Relazione per il progetto di "Programmazione ad Oggetti"

Domenico Rivoli

01 marzo 2016

Indice

1 Analisi **2**
1.1 Requisiti 2
1.2 Analisi e modello del dominio 3

2 Design 4
2.1 Architettura 4
2.2 Design dettagliato 6

3 Sviluppo **10**
3.1 Testing automatizzato 10
3.2 Metodologia di lavoro 10

4 Commenti finali 11
4.1 Autovalutazione e lavori futuri 11
4.2 Difficoltà incontrate e commenti per i docenti 11

A Guida utente **12**

Capitolo 1

Analisi

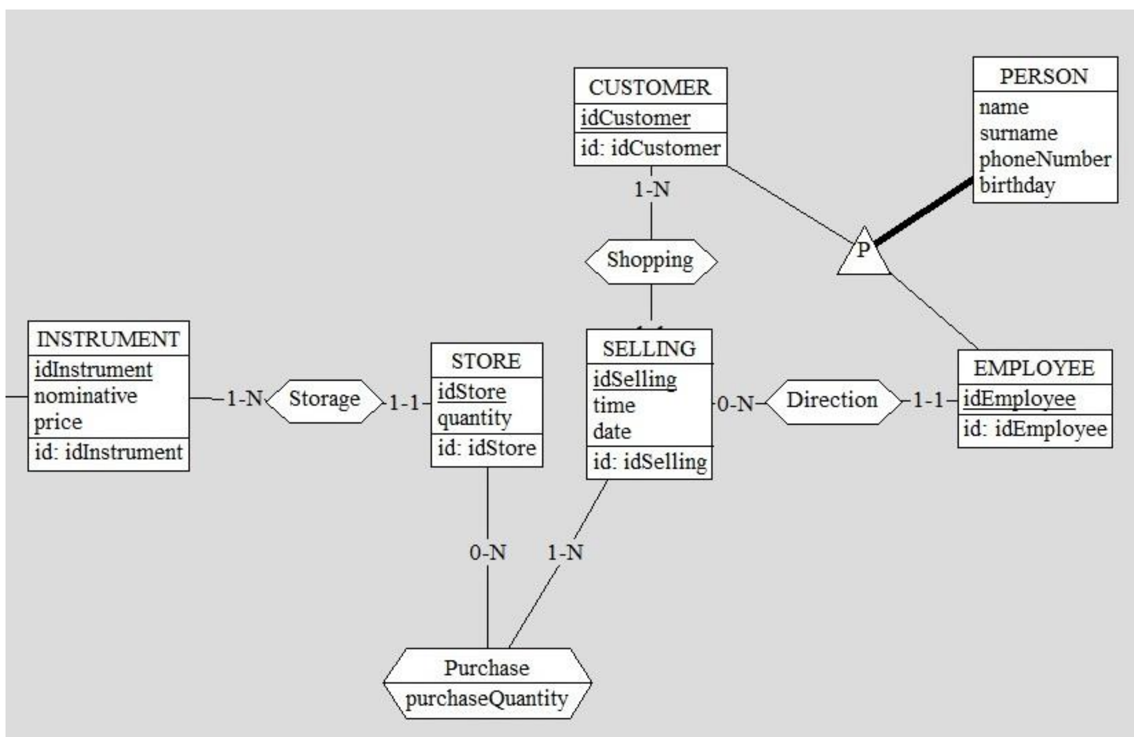
Il software realizzato, MusicArt-StoreManager, commissionato dal negozio di strumenti musicali MusicArt, prevede l'amministrazione delle entrate e delle uscite dello shop, la dirigenza dei turni dei dipendenti e la regolamentazione del rapporto con il cliente.

