Manuale Sviluppatore

Informazioni Documento

Versione	0.0.0
Data approvazione	10 Marzo 2018
Responsabile	Marco Focchiatti
Redattori	Manfredi Smaniotto, Marco Focchiatti, Cristiano Tessarolo, Giulio Rossetti, Kevin Silvestri
Verificatori	Manfredi Smaniotto, Marco Focchiatti
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo Graphite
Uso	Esterno
Recapito	graphite.swe@gmail.com

Registro delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
2.0.0	10-03-2018	Marco Focchiatti	Responsabile	Approvazione
1.2.0	09-03-2018	Giulio Rossetti	Verificatore	Verifica
1.1.1	08-03-2018	Kevin Silvestri	Verificatore	Stesura appendice §C.3 e §D
1.1.0	25-02-2018	Marco Focchiatti	Verificatore	Verifica
1.0.6	23-02-2018	Manfredi Smaniotto	Verificatore	Stesura appendice §B
1.0.5	22-02-2018	Manfredi Smaniotto	Verificatore	Spostamento appendice §B in appendice §C
1.0.4	16-02-2018	Giulio Rossetti	Verificatore	Rivisto e modificato §3
1.0.3	13-02-2018	Cristiano Tessarolo	Verificatore	Rivisti obiettivi di qualità (§2.2) e aggiunta politica della qualità (§2.4)
1.0.2	11-02-2018	Kevin Silvestri	Verificatore	Spostate definizioni metriche (§3) in NP
1.0.1	09-02-2018	Marco Focchiatti	Verificatore	Rivista struttura generale e ampliata §2
1.0.0	12-01-2018	Samuele Modena	Responsabile	Approvazione
0.2.0	11-01-2018	Giulio Rossetti	Verificatore	Verifica
0.1.2	10-01-2018	Samuele Modena	Verificatore	Stesura appendice §B
0.1.1	20-12-2017	Matteo Rizzo	Verificatore	Aggiornata §3
0.1.0	19-12-2017	Manfredi Smaniotto	Verificatore	Verifica
0.0.6	18-12-2017	Kevin Silvestri	Verificatore	Stesura appendice §A
0.0.5	17-12-2017	Kevin Silvestri	Verificatore	Stesura §4
0.0.4	15-12-2017	Matteo Rizzo	Verificatore	Stesura §3
0.0.3	14-12-2017	Samuele Modena	Verificatore	Stesura §2
0.0.2	13-12-2017	Matteo Rizzo	Verificatore	Stesura §1
0.0.1	13-12-2017	Matteo Rizzo	Verificatore	Creazione del template

Indice

1. Introduzione

1.1 Scopo del documento

Il documento ha la finalità di illustrare a coloro che volessero interfacciarsi con l'applicazione *"DeSpeect: un'interfaccia grafica per Speect"* le modalità di installazione e di utilizzo, i requisiti necessari per poterlo utilizzare, le librerie e i framework esterni utilizzati per lo sviluppo dell'applicazione, oltre alla sua architettura, così da aiutare nella ricerca o eventuale estensione delle sue funzionalità. Nonostante la versione attuale rappresenti una prima bozza del documento, una volta concluso esso rappresenterà un riferimento completo per l'utilizzo del prodotto da parte di uno sviluppatore.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un'interfaccia grafica per *Speect*$_{\mathrm{G}}$ [Meraka Institute(2008-2013)], una libreria per la creazione di sistemi di sintesi vocale, che agevoli l'ispezione del suo stato interno durante il funzionamento e la scrittura di test per le sue funzionalità.

1.3 Informazioni utili

La stesura di questo documento assume come utente target del prodotto un programmatore esperto nell'utilizzo di *Speect* e dei linguaggi di programmazione C e C++. Per completezza, viene riportato in appendice §A un glossario comprensivo di termini tecnici o riguardanti particolari funzionalità di *DeSpeect*. Per identificare i termini presenti nel glossario, la loro prima occorrenza all'interno del documento è riportata in corsivo e marcata con una G al pedice.

1.4 Riferimenti informativi

- **Documentazione Speect:**
  http://speect.sourceforge.net/contents.html;

  Documentazione ufficiale della libreria di *Text-To-Speech* di riferimento per il progetto.

- **Documentazione Qt:**
  http://doc.qt.io/;

  Documentazione ufficiale del framework utilizzato per lo sviluppo dell'interfaccia grafica.

- **Documentazione CMAKE:**
  https://cmake.org/documentation/.

  Documentazione ufficiale del framework utilizzato per la build del prodotto.

2. Requisiti di sistema

L'installazione ed esecuzione del software DeSpeect richiede i seguenti prerequisiti:

- Sistema operativo Unix / Unix-like (il software è stato testato solo per piattaforma Ubuntu 16.04 LTS)

  https://www.ubuntu.com/download/desktop

- CMake (versione minima 2.8)

  https://cmake.org/download/

- Compilatore ANSI C/ISO C90 GCC (versione minima 5.0)

  https://gcc.gnu.org/install/binaries.html

- Qt 5.9.0

  https://www.qt.io/download

- Git

  https://git-scm.com/

- Curl

  https://curl.haxx.se/

- Swig

  http://www.swig.org/

- libxml2-dev

  https://packages.debian.org/stretch/libxml2-dev

- python-dev

  https://pypi.python.org/pypi/dev/0.4.0

3. Tecnologie impiegate

Vengono di seguito illustrate le tecnologie impiegate dal software DeSpeect e nella verifica e monitoraggio del suo codice.

3.1 Tecnologie utilizzate da DeSpeect

3.1.1 Libreria di riferimento

Speect è un sistema di *Text To Speech* (TTS) multilingua. Esso offre un sistema TTS completo (analisi e decodifica del testo e sintesi vocale) con annesse varie API, nonché un ambiente per la ricerca e lo sviluppo di sistemi e voci TTS. Speect è scritto in linguaggio C, con una stretta conformità allo standard ISO / IEC 9899: 1990, consentendo così la massima portabilità su diverse piattaforme di calcolo. Le chiamate di sistema specifiche della piattaforma sono astratte per consentire le porte a nuove piattaforme. Speect v1.1.0-69-g65f4 rappresenta il cuore di DeSpeect, che in particolare ne realizza un'interfaccia grafica atta a semplificarne l'utilizzo e il debug. La tecnologia è open source e la sua documentazione è accessibile al seguente link:

http://speect.sourceforge.net/contents.html

3.1.2 IDE

QT è una libreria multipiattaforma per lo sviluppo di programmi con interfaccia grafica tramite l'uso di widget (congegni o elementi grafici). La libreria è scritta in C++ e gode di ampia diffusione e supporto. L'interfaccia grafica di DeSpeect è stata sviluppata mediante questa tecnologia nella versione v5.9 LTS. Maggiori informazioni su Qt sono reperibili al seguente link:

https://www.qt.io/what-is-qt/

Tale IDE è consigliato qualora si volesse lavorare su DeSpeect, e un approfondimento sulla sua installazione e sull'importazione del progetto è reperibile in §4.2 di questo documento.

