Product Baseline - Allegato tecnico

Informazioni Documento

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 la *Product Baseline* (PB in breve) per l'applicazione *"DeSpeect: un'interfaccia grafica per Speect"*, con particolare attenzione per lo stato attuale del prodotto, la sua architettura e la copertura di use case e requisiti funzionali obbligatori.

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 Riferimenti

Riferimenti normativi

- **Norme di Progetto v3.0.0**: documento *Norme di progetto v3.0.0*;
  §2.2.5 "Progettazione";
  §4.7.3 "Strumenti relativi allo sviluppo".

Riferimenti informativi

- **Analisi dei Requisiti v3.0.0**: documento *Analisi dei Requisiti v3.0.0*;
  Definisce nel dettaglio use case e requisiti.
- **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 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

3. Installazione ed esecuzione

Il codice relativo alla Product Baseline è reperibile da un apposito repository GitHub al seguente link:

https://github.com/graphiteSWE/Despeect-ProductBaseline

Per installare ed eseguire il software è necessario attenersi alla seguente procedura:

1. Clonare o scaricare la repository sulla propria macchina;
2. Entrare nella cartella scaricata ed eseguire lo script `build.sh`;
3. Eseguire da terminale il comando `cd DeSpeect/build/`;
4. Avviare l'eseguibile con il comando `./main`.

Tale procedura installerà la libreria Speect e genererà una build del software nella directory `DeSpeect/build`, nonché avvierà automaticamente un'esecuzione dello stesso. Ulteriori informazioni sono reperibili nel file `README.md` del repository.

4. Rapporto con il PoC

Precedentemente alla Product Baseline, è stato realizzato un *Proof of Concept* (PoC in breve) a dimostrazione della fattibilità del prodotto DeSpeect. Tale PoC è tuttora reperibile al seguente link:

https://github.com/graphiteSWE/TB-PoC

Segue una tabella che evidenzia le differenze tra le caratteristiche del PoC, contestualizzato nel suo scopo, e quelle della PB.

Tabella 4.1: Tabella di confronto tra PoC e PB

Proof of Concept	**Product Baseline**
Architettura abbozzata e sommaria, priva di design pattern	Architettura basata su MVVM e facente uso di vari design pattern
Implementazione di un'interfaccia grafica provvisoria e carente di elementi fondamentali per il soddisfacimento di molti requisiti obbligatori	Interfaccia grafica quasi completa predisposta all'implementazione della totalità dei requisiti obbligatori
Implementazione di poche funzionalità dimostrative (per esempio la stampa parziale del grafo)	Implementazione della maggior parte delle funzionalità richieste dai requisiti funzionali obbligatori
Modalità di installazione e configurazione macchinose	Modalità di installazione e configurazione semplificate

5. Architettura

5.1 Model-View-ViewModel

5.2 Model

5.2.1 Design pattern

5.2.1.1 Command

5.2.1.2 Builder

5.2.1.3 Adapter

5.3 View

5.4 ViewModel

5.4.0.1 Observer

5.4.1 Design pattern

6. Use case coperti

6.1 Tabella della copertura degli use case

6.2 Grafico della copertura degli use case

7. Requisiti soddisfatti

7.1 Tabella del soddisfacimento dei requisiti

7.2 Grafico del soddisfacimento dei requisiti

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.

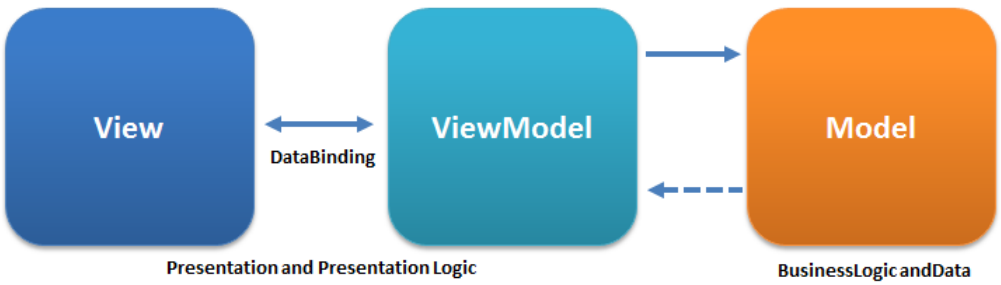

Figura A.1: Diagramma generale dell'architettura MVVM

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. Design Pattern

Vengono di seguito illustrati sinteticamente i design pattern implementati dal prodotto DeSpeect.

B.1 Command

Il Command pattern è un design pattern che permette di isolare la porzione di codice che effettua un'azione (eventualmente molto complessa) dal codice che ne richiede l'esecuzione; l'azione è incapsulata nell'oggetto Command. L'obiettivo è rendere variabile l'azione del client senza però conoscere i dettagli dell'operazione stessa. Altro aspetto importante è che il destinatario della richiesta può non essere deciso staticamente all'atto dell'istanziazione del comando ma ricavato a tempo di esecuzione. Segue il diagramma UML rappresentativo del pattern.

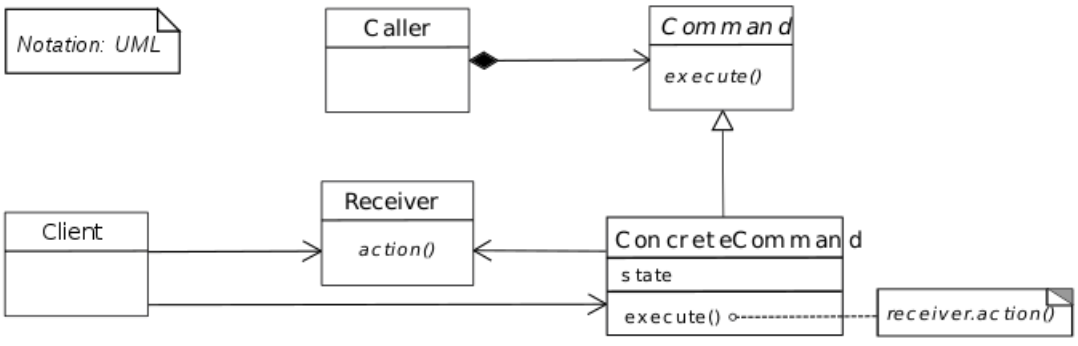

Figura B.1: Diagramma UML del design pattern Command

B.2 Builder

B.3 Adapter

B.4 Observer