1.1 Requisiti

- Il software mette a disposizione due aree di interesse per l'utente utilizzatore. La prima consente di modificare l'archivio nel quale vengono depositate le informazioni relative alla merce, ai dipendenti, ai clienti, ai fornitori e ai produttori. La seconda area dà la possibilità all'utente di visionare le attività del negozio, come ad esempio la tracciatura delle vendite avvenute.
- MusicArt-StoreManager permette la registrazione degli strumenti, in entrata e in uscita. Gli strumenti in entrata rappresenteranno una semplice entrata della merce all'interno del magazzino del negozio, mentre, per quanto riguarda l'uscita, si terrà traccia della vendita avvenuta e quindi, della consegna al cliente della merce.
- Viene messa inoltre, a disposizione dell'utente la possibilità di registrare nuovi produttori e fornitori che saranno quindi messi in contatto con il negozio. E' inoltre consentito l'aggiunta di clienti nell'archivio e nuovi dipendenti.
- Il software fornisce una panoramica delle attività del negozio, riportando informazioni circa ogni transazione, servizio e operazione messa in pratica durante il regolare svolgimento dell'attività.

1.2 Analisi e modello del dominio

Il software dovrà essere in grado di analizzare quelle che sono le risorse dello shop e di coadiuvare le transazione in funzione di esse. Le entità principali del software sono: gli strumenti musicali, i produttori, i fornitori, le vendite, i clienti e gli impiegati. Il problema maggiore è riscontrabile nell'analisi e nella gestione della base di dati in cui vengono inseriti gli elementi del negozio. Il software deve essere in grado di raggiungere adeguatamente le risorse e aggiornarle opportunamente. In particolare si vuole trovare una strategia per poter dirigere i turni dei dipendenti. A tale scopo si desidera mettere in relazione l'effettivo orario corrente con la presa al servizio di ciascun impiegato, in modo tale che le vendite vengano gestite opportunamente, e il programma sia in grado di registrare una vendita solo attraverso i dipendenti che sono correntemente al lavoro, senza che l'utente debba intervenire in alcun modo. Qui di seguito mostro un primo schema scheletro delle entità che verranno sviluppate.