3.1.3 Compilazione

Per la compilazione di DeSpeect vengono utilizzati i seguenti strumenti:

- **GCC**: il compilatore che viene usato per la compilazione del software è il *GCC*$_{\text{G}}$ (GNU Compiler Collection). Maggiori informazioni sul compilatore sono reperibili al seguente link:

  https://gcc.gnu.org/

- **CMake**: CMake è una famiglia di strumenti open source e multipiattaforma progettati per creare, testare e pacchettizzare software. CMake viene utilizzato per controllare il processo di compilazione del software utilizzando semplici file di configurazione indipendenti dalla piattaforma e dal compilatore e generare makefile e aree di lavoro nativi che possono essere utilizzati nell'ambiente del compilatore di propria scelta. DeSpeect utilizza questa tecnologia nella versione v3.10.2 per l'automazione della compilazione. Maggiori informazioni su CMake sono reperibili al seguente link:

  https://cmake.org/overview/

3.1.4 GUI

Per progettare l'interfaccia grafica viene utilizzato *Qt Creator*$_{\text{G}}$. Questo strumento permette di realizzare interfacce grafiche mediante le librerie grafiche Qt, diventate in questo ambito quasi uno standard per piattaforme Linux Based. Maggiori informazioni su Qt Creator sono disponibili al seguente link::

https://www.qt.io/qt-features-libraries-apis-tools-and-ide/

3.2 Verifica e monitoraggio di DeSpeect

3.2.1 Test

Google Test è un framework per la realizzazione di test per il linguaggio C++. DeSpeect utilizza questa tecnologia per la realizzazione dei test del software, ulteriori informazioni sulla stessa sono reperibili al seguente link:

https://github.com/google/googletest/blob/master/googletest/docs/Primer.md

3.2.2 Verifica della build e testing automatico

Travis CI è un servizio di integrazione continua distribuito utilizzato per costruire e testare progetti software ospitati su GitHub. I progetti open source possono essere testati gratuitamente attraverso il sito web travis-ci.org. Questa tecnologia viene utilizzata per l'esecuzione di test automatici su DeSpeect a seguito del caricamento di codice sul repository relativo al progetto, così da garantirne la correttezza. Documentazione relativa a Travis CI è accessibile al seguente link:

https://docs.travis-ci.com/

3.2.3 Misurazioni metriche

Per il controllo delle varie metriche legate al codice vengono utilizzati i software integrati con il repository GitHub:

- **Better Code Hub**: *Better Code Hub*$_G$ è un servizio di analisi del codice sorgente web-based che controlla il codice per la conformità rispetto a 10 linee guida per l'ingegneria del software e fornisce un *feedback*$_G$ immediato per capire dove concentrarsi per miglioramenti di qualità. Better Code Hub è accessibile al seguente link:

  https://bettercodehub.com/

- **Codacy**: Codacy è uno strumento di analisi / qualità del codice automatizzato, con cui si ottengono analisi statiche, complessità ciclomatica, indicazioni sulla duplicazione e le variazioni della copertura dei test dell'unità di codice in ogni richiesta di commit e pull. È inoltre possibile utilizzare Codacy per applicare uno standard di qualità del codice e applicare le best practice sulla sicurezza, il tutto integrato con

GitHub. Ulteriori informazioni su Codacy sono disponibili al seguente link:

https://www.codacy.com/features

4. Ambiente di sviluppo

Vengono di seguito illustrate le modalità di installazione e configurazione del software DeSpeect, nonché le modalità di installazione e importazione del progetto in relazione all'IDE consigliato per lavorare con lo stesso.

4.1 Installazione e configurazione

DeSpeect è reperibile su *GitHub*$_{\text{G}}$ al seguente link:

https://github.com/graphiteSWE/DeSpeect

Una volta soddisfatti i prerequisiti descritti in §2 "Requisiti di sistema" di questo documento, per installare ed eseguire il software è necessario seguire la seguente procedura:

1. Clonare o scaricare la repository sulla propria macchina;
2. Entrare nella cartella scaricata ed eseguire lo script `build.sh`.

Tale procedura installerà la libreria Speect e genererà una build del software nella directory `DeSpeect/build`, nonché avvierà automaticamente un'esecuzione di DeSpeect.

4.2 IDE

Il progetto DeSpeect non è necessariamente legato ad alcun *IDE*$_{\text{G}}$, tuttavia si consiglia l'uso di *Qt*$_{\text{G}}$ e *Qt Creator*. Le librerie Qt sono infatti utilizzate all'interno del prodotto per la realizzazione dell'interfaccia grafica, e l'adozione dei succitati IDE ne agevola notevolmente l'impiego nell'ottica di modifiche e/o estensioni alla stessa. Viene di seguito illustrata la procedura per l'installazione dell'IDE e l'importazione del progetto.

4.2.1 Installare Qt

Il pacchetto d'installazione di Qt è reperibile gratuitamente per progetti open source al seguente link:

https://www.qt.io/download

Una volta eseguito l'installer, previa configurazione dei pacchetti d'installazione, l'IDE sarà installato sulla macchina desiderata. Ulteriori informazioni sull'installazione e procedure alternative sono reperibili al seguente link:

http://doc.qt.io/qt-5/linux.html

4.2.2 Importare il progetto

Per importare i file di DeSpeect all'interno dell'IDE, è necessario seguire la seguente procedura:

1. Installare Speect sulla propria macchina. Un modo per farlo consiste nell'eseguire lo script `install.sh` contenuto all'interno della directory `SpeectInstaller/` di DeSpeect;
2. Aprire l'IDE installato seguendo le indicazioni riportate nella precedente sezione (§3.2.1);
3. Dalla barra dei menù selezionare "File / Apri progetto" (ctrl + o);
4. Selezionare il file `CMakeLists.txt` contenuto nella directory `DeSpeect/`;
5. Confermare l'importazione all'interno dell'IDE.

