Progetto “DataAggregator & Connettori” – TLP 2013

Documentazione tecnica

Stefano Pisciella (Matricola 206785), Guerino Angelozzi (Matricola 196076)

Sommario

PROGETTO “DATAAGGREGATOR & CONNETTORI” – TLP 2013 ..... **1**

DOCUMENTAZIONE TECNICA 1

- **1) CONFIGURAZIONE** **2**
- **2) ARCHITETTURA SOFTWARE** **2**
  - **2.1) DATALAYER** 2
  - **2.2) CONNETTORI** 3
  - **2.3) DATAAGGREGATOR** 3
  - **2.4) GUI** 3
- **3) DESIGN PATTERNS** **3**
  - **3.1) STRATEGY PATTERN** 3
  - **3.2) ABSTRACT FACTORY** 4
    - ***3.2.1) Datalayer*** *4*

1) Configurazione

Per il funzionamento del progetto è richiesta la configurazione di una connessione al database. A tal fine è stata sviluppata una classe apposita per la configurazione dei dati richiesti, *it.univaq.tlp.db.Config*.
Di seguito uno screenshot della classe, con commenti che spiegano il significato di ogni campo:

```
public class Config {
    /**
     * L'indirizzo del server dove è il database
     */
    public static final String hostname = "localhost";
    /**
     * La porta su cui il database è in ascolto
     */
    public static final String port = "3306";
    /**
     * Il nome del database a cui connettersi
     */
    public static final String db_name = "tlp";
    /**
     * Username di un utente con privileggi di lettura e inserimento
     */
    public static final String username = "root";
    /**
     * Password dell'utente sopra specificato
     */
    public static final String password = "";
    /**
     * Tipo di database utilizzato
     */
    public static final String type = "mysql";
```

2) Architettura software

Il progetto rappresenta solo 2 delle componenti di un sistema più grande, per questo è stato pensato di sviluppare un architettura il più possibile flessibile ed adattabile.
Il sistema sviluppato è diviso in 3 componenti principali e una GUI sviluppata solo per il testing dell’applicazione.

- **2.1) Datalayer**
  Dal momento che il progetto poggia su un database, si è scelto di creare un’interfaccia *“Datalayer”* in cui sono esplicitati i metodi richiesti per l’interazione con il database. In questo modo si possono creare varie implementazioni per ogni

tipo di database utilizzato (nel nostro caso abbiamo creato un'unica implementazione per il database MYSQL).
Oltre che rendere indipendente dal database, il datalayer è l’unico con la responsabilità di comunicare con la base di dati permettendoci di separare le logiche. (tipico del pattern architetturale MVC).
(*Vedere allegato “Componente Datalayer - Class Diagram.jpg”).*

- **2.2) Connettori**
  I connettori sono i responsabili della comunicazione con le API dei social network utilizzati. Tutti i connettori implementano l’interfaccia *“Connector”* che espone i metodi richiesti dal DataAggregator. L’interfaccia ci permette di rendere indipendente le implementazioni dalle altre componenti (purchè rispettino l’interfaccia).
  I connettori quindi dovranno comunicare con le API dei social network utilizzate (in questo caso Twitter4j e Facebook4j) e per renderli indipendenti dalle API è stata creata un interfaccia Adapter che espone le funzionalità richieste dai connettori, le implementazioni poi si occuperanno della comunicazione diretta.
  (*Vedere allegato “Componente Connettori - Class Diagram.jpg”).*

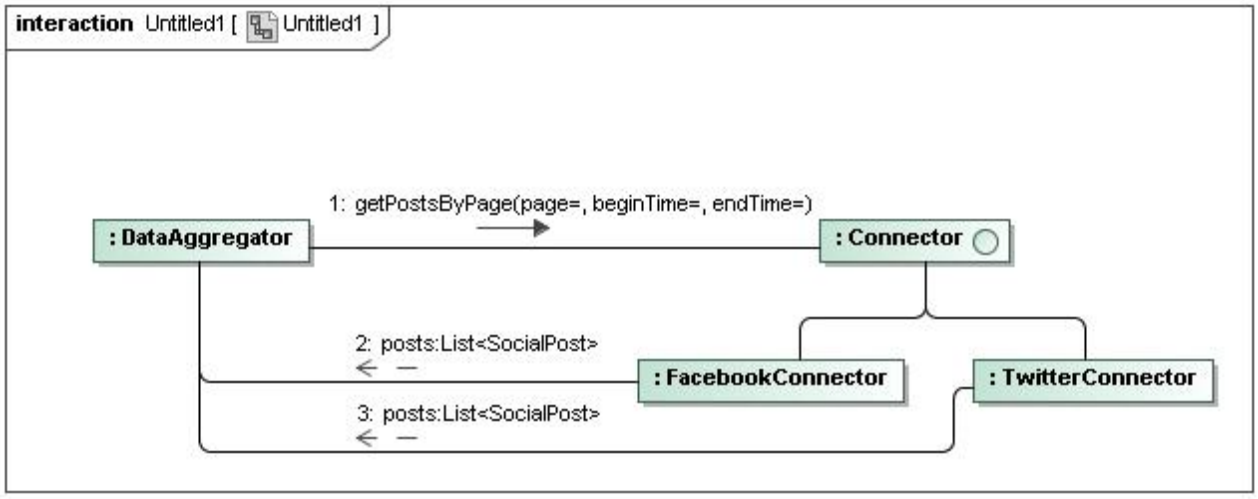
- **2.3) DataAggregator**
  Il DataAggregator è la componente principale, comunica con il datalayer e i connettori e si occuperà della “traduzione” dell’outuput dei connettori a input del datalayer.
  (*Vedere allegato “Componente DataAggregator - Class Diagram.jpg”).*
- **2.4) Gui**
  La gui è stata principalmente sviluppata per il testing dell’applicazione. E’ composta da un input per scegliere la data per la ricerca, 2 bottoni per l’avvio e per lo stop del DataAggregator e una textarea dove viene redirectato il *System.out* per far stampare errori e azioni del sistema.