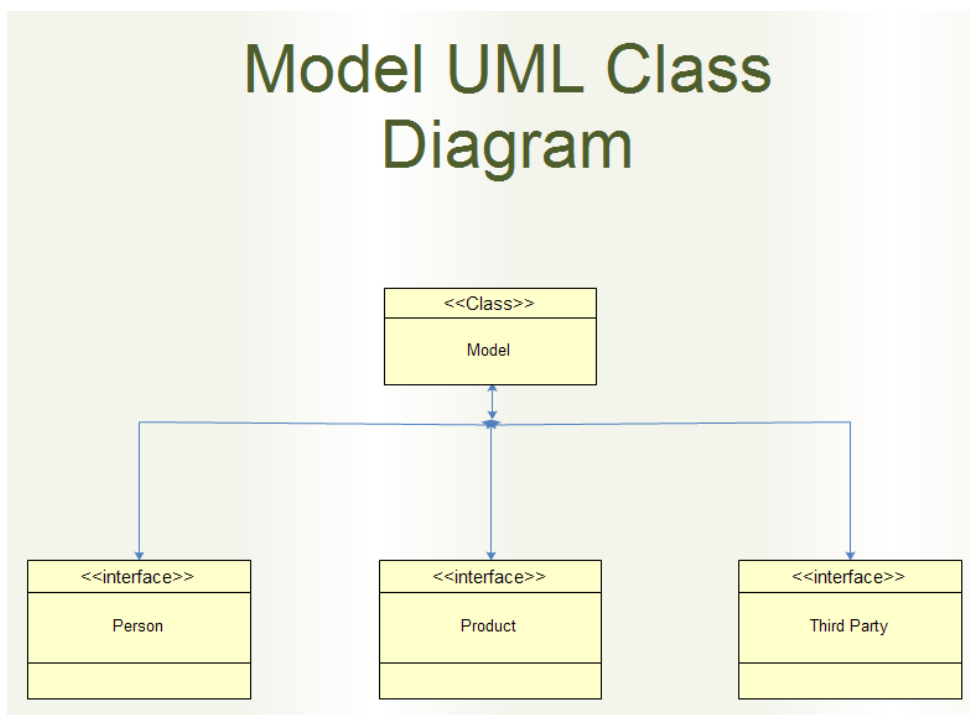
Capitolo 2

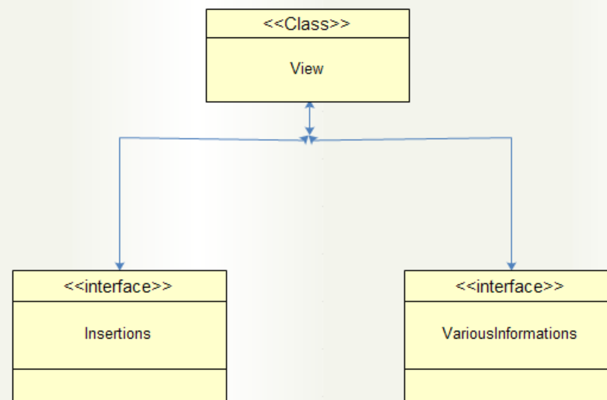
Design

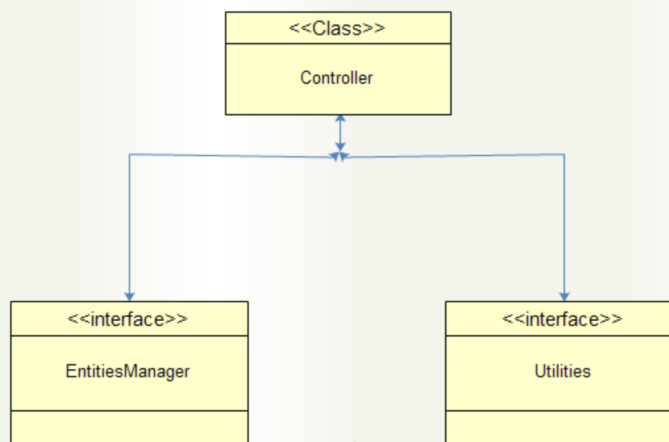
Le strategie messe in campo per ovviare ai problemi mi hanno portato a realizzare un'architettura di tipo MVC, con alcune variazioni che ho apportato al pattern, affinché la mia idea di progettazione potesse essere efficace.

2.1 Architettura

Dal punto di vista architetturale mi sono appoggiato al pattern di progettazione MVC. L'idea principale era quello di realizzare una classe per ogni entità che compone il problema, ma per le scelte di progettazione prese in corso d'opera risultava conveniente procedere con un modello semplificato dell'architettura. Ho preferito delegare il flusso di esecuzione alle view stesse, delegando alla main view l'onere di "ospitare" il main-method che lancia l'applicazione. Ogni view è stata predisposta al collegamento con il relativo controller al fine di consentire un adeguato funzionamento del software. Qui di seguito presento alcuni schemi UML che riassumano la logica di progettazione del software.




View UML Class Diagram
<<Class>>
View
<<interface>>
Insertions
<<interface>>
VariousInformations


Controller UML Class Diagram
<<Class>>
Controller
<<interface>>
EntitiesManager
<<interface>>
Utilities

2.2 Design dettagliato

Come accennato precedentemente, in fase di progettazione ho optato per non appoggiarmi ad uno schema in cui ciascuna entità del programma fosse gestita mediante una classe a sé stante, ma gestire il flusso di esecuzione e gli inserimenti attraverso le view, coordinando il tutto mediante degli opportuni controller richiamati all'occorrenza. Ho comunque preferito predisporre il modello e lasciarlo inalterato, con le classi e le interfacce che avevo ideato inizialmente, in modo da favorirne un eventuale riuso, estensione o perfezionamento del software. Tuttavia le varie registrazioni sono tutte quante gestite al momento, memorizzando su file le varie risorse. Ho preferito quindi non utilizzare le classi membro che avevo inizialmente realizzato , per facilitare maggiormente la correlazione tra il software e la sua interazione con i file. Ho quindi deciso di orientarmi verso uno scenario di questo tipo:

1. L'utente inserisce i dati che vuole registrare in locale
2. I dati inseriti sono controllati mediante classi controller che predispongono dei metodi per l'analisi degli inserimenti
3. I dati, inseriti e certificati, sono direttamente inseriti all'interno del file di destinazione.
4. Quando l'utente vuole visionare le informative relative alle entità del negozio, si reca nella sezione relative alle stesse e la lettura viene fatta direttamente dal file sul quale sono collocati i dati inseriti.

Aspetti strutturali Generali

La suddivisione delle classi è stata ripartita in 2 marco-package. Nel primo ho inserito tutto ciò che riguarda la parte grafica, mentre nel secondo ho inserito due ulteriori package, uno che rappresenta il Model e l'altro che rappresenta il Controller.

2.2.1 Model

Nel Model ho riportato le classi necessarie all'identificazione di ciascuna entità. Il modello è inserito all'interno del package type.classes. Al suo interno vi sono una classe astratta e una interfaccia per settorializzare al meglio le entità, e per conferire estensibilità al codice, qualora si volesse migliorare in futuro. Ciascuna classe del modello riporta gli attributi di ciascuna entità con relativi getter. Ho inserito inoltre una classe contenente una enum innestata, rappresentante gli errori che si possono ottenere in fase di inserimento. Ciascuna enum è quindi gestita tramite un metodo che collega a ciascuna variabile il messaggio, che verrà poi recapitato all'utente con una message-dialog. Ai fini di un'evoluzione futura ho quindi realizzato un'ulteriore classe che ospita le enum relative ai tipi e alle entità del software realizzato, per poter facilitare la scrittura di codice relativo ad esse.