5. Architettura generale del prodotto

L'architettura generale del prodotto segue il pattern Model-View-ViewModel. Questo pattern è basato su tre componenti principali:

- **Model**: un'implementazione del modello del dominio dei dati;
- **View**: la struttura, il layout e l'aspetto di ciò che l'utente visualizza a schermo;
- **ViewModel**: un'astrazione della view che espone proprietà pubbliche e comandi.

Esso permette tra le altre cose un totale disaccoppiamento tra logica di business e presentazione, informazioni più dettagliate a riguardo sono reperibili nell'appendice §A "Model-View-ViewModel" di questo documento. I seguenti diagrammi illustrano sinteticamente la struttura del software attraverso i package che lo costituiscono, con livello di dettaglio crescente. Tali diagrammi sono consultabili anche all'interno del repository nella directory `AllegatoTecnico/Diagrammi/DiagrammiPackage/`.

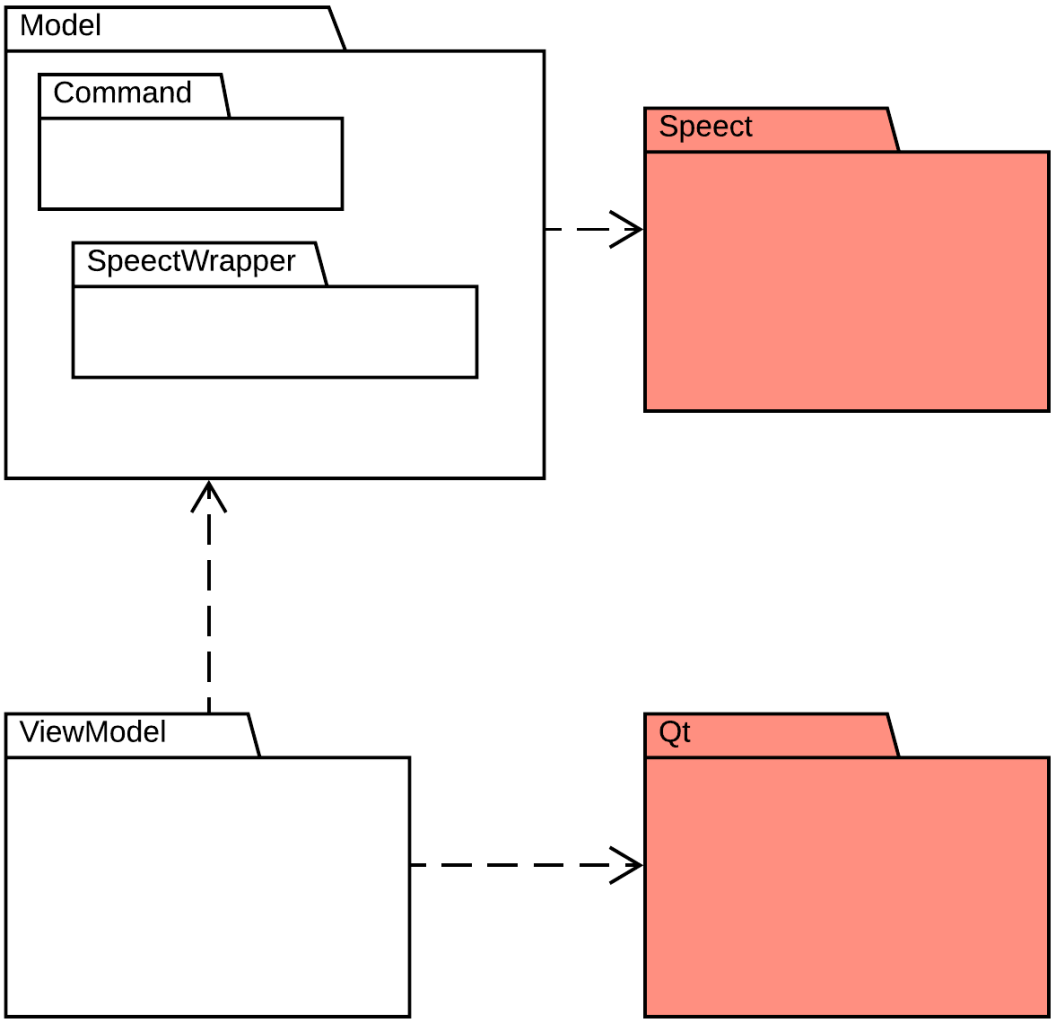

Figura 5.1: Diagramma generale dei package

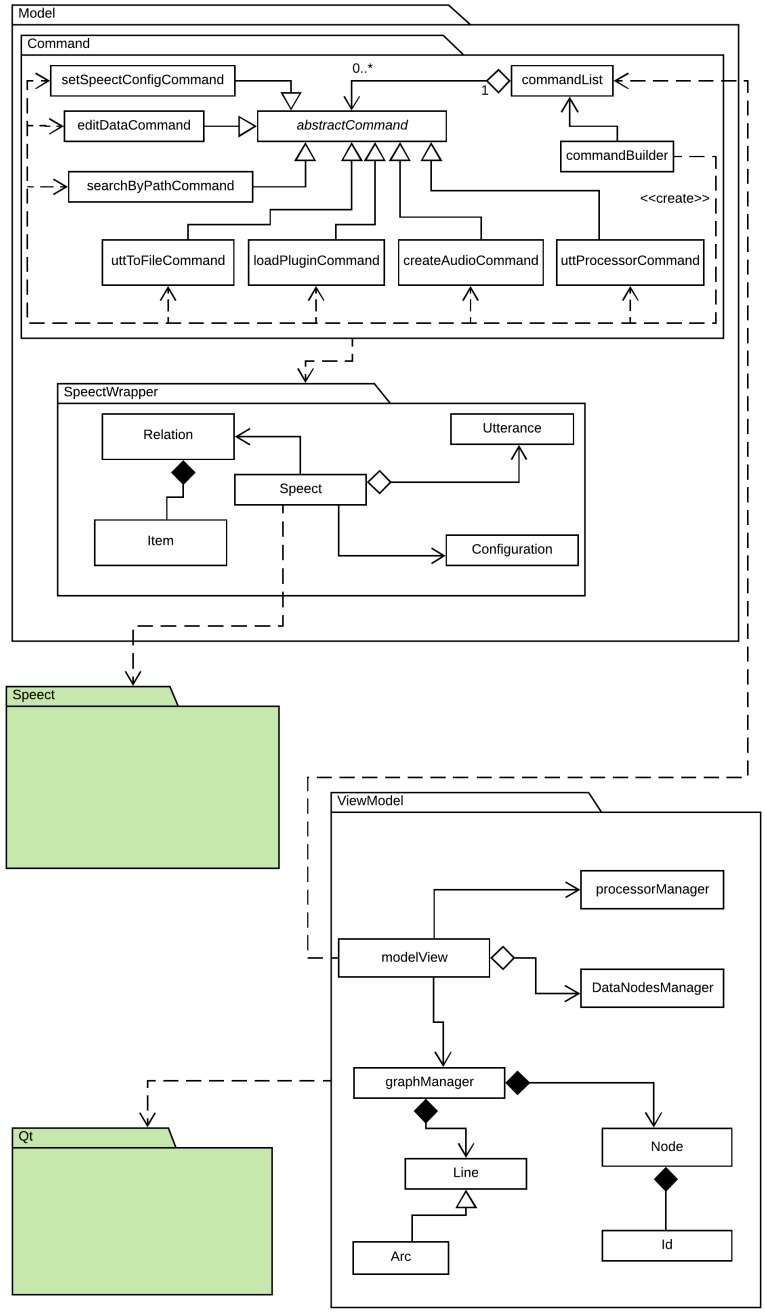

Figura 5.2: Diagramma dei package nel dettaglio

6. Implementazione MVVM

Vengono di seguito illustrate le implementazioni per le componenti del pattern Model-View-ViewModel in relazione all'architettura della Product Baseline. Per ogni componente, vengono illustrati:

- **Contestualizzazione**: spiegazione generale dell'architettura del componente all'interno del sistema.
- **Diagramma delle classi**: diagramma generale delle classi per il componente;
- **Dettaglio delle classi**
- **Design Pattern**: una descrizione e contestualizzazione esaustiva dei design pattern impiegati all'interno del componente.

6.1 View

La View, conseguentemente all'uso del framework Qt, consiste di un file *qml* trasformato durante la compilazione in classi compatibili C++. Il comportamento della View è indi gestito dal package ViewModel.

6.2 Model

6.2.1 Contestualizzazione

Nella progettazione del Model è emersa la necessità di interagire con la libreria Speect incapsulandone alcune funzionalità rilevanti. Per realizzare ciò il Model è stato diviso in due package corrispondenti all'implementazione dei design pattern Façade (SpeectWrapper), per quanto riguarda l'incapsulamento della libreria, e Command, per quanto riguarda la suddivisione delle funzionalità implementate in un'ottica di componibilità ed estendibilità.

6.2.2 Diagramma delle classi

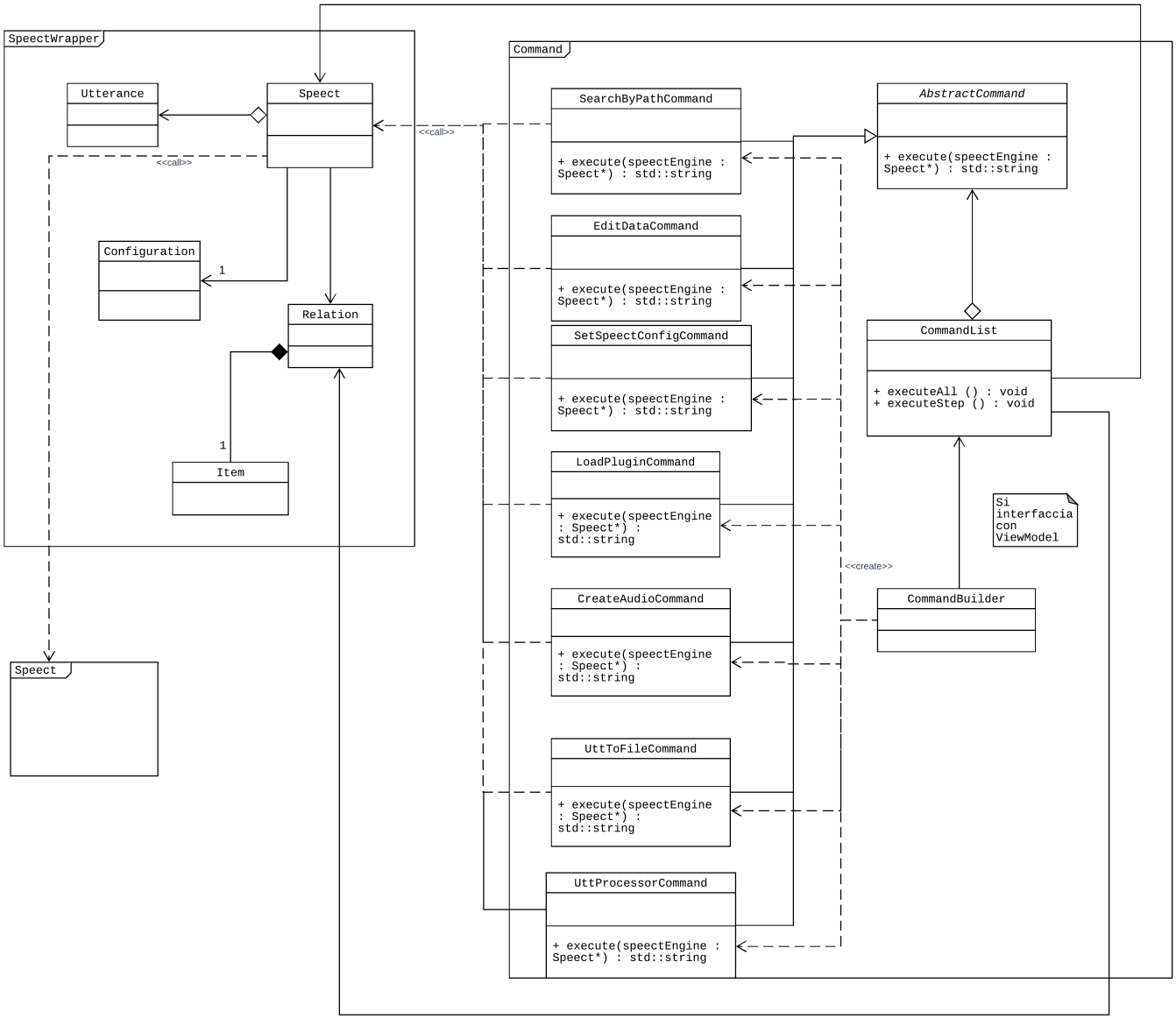

Figura 6.1: Diagramma delle classi del componente Model

6.2.3 Dettaglio delle classi

Nelle sezioni successive vengono illustrate le classi costituenti il componente Model del prodotto. Tali classi sono aggregate in due principali package:

- **SpeectWrapper**: incapsula la libreria Speect gestendone la memoria.
- **Command**: offre la possibilità di creare i comandi che, aggregando funzioni Speect, rappresentano le funzionalità base di DeSpeect.

6.2.3.1 SpeectWrapper

- **Speect**: incapsula le componenti base della libreria Speect e ne gestisce la memoria;
- **Configuration**: incapsula la configurazione della libreria Speect per garantirne il corretto funzionamento;
- **Utterance**: incapsula i dati relativi all'utterance e ne gestisce la memoria;
- **Relation**: incapsula i dati relativi alla relation e ne gestisce la memoria;
- **Item**: un'iteratore per scorrere una relation di Speect, fornendo inoltre un path relativo al nodo di partenza per raggiungere il nodo attualmente selezionato.

6.2.3.2 Command

- **AbstractCommand**: definisce l'interfaccia base di un comando;
- **CommandList**: aggrega i comandi in una lista, di cui permette l'esecuzione completa o step-by-step;
- **CommandBuilder**: permette la costruzione di una CommandList;
- **SearchByPathCommand**: implementazione del comando che esegue la ricerca di un nodo dato un path;
- **EditDataCommand**: implementazione del comando che esegue la modifica dei dati relativi ad un nodo;
- **SetSpeectConfigCommand**: implementazione del comando che esegue l'impostazione di una data configurazione di Speect;
- **LoadPluginCommand**: implementazione del comando che esegue il caricamento di un plugin;
- **CreateAudioCommand**: implementazione del comando che genera il file audio dall'utterance attuale;

- **UttToFileCommand**: implementazione del comando che esegue il salvataggio dell'utterance in un file secondo il formato desiderato;
- **UttProcessorCommand**: implementazione del comando che esegue un dato utterance processor.

6.2.4 Design pattern

6.2.4.1 Command

Permette la suddivisione delle funzionalità implementate in un'ottica di componibilità ed estendibilità. I comandi concreti sono aggregati in una lista (`CommandList`) che si interfaccia con la componente ViewModel. Tali comandi interagiscono a loro volta con il package SpeectWrapper per ottere i dati da elaborare dalla libreria Speect. Ai comandi è delegata l'esecuzione degli utterance processor per la successiva stampa dei dati nel grafo, ma anche l'implementazione di funzionalità quali il caricamento dei plug-in e della configurazione di Speect.

6.2.4.2 Builder

Questo design pattern separa la costruzione di un oggetto complesso dalla sua rappresentazione, cosicché il processo di costruzione stesso possa creare diverse rappresentazioni. Contestualizzato nel sistema della PB, esso si interfaccia con il ViewModel per la configurazione e costruzione di una specifica `CommandList`.

6.2.4.3 Façade

Il design pattern Façade permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi. In questo contesto, il package SpeectWrapper incapsula la libreria Speect rendendola accessibile tramite omonima classe proprietaria.

6.3 ViewModel

6.3.1 Contestualizzazione

Il package ViewModel funge da tramite tra Model e View, prelevando i dati dal primo per aggiornare il secondo. Per quanto riguarda la stampa del grafo, la classe `GraphManager` si occupa di aggiornarne la presentazione interfacciandosi con le librerie Qt e con le classi `Line`, `Arc` e `Node`.