3) Design Patterns

Abbiamo sfruttato alcuni design patterns sempre per rendere il progetto adattabile in ogni sua componente.

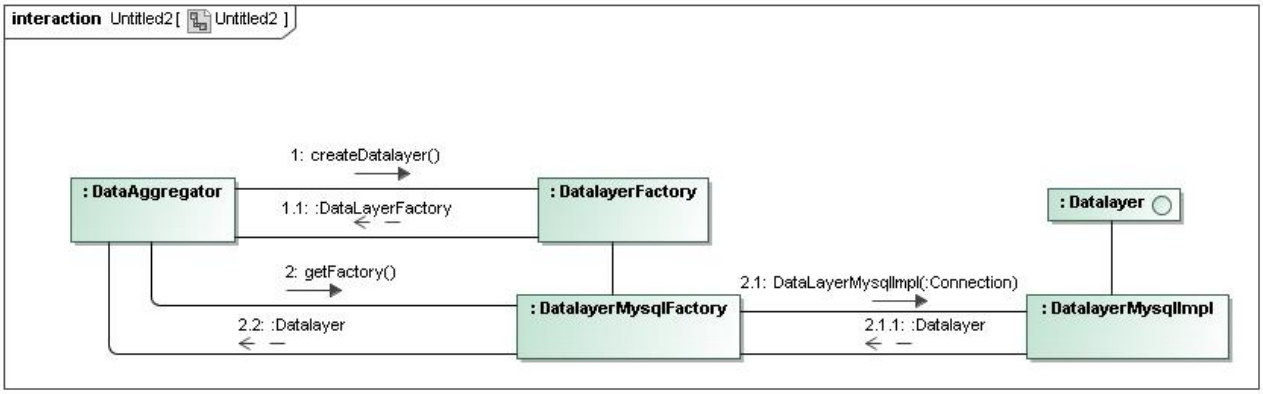
- **3.1) Strategy pattern**
  Abbiamo utilizzato questo design pattern nello scambio di messaggi tra il DataAggregator e i Connettori.
  Il data aggregator si compone di una mappa <String,Connector>, che collega il type (“facebook”, “twitter”) al relativo Connettore.
  Il client quindi cicla la mappa e invoca il *“getPostByPage(-,-,-)”* sull’interfaccia

Connector, a runtime però sarà chiamata sulla giusta implementazione del *Connector.*



- **3.2) Abstract Factory**
  Abbiamo sfruttato questo design factory in due occasioni: per astrarci dal tipo di database utilizzato e dall’adapter utilizzato per le API(*vedi 3.3)*.

  - **3.2.1) Datalayer**
    E’ stato pensato di utilizzare un factory per la creazione del datalayer apposito in base al type specificato nelle configurazione del database (es. mysql, oracle, etc..).
    All’interno del metodo *getFactory()* della classe astratta *DatalayerFactory* è presente una mappa <String, DatalayerFactory> che collega il type (specificato nel config) al giusto factory. (Per il progetto è stata sviluppato solo l’implementazione mysql del datalayer).
    L’estensione del *DatalayerFactory* restituirà poi, tramite il metodo *createDatalayer()*, la giusta implementazione del Datalayer (nel nostro caso DatlayerMysqlImpl).



- **3.2.2) Adapter**
  Utilizziamo il factory anche per la creazione della giusta implementazione dell’interfaccia adapter *(vedi 3.3)*.

- **3.3) Adapter**

Si è sfruttata l’idea del pattern Adapter, e non precisamente la struttura, per realizzare un interfaccia che ci permetta di astrarci dalle API per il collegamento diretto con i SocialNetwork. Per ogni social network è stata creata un interfaccia adapter che, anche se espongono gli stessi metodi, sono state divise sempre per il discorso di separare le logiche il più possibile. L’interfaccia adapter espone i metodi che poi le varie implementazioni si occuperanno di definire per le API corrispondenti.
In questo modo, in caso di cambio delle API utilizzate, sarà richiesto solo una nuova implementazione dell’interfaccia adapter relativa.