Relazione progetto Orario Università

Lorenzo Cottignoli

31/05/2015

Sommario

1. Analisi

1.1 Requisiti

Il software, creato su consiglio del professor Viroli Mirko, mira a costruire un gestionale per l'orario universitario annuale della facoltà di scienze e tecnologie informatiche del polo di Cesena.
Le funzionalità che questo software dovrà gestire sono:

- inserimento e rimozione di una materia in una lista per la memorizzazione delle varie materie;
- possibilità di importare ed esportare la suddetta lista;
- inserimento e rimozione di una materia nell'orario;
- possibilità di salvare e ricaricare il proprio orario;
- possibilità di avere diverse viste dell'orario per aiutare l'utente nella leggibilità (orario totale, per professore, materia, ecc.);
- possibilità di utilizzare i comandi undo e redo, cioè ripristinare l'ultima operazione effettuata.

1.2 Problema

Ogni orario universitario è composto da due semestri. Ogni semestre è possibile definirlo tramite l'orario di una settimana. Ogni orario settimanale è formato da più orari giornalieri (che corrispondono al numero di giorni in cui l'università è aperta), che a sua volta è formato dall'insieme dell'orario giornaliero di ogni singola aula. Un orario giornaliero sarà composto da varie ore (durata di una giornata universitaria), occupate, nel caso vi sia una lezione, oppure libere, se non vi è stata inserita ancora alcuna materia.
Ogni materia è definita tramite il nome del corso, il nome del professore responsabile del corso e dalla tipologia della materia (primo anno della triennale, primo anno della magistrale, opzionale, ecc.)
Inoltre bisognerà tener di alcune specifiche:

- Una materia non potrà essere inserita in un'ora già occupata da un'altra materia;
- Una materia può essere inserita allo stesso orario in due aulee differenti;
- Un professore non può insegnare contemporaneamente in due aulee e due materie differenti.

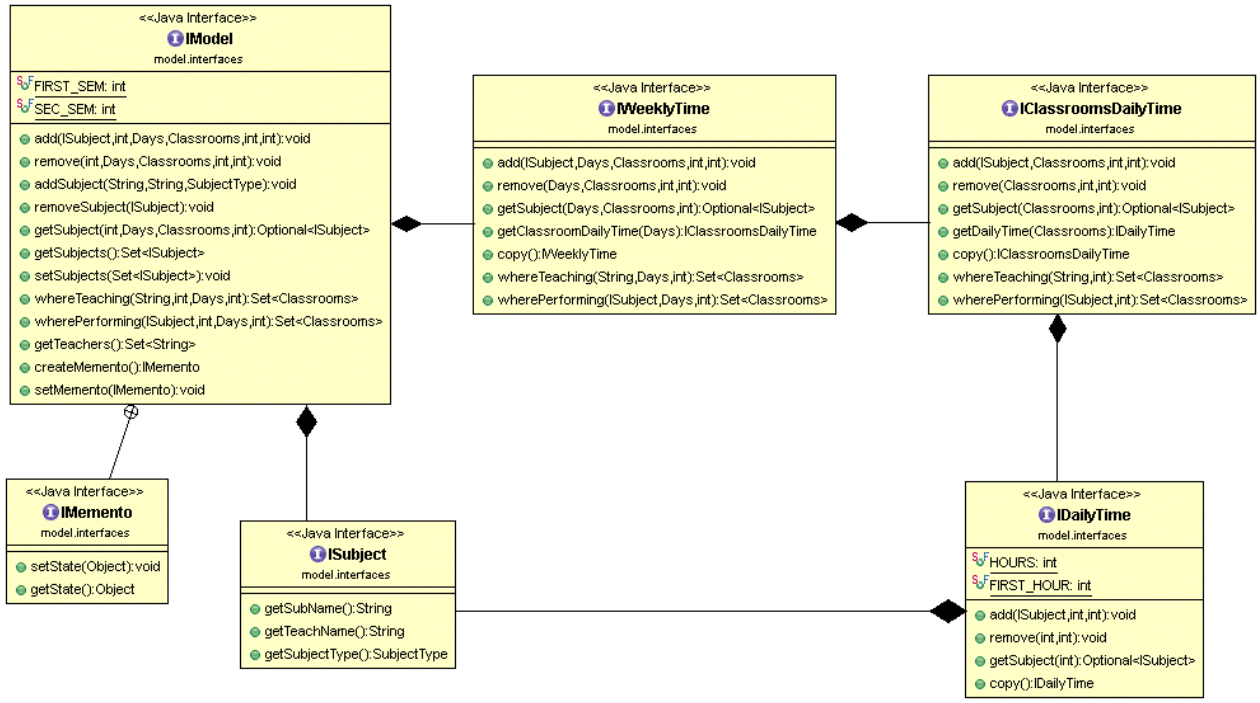


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro.