6.3.2 Diagramma delle classi

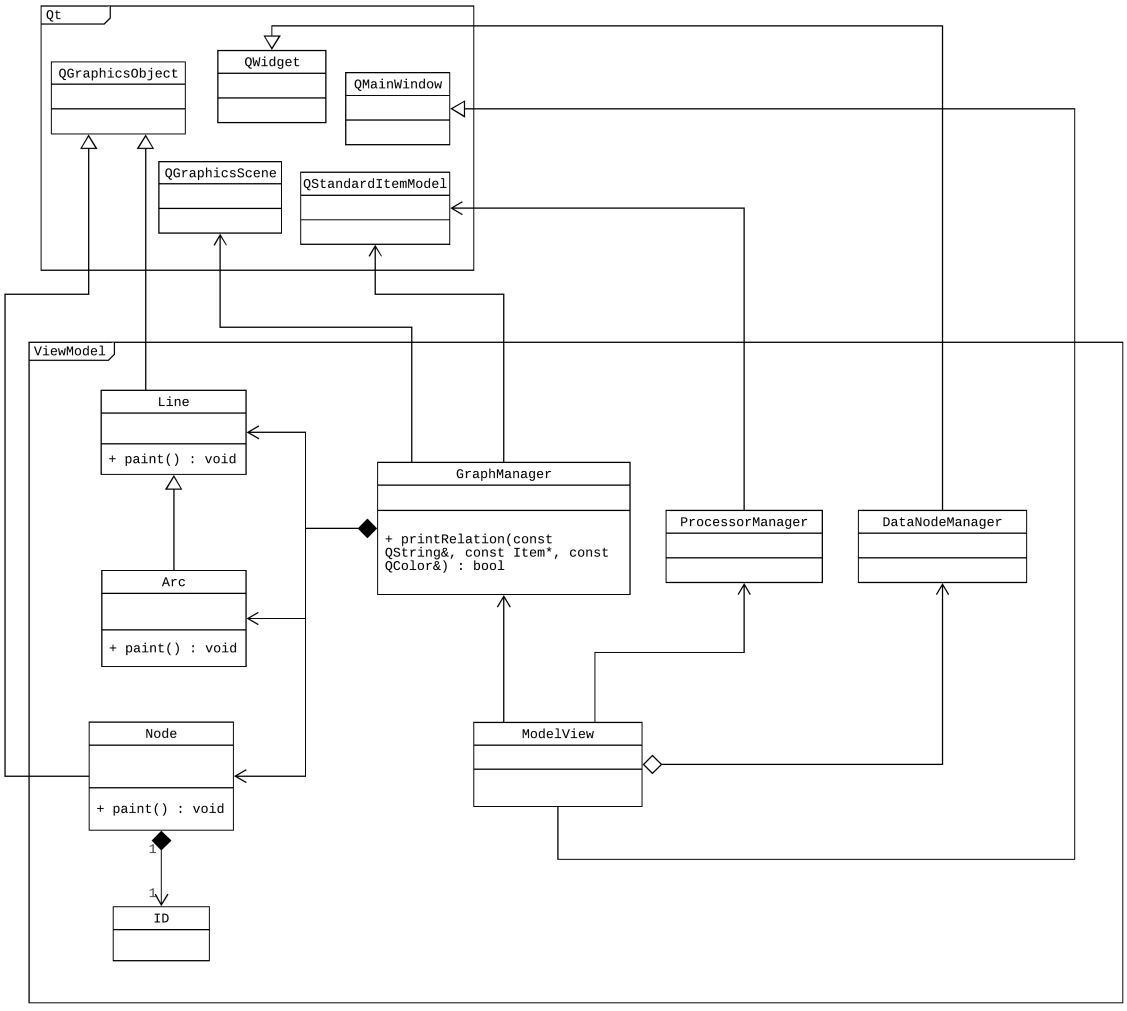

Figura 6.2: Diagramma delle classi del componente ViewModel

6.3.3 Dettaglio delle classi

Nelle sezione successive verranno illustrate le classi che costituiscono il package ViewModel.

6.3.3.1 ViewModel

- **ModelView**: gestisce le richieste fatte dal client tramite la View;
- **GraphManager**: gestisce il modello del grafo;
- **ProcessorManager**: gestisce la selezione dei processor per l'esecuzione;
- **DataNodeManager**: gestisce i dati relativi ad uno specifico nodo;
- **Line**: modella una linea;
- **Arc**: modella un arco orientato del grafo;
- **Node**: modella un nodo del grafo;
- **ID**: rappresenta una struttura dati che definisce l'identificatore specifico di un nodo.

6.3.4 Design pattern

6.3.4.1 Observer

Il framework Qt, attraverso il sistema di signal e slot, implementa tale design pattern. Esso permette di reagire efficientemente ad un cambiamento dell'interfaccia grafica, chiedendo se necessario l'aggiornamento dei dati del Model attraverso il ViewModel. Il meccanismo di signal e slot è implementato dalle classi pertinenti all'interno del package.

7. Estensione delle funzionalità

7.1 Modificare l'interfaccia grafica

Per modificare l'attuale interfaccia grafica, è sufficiente apportare le modifiche desiderate al file qml `view.ui` contenuto in `DeSpeect/Qt/src/`. Le modifiche da apportare a tale file vanno eseguite tramite Qt Creator, ulteriori informazioni riguardo i file qml sono reperibili al seguente link:

https://doc.qt.io/qt-5.10/qtqml-index.html

Qualora le modifiche all'interfaccia grafica corrispondano all'implementazione di nuove funzionalità, è necessario estendere i package Model e ViewModel per garantirne la coerenza.

7.2 Aggiungere un plug-in

L'architettura di DeSpeect prevede un apposito comando per il caricamento di nuovi plug-in all'interno del package Command. Qualora il plug-in richiedesse l'invocazione di metodi specifici, sarà necessario incapsularli in un nuovo comando che definisce la corretta procedura d'esecuzione del plug-in stesso. Sarà dunque necessario implementare un nuovo comando concreto estendendo la classe AbstractCommand.

8. Verifica delle estensioni

8.1 Aggiungere un test

Quando DeSpeect viene esteso con una nuova funzionalità è opportuno creare i relativi test di unità per verificarne la correttezza. I test per DeSpeect sono implementati mediante la tecnologia Google Test, come citato in §3.2.1 di questo documento. Per completezza viene di seguito riportato il link alla documentazione di tale tecnologia:

https://github.com/google/googletest/blob/master/googletest/docs/Primer.md

Si rimanda a tale documentazione per la corretta sintassi e configurazione dei file di test, che vanno inseriti nella cartella `DeSpeect/Test/` del progetto affinché vengano correttamente eseguiti automaticamente da Travis CI (tecnologia approfondita in §3.2.2 di questo documento).

8.2 Eseguire un test

8.2.1 Esecuzione automatica dei test

Il servizio Travis CI, integrato con il repository GitHub relativo al progetto, testa la build ed esegue i test automaticamente ad ogni modifica del codice sul repository. Lo stesso rilascia successivamente i dati sul *code coverage* al servizio web *coveralls.io*, anch'esso collegato al repository GitHub, che aggiorna così i propri report. É possibile prendere spunto dal file `.travis.yml` presente nel repository e dalla documentazione reperibile al link:

https://docs.travis-ci.com/user/languages/cpp/

Per predisporre il proprio sistema di integrazione continua con Travis CI qualora non si volesse creare una fork del progetto originale. Per la corretta integrazione del testing con Google Test e coveralls.io si suggerisce la presa visione della seguente documentazione esemplificativa:

https://github.com/bast/gtest-demo

8.2.2 Esecuzione manuale dei test

A seguito dell'esecuzione dello script `build.sh` citato in §4.1 di questo documento viene generata automaticamente una build e un'esecuzione dei test contenuti nella directory `DeSpeect/Test/`. I test sono eseguibili sia con il comando `ctest` che con il comando `unit_tests` e la loro esecuzione mostra sinteticamente il code coverage relativo.

A. Model-View-ViewModel

A.1 Struttura del pattern

Il design pattern architetturale *Model-View-ViewModel* (MVVM in breve) facilita la separazione dell'interfaccia grafica, che si tratti di linguaggio di markup o codice GUI, dallo sviluppo della logica di business o della logica di back-end, ovvero dal modello dei dati. Il *view model* di MVVM è un convertitore di valori, nel senso che il view model è responsabile dell'esposizione (conversione) degli oggetti dati dal modello così da renderli facilmente gestibili e presentabili. In quest'ottica, la view è più un modello che una vista e gestisce la maggior parte se non tutta la logica di visualizzazione della vista. Il pattern è riassunto dal seguente schema ed i suoi componenti principali sono di seguito approfonditi.

