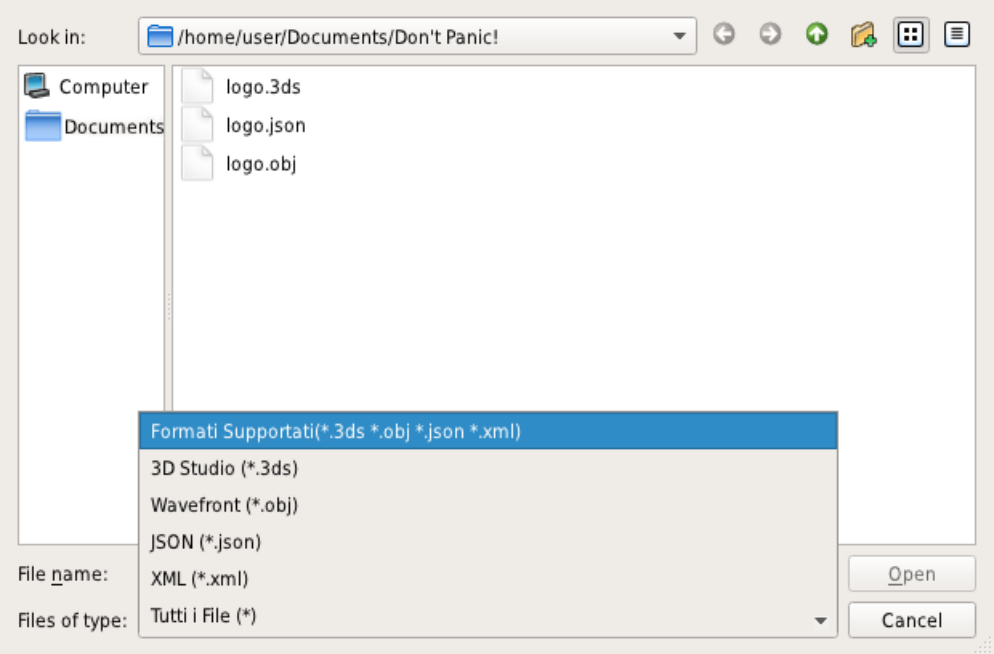

Figura 51: Mockup della finestra usata per selezionare il file in fase di importazione

B.4 Finestra di esportazione file

Quando un utente decide di esportare un file, viene visualizzata una finestra (figura 52) che permette di navigare il file system e di inserire il nome del file in cui verrà esportata la scena$_G$. L'utente può scrivere per intero indirizzo e nome del file o solo nome del file se presente della directory corrente visualizzata nella finestra o selezionare un file esistente se lo si vuole sovrascrivere.

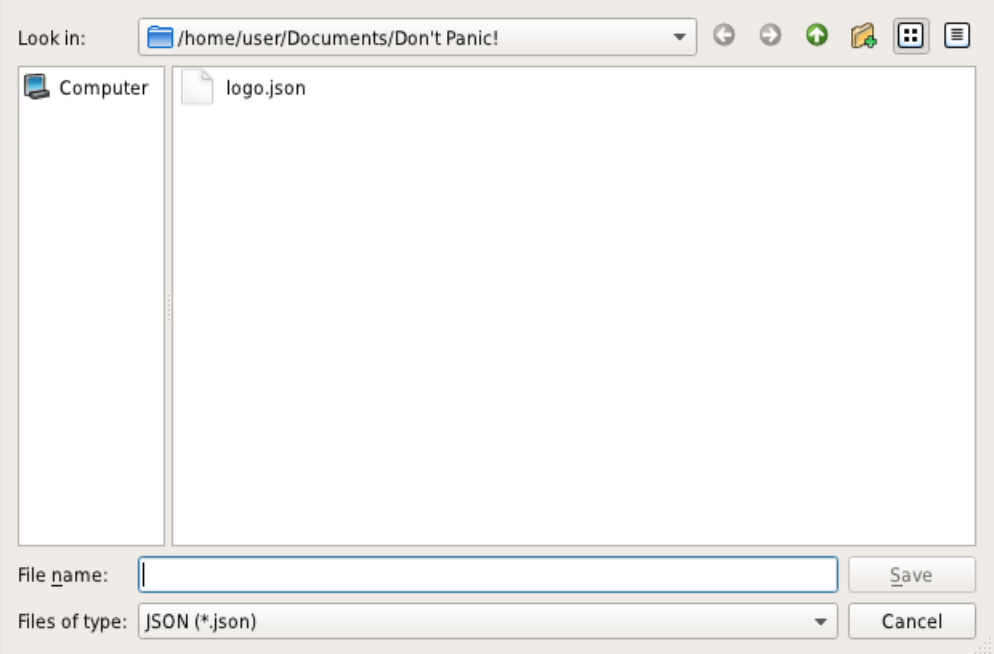

Figura 52: Mockup della finestra usata per selezionare il file in fase di esportazione

B.5 Finestra di aiuto

L'utente può accedere al sistema di help utilizzando l'apposita finestra (figura 53) che fornisce funzionalità di visualizzazione, ricerca e navigazione dei contenuti del manuale.