2. Design

2.1 Architettura

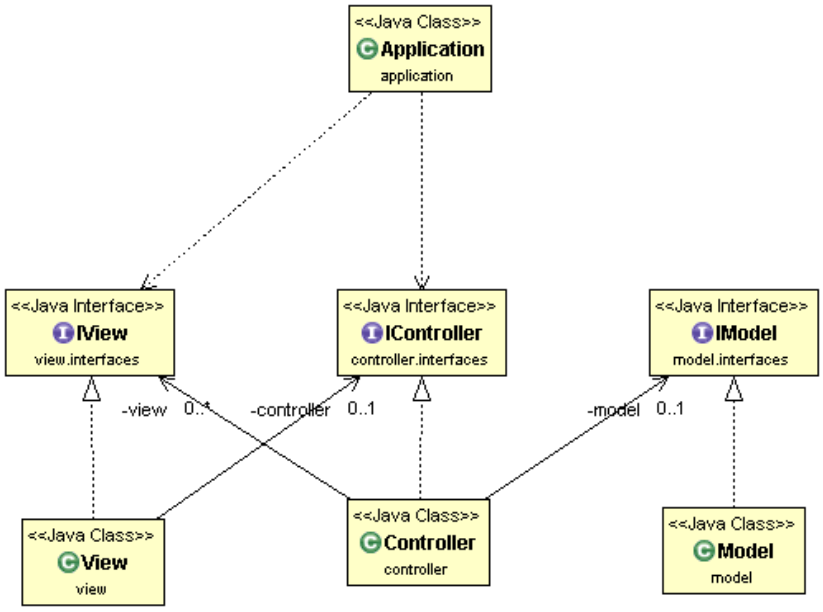


Figura 2.1: In questo schema UML viene rappresentato come è stato implementato il pattern architetturale MVC nel progetto. In questo modo si possono modificare piccole parti di un'entità oppure le implementazioni stesse (ovviamente rispettando le interfacce che estendono) senza dover modificare l'intero software.

2.2 Design dettagliato

La suddivisione delle classi nei package seguono questo schema:

- application: contiene solamente la classe per far partire l'intero software;
- model.interfaces: contiene le interfacce di figura 1.1, cioè delle classi che definiscono il modello del progetto;
- model: contiene tutte le classi che implementano le interfacce presenti nel package "model.interfaces" e alcuni classi enumerator (Days e Classrooms);
- controller.interfaces: contiene tutte le interfacce relative al controller;
- controller: contiene tutte le implementazioni delle interfacce presenti nel package "controller.interfaces" e una classe di utilità per il salvataggio/caricamento da file;
- view.interfaces: contiene le interfacce relative alla view;
- view: contiene le implementazioni delle interfacce presenti nel package "view.interfaces" e le "ridefinizioni" di alcune classi della libreria swing per creare un'interfaccia user-friendly;
- test: contiene classi di testing su alcune classi che compongono il model.

Pattern utilizzati:

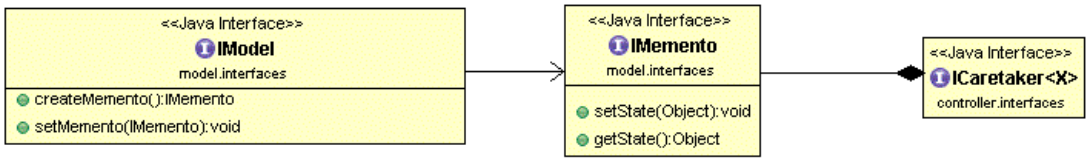


Figura 2.2: Schema UML in cui vi è rappresentato il pattern memento utilizzato nel progetto (Ho omesso in questo schema i metodi non fondamentali, di queste interfacce, per la descrizione del pattern in questione). Questo pattern viene utilizzato nel progetto per realizzare i comandi di undo/redo. Il model può creare, ogni qual volta verrà richiamato il metodo createMemento(), un oggetto memento che conterrà lo stato del modello in quell'istante. Inoltre può effettuare l'operazione duale, ovvero ripristinare uno stato passato del modello tramite l'operazione setMemento(IMemento) passandogli l'oggetto memento istanziato precedentemente.

```
private List<Object> viewFunct(final int sem, final MyFunction<?> fun) {
    INTEST.set(0, "Days");
    final List<Object> list = new ArrayList<>(INTEST);
    for (final Days d : Days.values()) {
        list.add(d.getName());
        for (int i = IDailyTime.FIRST_HOUR; i < (IDailyTime.FIRST_HOUR + IDailyTime.HOURS); i++) {
            list.add(fun.apply(sem, d, i));
        }
    }
    return list;
}
```

figura 2.3: In questa figura è mostrato un metodo privato della classe "ViewController" del package "controller". Questo metodo aderisce al pattern strategy, in quanto alla lista di ritorno viene aggiunto l'oggetto restituito dalla MyFuction fun quindi la "strategia" di scegliere che oggetto aggiungere spetta ad all'oggetto "fun" passato come parametro.
Questo metodo viene utilizzato quando il controller aggiorna la view. Dovendo permettere visualizzazioni differenti del modello (vista per docente, materia, ecc.) si può notare che alcune di queste tipologie presentano la stessa struttura e si differenziano soltanto per la scelta dell'oggetto da mostrare (es. nel caso della vista per docente verrà mostrato "nome della materia che insegna + nome dell'aula", mentre nella vista per aula verranno mostrati tutti i campi relativi ad una materia).
MyFunction è un'interfaccia funzionale creata da me per risolvere questo problema.

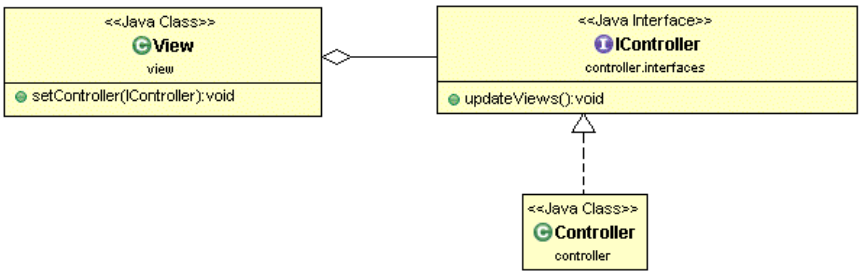


Figura 2.2: Schema UML in cui vi è rappresentato il pattern observer utilizzato nel progetto (Ho omesso in questo schema i metodi non fondamentali, di queste entità, per la descrizione del pattern in questione). In questo schema possiamo notare che la view è il soggetto da osservare mentre l'IController corrisponde ad all'observe/listener. In questo modo il controller può catturare tutti gli eventi della view e aggiornarla di conseguenza.

3 Sviluppo

3.1 Testing automatizzato

All'interno del progetto ho utilizzato il testing automatico per controllare che il funzionamento di alcune classi fosse corretto. Le classi sottoposte a testing automatico sono quelle che gestiscono la maggior parte del lavoro del modello. Per quanto riguarda il controller e la view non sono stati usati test automatici ma solamente controlli manuali, in quanto risultavano più semplici e rapidi per controllare il corretto funzionamento.
Per il testing automatico ho utilizzato la suite di JUnit. Grazie a questo strumento per il testing ho potuto ridurre di molto la parte di debugging relativa al model.

3.2 Divisione dei compiti e metodologia di lavoro

Avendo fatto il progetto da solo non vi è stato il problema della suddivisione dei compiti.
Il DVCS che ho utilizzato è Mercurial. Essendo da solo nella creazione di questo progetto non era fondamentale l'utilizzo di Mercurial e BitBucket, ma ho voluto sfruttare questo servizio lo stesso in quanto mi potrà essere utile in futuro conoscere discretamente il suo funzionamento.