2.2.2 View

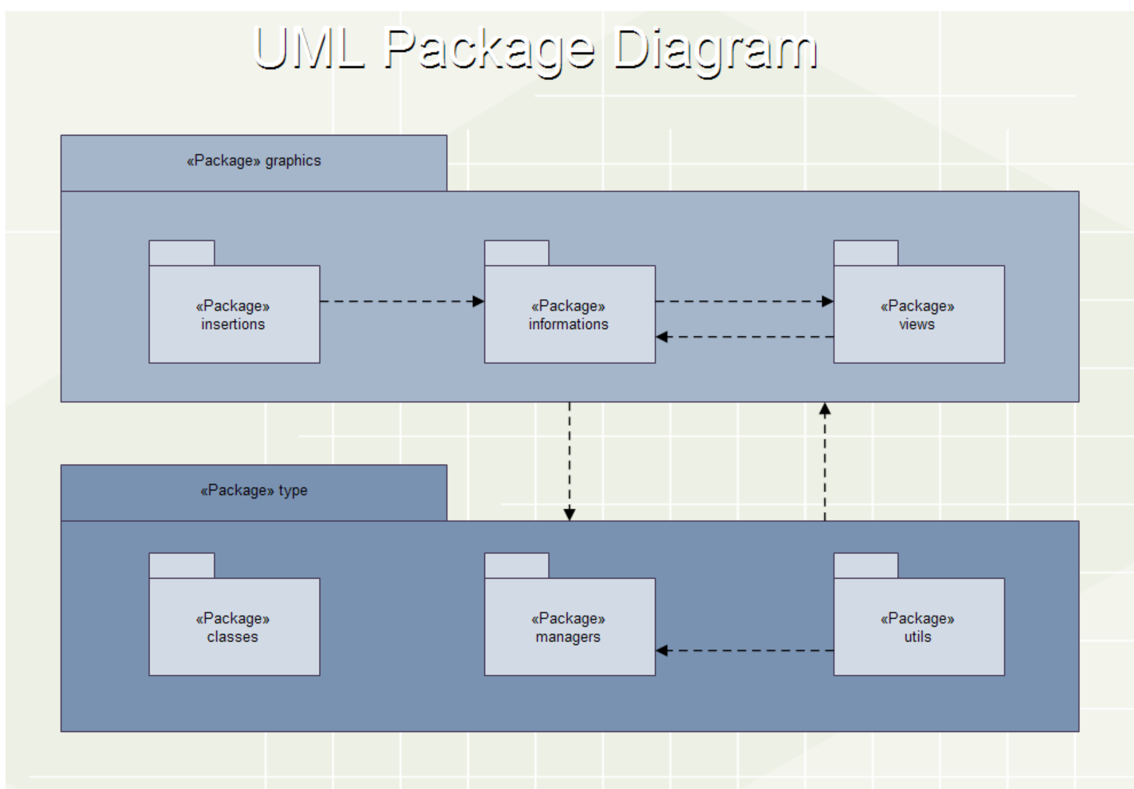
Per quanto riguarda le view ho optato per delegare il main alla view principale del software, ovvero quella che dà il welcome e consente all'utente di raggiungere entrambe le aree di interesse per l'utilizzo del programma. Ho pensato non fosse necessario realizzare una classe a parte che gestisca solo il main, in quanto avrebbe richiamato semplicemente la view principale. Questo poiché il flusso di esecuzione è ripartito fra le varie view del software. Ho quindi realizzato un macro-package con 3 sotto-package. Il primo sotto-package, (graphics.insertion) è quello relativo agli inserimenti, in cui ho realizzato una classe per ciascun tipo di inserimento. Al suo interno sono inoltre presenti due classi: il compito della prima, FrmChooseData, è quello di consentire all'utente di scegliere quale tipo di inserimento vuole effettuare, e in base alla scelta, l'utente viene rimandato alla view di interesse. Ho inserito questa view, se pur la funzionalità messa a disposizione è attualmente una sola, per consentire eventualmente di inserire in futuro nuove informazioni da mostrare, come ad esempio le volte in cui un utente ha effettuato acquisti nel negozio oppure l'ammontare ricavato da ciascun dipendente durante le vendite. La seconda view, FrmChooseEmployee, consente invece all'utente di scegliere quale dipendente vuole eventualmente sostituire, se al team si sta aggiungendo un nuovo impiegato. Nel package relativo alle views (graphics.information.views) sono collocate tutte le view che consentono all'utente di visualizzare le attività del negozio. Per la registrazione delle informazioni anagrafiche, come per esempio per la data di nascita, ho sfruttato una libreria esterna, ovvero JCalendar-1.4, che mi ha quindi consentito di sfruttare un widget che facilitasse l'inserimento della data di nascita dei clienti e dei dipendenti.