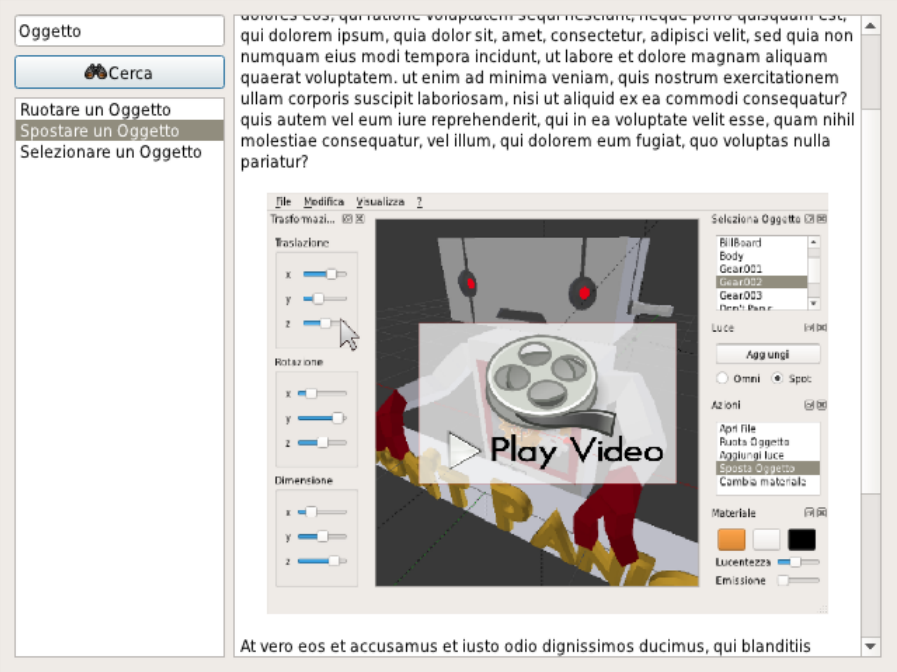

Figura 53: Mockup della finestra di aiuto

B.6 Finestra di informazioni sul sistema

L'utente può accedere alle informazioni di sistema che vengono visualizzate tramite l'apposita finestra (figura 54) che fornisce informazioni sul sistema quali ad esempio la licenza e la versione.

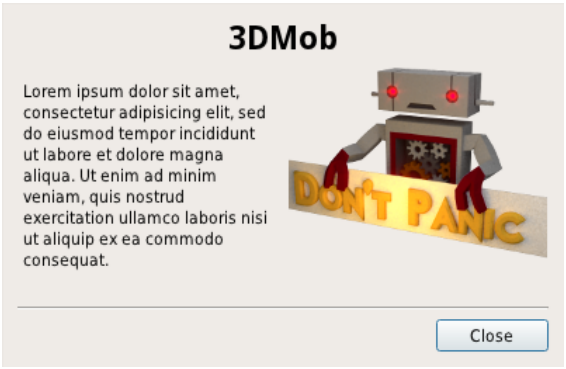

Figura 54: Mockup della finestra di informazioni di sistema

B.7 Finestra dei messaggi di sistema

L'utente riceve messaggi informativi dal sistema a mezzo di una finestra di popup$_G$ (figura 55).

Questi messaggi riguardano lo stato del sistema, dei file aperti e delle conversioni. Nessuno di questi messaggi è di natura bloccante per il sistema. Tra i messaggi che potrebbero essere segnalati troviamo ad esempio la segnalazione del caricamento di un file con una scena$_G$ che eccede i limiti impostati nel sistema.

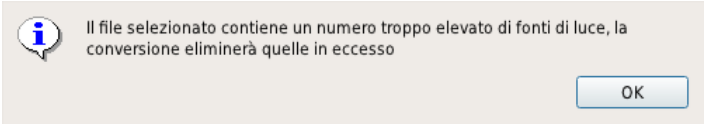

Figura 55: Mockup della finestra di messaggio di sistema

B.8 Finestra degli errori di sistema

L'utente riceve messaggi di errori dal sistema a mezzo di una finestra di popup$_G$ (figura 56).
Questi messaggi di errore del sistema sono di natura bloccante in relazione all'azione appena eseguita dall'utente o attività del sistema in corso. Tra gli errori che potrebbero essere segnalati troviamo l'errore di formato del file da aprire o l'errore nel salvataggio.

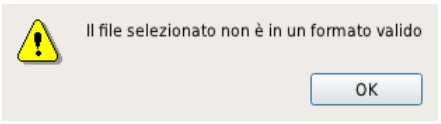

Figura 56: Mockup della finestra di errore di sistema