4 Commenti finali

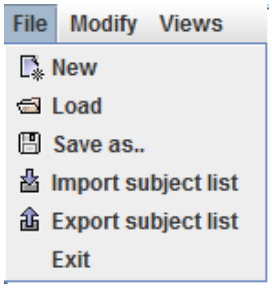
4.1 Conclusioni e lavori futuri

Sono soddisfatto del mio progetto e credo sia ben svolto. Ho cercato di inserirci la maggior parte degli argomenti trattati a lezione (alcuni pattern, interfaccie funzionali, interfacce generiche, testing, ecc). Ha un'interfaccia utente snella e di facile comprensione, in modo da facilitarne l'utilizzo. Alcuni aspetti potrebbero essere migliorati, esempio l'unione di celle adiacenti con lo stesso contenuto nella tabella per rendere il tutto più leggibile oppure aggiungere una materia nell'orario cliccando direttamente su una cella della tabella, recuperando così automaticamente i dati necessari per l'inserimento nel modello dalla tabella. Sfortunatamente, essendo stato da solo ed avendo quasi raggiunto il massimo delle ore disponibili per lo sviluppo del progetto, non ho potuto implementare i salvataggi/caricamenti vari da file con estensione .xls (funzionalità che avevo specificato nella proposta di progetto che avrei sviluppato solo se avessi rimasto altro tempo) ma ho semplicemente gestito i vari dati (essendo serializable) come visto a lezione. Se il professor Viroli, essendo colui che mi ha "commisionato" il sofware, sarà soddisfatto di questo

lavoro e vorrà utilizzarlo, potrei implementare questa funzionalità successivamente. In caso contrario non prevedo migliorie o upgrade per il futuro. In conclusione posso affermare che, come primo progetto basato sul paradigma ad oggetti, questo software è svolto correttamente.

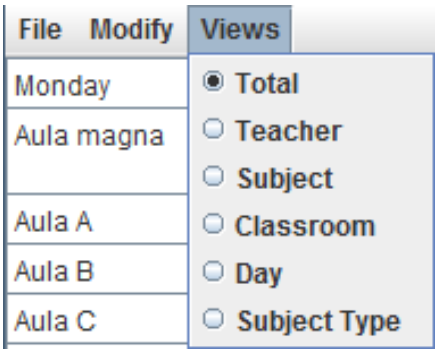
Appendice

Guida utente

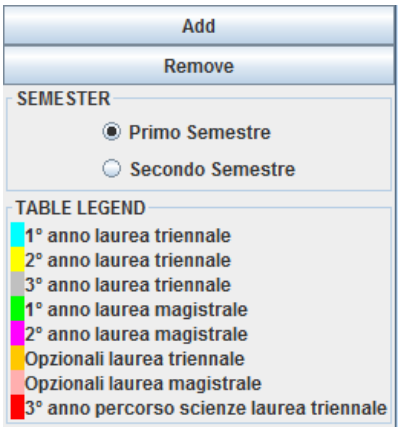


New: Creazione di un nuovo orario e permette di salvare il file corrente.
Load: Caricamento dell'orario da file.
Save as: Salvataggio dell'orario corrente su un file (compreso di lista delle materie disponibili).
Import subject list: Caricamento della lista delle materie disponibili da file.
Export subject list: Salvataggio della lista delle materie disponibili su un file.
Exit: Chiusura del programma e permette di salvare il file corrente prima di uscire.

Undo: Annullamento dell'ultima modifica effettuata sull'orario (è possibile annullare al massimo le ultime 9 modifiche).
Redo: Ripristino di una modifica precedentemente annullata.
Add new subject to list: Permette di aggiungere una nuova materia all'elenco delle materie disponibili (cioè le materie inseribili nell'orario).
Remove subject from list: Permette di rimuovere una materia dall'elenco delle materie disponibili.



Total: Visualizzazione completa dell'orario del semestre selezionato.
Teacher: Visualizzazione dell'orario del professore scelto nel semestre selezionato.
Subject: Visualizzazione dell'orario della materia scelta nel semestre selezionato.
Classroom: Visualizzazione dell'orario dell'aula scelta nel semestre selezionato.
Day: Visualizzazione completa dell'orario del giorno scelto nel semestre selezionato.
Subject Type: Visualizzazione dell'orario della tipologia di materia scelta nel semestre selezionato.



Add: Inserimento di una delle materie presenti nella lista delle materie disponibili nell'orario.
Remove: Rimozione di una o più materie nell'orario.
First Semester: Permette di lavorare sull'orario del primo semestre se selezionato.
Second Semester: Permette di lavorare sull'orario del secondo semestre se selezionato.
Table legend: Serve per aiutare a capire all'utente a che tipologia di materia corrispondono i vari colori delle celle della tabella.