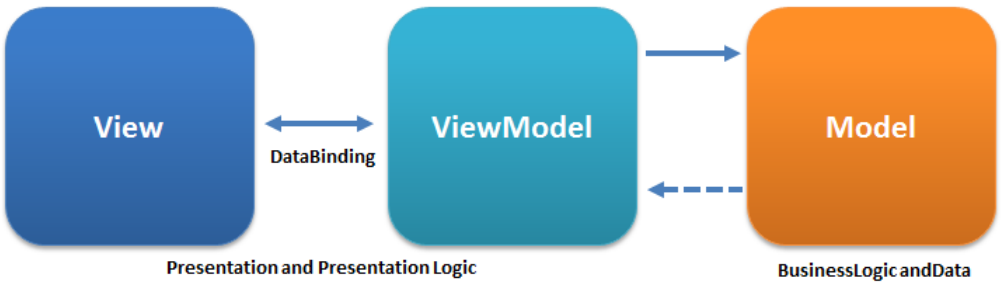

Figura A.1: Diagramma generale dell'architettura MVVM

I tre componenti principali dell'architettura sono i seguenti:

- **Model**: il *Model* (o modello) è un'implementazione del modello di dominio dell'applicazione ed include un modello dei dati affiancato alla logica di business e di validazione;
- **View**: la *View* (o vista) è responsabile della definizione della struttura, del layout e dell'aspetto di ciò che l'utente visualizza su schermo.

Idealmente, la vista è definita puramente con linguaggio di markup o generico codice GUI che non contiene la logica di business;

- **ViewModel**: la *ViewModel* (o modello di presentazione) funge da intermediario tra la vista e il modello ed è responsabile della gestione della logica di visualizzazione. In genere, il ViewModel interagisce con il modello richiamandone i metodi delle classi: esso fornisce quindi dati dal modello in una forma facilmente utilizzabile dalla View. Il ViewModel recupera i dati dal modello, rendendoli disponibili alla View, e può riformattarli in un modo che renda più semplice la gestione della vista. Esso fornisce anche l'implementazione dei comandi che un utente dell'applicazione avvia nella vista (ad esempio, quando un utente clicca un pulsante nell'interfaccia grafica, tale azione può attivare un comando nel ViewModel) e può essere responsabile della definizione delle modifiche dello stato logico che influiscono su alcuni aspetti della visualizzazione nella vista, ad esempio l'indicazione che alcune operazioni sono in sospeso.

A.2 Vantaggi offerti dal pattern

Il MVVM offre i seguenti vantaggi:

- Durante il processo di sviluppo, i programmatori e i designer possono lavorare in modo indipendente e simultaneamente sui loro componenti. Quest'ultimi possono concentrarsi sulla vista e, utilizzando appositi strumenti, generare facilmente dati di esempio con cui lavorare, mentre i programmatori possono lavorare sul modello di presentazione e sui componenti del modello;

- Gli sviluppatori possono creare test unitari per il ViewModel e per il Model senza utilizzare la View;

- È facile riprogettare l'interfaccia grafica dell'applicazione senza toccare il resto del codice, una nuova versione della vista dovrebbe poter funzionare con il modello di presentazione esistente;

- Se esiste un'implementazione del modello che incapsula la logica di business, potrebbe essere difficile o rischioso cambiarla. In questo scenario, il ViewModel funge da adattatore per le classi del Model e consente di evitare modifiche importanti al codice dello stesso.

B. Contribuire a DeSpeect

DeSpeect è software open source e permette quindi di essere modificato. Se si vuole contribuire al progetto originale è possibile farlo mediante la creazione di fork sul repository GitHub. Risulta inoltre possibile aprire dei ticket per segnalare errori, malfunzionamenti o la richiesta di nuove funzionalità.

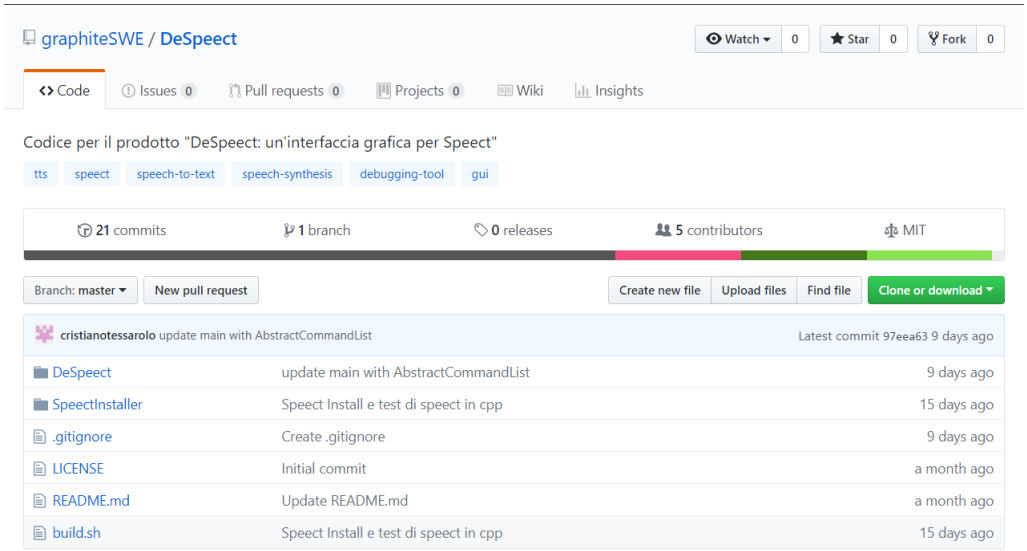

Figura B.1: Diagramma delle classi del componente ViewModel

B.1 Aprire un ticket

Per aprire un *ticket* è sufficiente recarsi nella sezione *Issues* del repository GitHub del progetto e cliccare il pulsante *New Issue*. A questo punto verrà chiesto di inserire un titolo e una descrizione per il ticket.