2.2.3 Controller

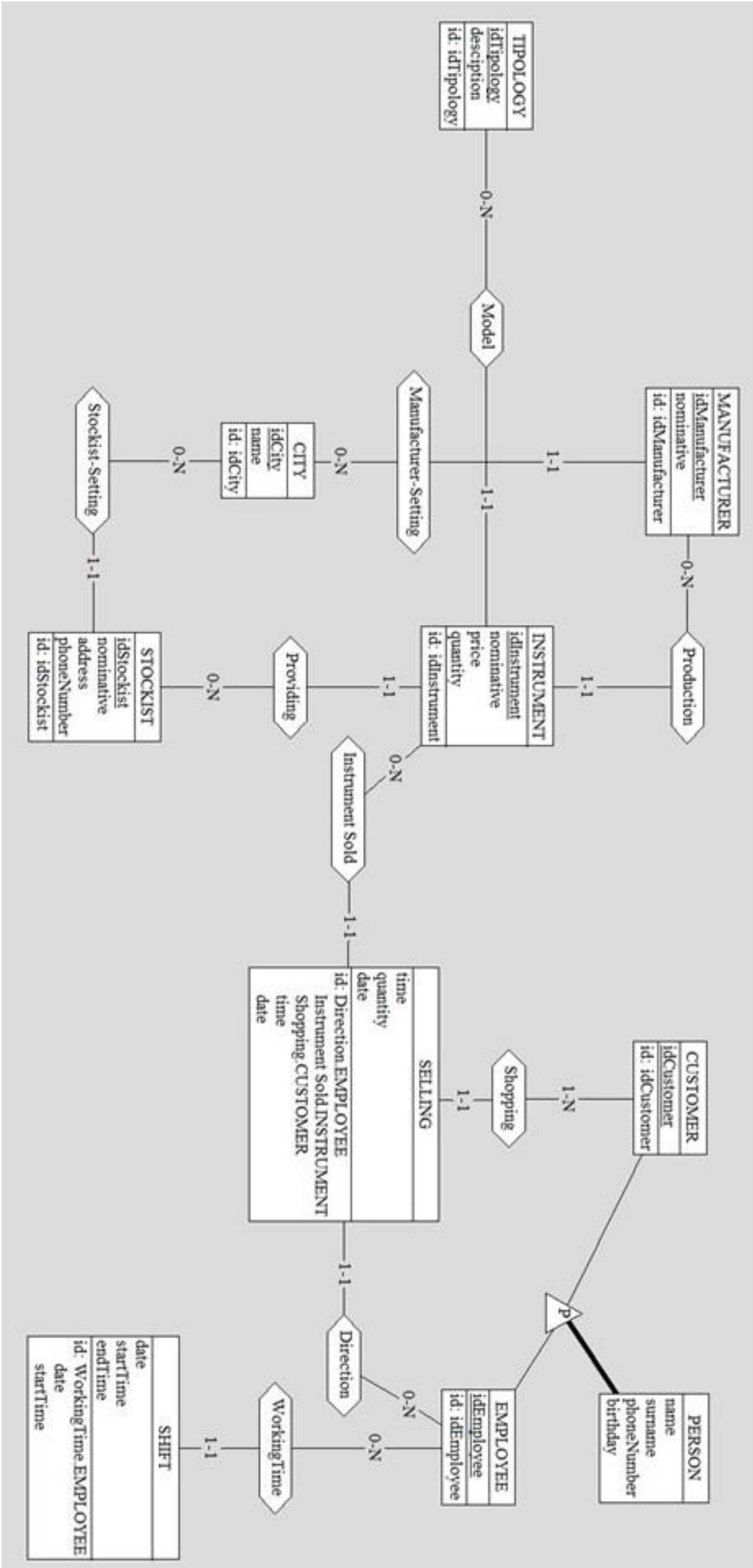
La parte relativa al controllo della corretta esecuzione del programma è stata delimitata all'interno di un package di nome type.managers. In tale package ho predisposto una classe per ciascun elemento inseribile all'interno dell'archivio. Queste classi mettono a disposizione quindi dei metodi che si occupino di analizzare gli inserimenti dell'utente, coordinando l'output in funzione degli input ottenuti dalla digitazione delle nuove risorse. All'interno del package type.managers vi è un sotto-package di nome type.managers.utils che predispone al suo interno 3 classi: la prima, FilesManagement si occupa di gestire le entrate e le uscite in archivio, gestito tutto mediante file di testo, la seconda, ShiftsManagement, che coordina i turni dei dipendenti e la terza, UsersManagement che controlla opportuni inserimenti della clientela in archivio.