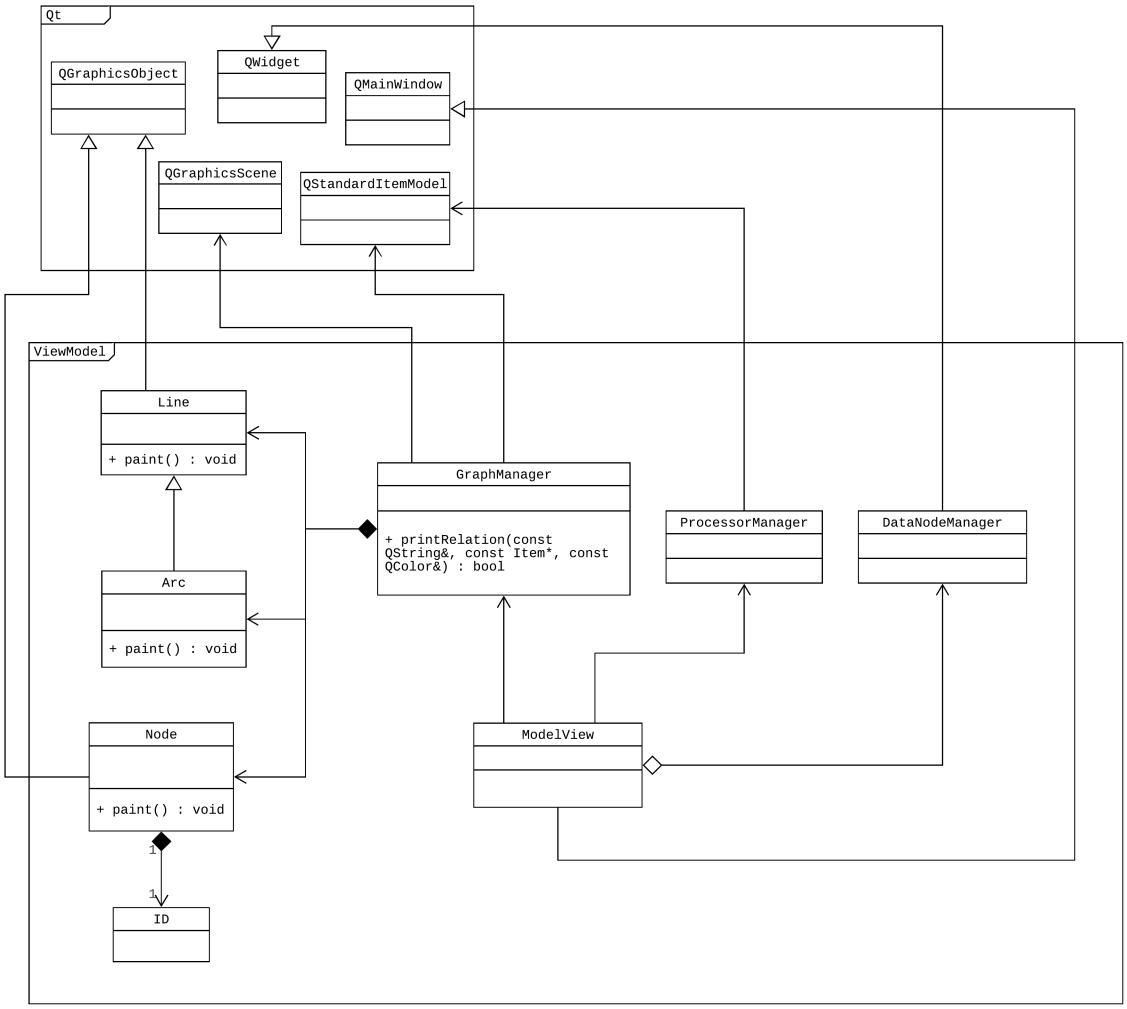

Figura B.2: Diagramma delle classi del componente ViewModel

Si segnala che è obbligatorio aggiungere allo stesso delle *labels*, atte a rendere facilmente reperibili e categorizzabili i ticket aperti. Un ticket privo di labels viene automaticamente rifiutato. Per agevolare la pronta risoluzione di un ticket, si invita a seguire i seguenti accorgimenti:

- Per segnalare un malfunzionamento, comunicare nella descrizione del ticket le seguenti informazioni:
  - Sistema operativo completo di specifiche tecniche;
  - Versione utilizzata di Qt;
  - Versione utilizzata di CMake;

  - Versione utilizzata di GCC.

- Per richiedere una nuova funzionalità è necessario attendere l'approvazione di uno degli amministratori del repository e l'assegnamento del relativo ticket;
- La descrizione del ticket deve essere esaustiva, chiara e rilevante.

Non rispettare uno dei punti precedenti comporta l'automatica cancellazione del ticket.

B.2 Creare una Pull Request

Per configurare il progetto DeSpeect è possibile creare una fork del repository originale, il che consente di lavorare in completa autonomia su una certa funzionalità che si vuole aggiungere all'applicazione. In seguito, se si vuole inserire tale funzionalità nel progetto originale, è necessario creare una *Pull Request* all'interno di GitHub. Una volta creata la Pull Request cliccando sulla relativa tab all'interno del repository e successivamente sul pulsante *Create pull request*, è sufficiente attendere l'approvazione di uno degli amministratori del repository originale per poter eseguire il merge delle modifiche. Di seguito sono riportati i passi da eseguire per creare una Pull Request utilizzando la fork di DeSpeect:

1. Creare la fork di DeSpeect;
2. Recarsi nella sezione *Pull Request* del repository originale;
3. Selezionare *compare across forks*;
4. Su *base fork* e relativo *base* lasciare invariati i valori inseriti (riguardano il branch master del progetto originale);
5. Su *head fork* selezionare la propria fork e di conseguenza sul relativo *head* scegliere il branch dove sono state apportate le modifiche;
6. Infine selezionare *Create pull request* aggiungendovi titolo e commento.

Anche in questo caso valgono gli stessi accorgimenti illustrati in relazione ai ticket, sintetizzati in descrizioni esaustive, chiare e rilevanti.

C. Licenza

Il codice di DeSpeect, fatta eccezione per le librerie esterne quali Speect e Qt, è rilasciato sotto licenza *MIT*. Tale licenza è visualizzabile nella sua interezza al seguente link:

https://choosealicense.com/licenses/mit/

D. Glossario