2.2.4 Diagramma dei Package

Qui di seguito espongo una rappresentazione di come ho voluto organizzare i package in modo da favorire la parte strutturale del progetto



Schema concettuale



Capitolo 3

Sviluppo

3.1 Testing

Per la parte di testing ho pensato di effettuare tutte le prove manualmente, valutando gli output risultanti dall'attività più o meno corretta da parte dell'utente. Dopo alcune valutazioni ho pensato che la realizzazione di un test automatizzato avrebbe portato via molto più tempo del previsto. Sarebbe stato ottimo delegare la parte di testing ad un altro operatore, come succede abitualmente per i progetti software importanti, che si dedicasse unicamente a quella parte del software. La scelta di lavorare da solo ad un progetto porta purtroppo a scegliere a delle scelte di progettazione che siano favorevoli ad uno sviluppo il più rapido possibile del progetto, che talvolta non sono sempre le migliori.

3.2 Metodologia di lavoro

Durante tutto il corso dello sviluppo non ho avuto la necessità di utilizzare alcun version control distribuito, lavorando da solo al progetto. Il lavoro è stato portato unicamente in locale e aiutandomi di tanto in tanto con la history e facendo alcune copie di backup dei singoli file prodotti. Nella risoluzione di problemi che insorgevano durante lo sviluppo ho fatto affidamento ad alcuni testi e manuali per Java e testando le varie sotto-funzionalità esternamente al software, in modo da isolare il più possibile eventuali problemi e poi integrandoli nel progetto

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

A lavoro finito posso riconoscere che se pur il software soddisfa i requisiti, si potrebbero applicare in futuro delle migliorie. Le migliorie che personalmente applicherei è curare maggiormente l'aspetto OO. Cercherei di studiare meglio la correlazione fra le classi-membro realizzate, le quali sono già state predisposte e pronte per un'eventuale integrazione del software. Cercherei quindi di migliorare il progetto dal punto di vista organizzativo, cercando di migliorare il più possibile la suddivisione dei package, migliorando l'isolamento dei vari aspetti MVC.

4.2 Difficoltà incontrate e commenti per i docenti

Nello sviluppo del software ho riscontrato non poche difficoltà, non avendo modo di confrontarmi con chi poteva essere un compagno nello sviluppo di questo software, che sicuramente avrebbe avuto un impatto positivo sul lavoro. Al termine di questo lavoro posso dire con consapevolezza che lavorare da soli ad un progetto software è sicuramente uno svantaggio, soprattutto per uno studente alle prime armi con l'aspetto vero e proprio dello sviluppo software, che non è semplicemente smanettare con un linguaggio di programmazione, ma è bensì una vera e propria pratica per realizzare un prodotto buono, dal punto di vista implementativo e dal punto di vista dell'esperienza utente. Può essere un vantaggio magari se si hanno le idee molto chiare su come si vuole progettare il software, soprattutto dal punto di vista architetturale, gestendo il tutto a proprio piacimento. Prima di iniziare il progetto ho pensato che lavorando da solo sarebbe potuto essere un vantaggio per quanto riguardava l'aspetto organizzativo del tutto, ma lo sforzo per cercare di capire qual è la direzione giusta da seguire durante la realizzazione del lavoro non è sicuramente di poco conto. Esco da questa esperienza sicuramente più fortificato rispetto a prima.

Lavorare da solo ti porta a cercare e trovare una soluzione a tutti i costi, nonostante le difficoltà. Ma credo che in futuro lavorare in gruppo renderà le cose sicuramente migliori e meno frustranti sotto alcuni aspetti. Sono più che consapevole che il software sia ben lontano dalla perfezione, ma sono comunque molto soddisfatto per quello che sono riuscito a portare a termine nonostante le problematiche. Prenderò pertanto questa esperienza come un grosso insegnamento per me stesso e per il mio futuro professionale.

Appendice A

Guida utente

L'uso del software è piuttosto intuitivo. L'utente è accompagnato passo per passo nell'utilizzo del software attraverso le etichette che descrivono le funzionalità messe a disposizione dell'utente e attraverso alcuni tip che lo agevolano nelle scelte. Qualora l'utente dovesse fare un utilizzo scorretto del software viene informato degli eventuali problemi riscontrati, in modo tale che, in un secondo momento possa intervenire correttamente sul suo utilizzo.