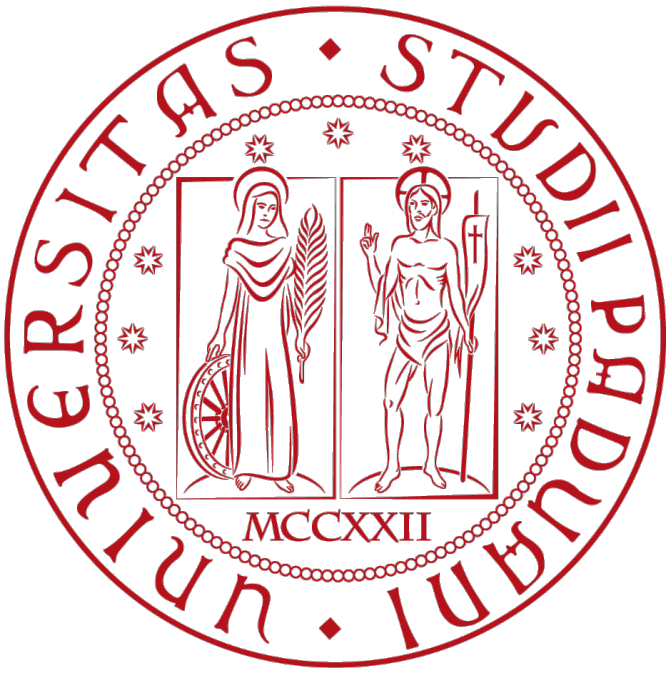
Relazione Progetto Programmazione ad Oggetti 2014/2015

Suierica Bogdan Ionut 1008089

March 23, 2015



1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è quello di presentare le principali scelte architetturali del progetto.

1.2 Scopo del progetto

Lo scopo del progetto è lo sviluppo in C++/Qt di un sistema minimale per l'amministrazione ed utilizzo tramite interfaccia utente grafica di un (piccolo) database di contatti professionali ispirato a Linkedin.

1.3 Specifiche progettuali

Il progetto è stato sviluppato in un sistema Mac Os X Yosemite. Nello sviluppo è stato utilizzato l'ide Qt Creator versione 3.3.0 con versione delle librerie 5.4.4. Il tool Qt Designer non è stato utilizzato nello sviluppo delle interfacce grafiche ma solo per avere un riferimento grafico.
Il progetto è stato provato sui computer del laboratorio in Paolotti e compila ed esegue correttamente.
Il programma consente di aprire e salvare il database in formato XML. L'apertura e il salvataggio del database avvengono in automatico in quanto il path è di default.
Nel progetto ho cercato di dividere il più possibile la parte logica dalla parte grafica.

2 Classi logiche

Ho sviluppato una gerarchia da 4 classi:

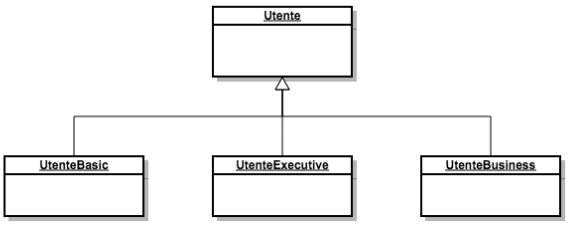


Figure 1: Gerarchia Utenti

La classe base è la classe *Utente* e rappresenta un utente LinQedIn generico. Per questo motivo si è scelto di rendere questa classe astratta e quindi non è possibile dichiarare oggetti di questo tipo. Non avedo a disposizione metodi virtuali puri è stato scelto di rendere il distruttore virtuale puro.
Le classi *UtenteBasic*, *UtenteExecutive* e *UtenteBusiness* sono classi concrete derivate direttamente dalla classe base *Utente*.
La classe *Rete* raffigura la relazione di amicizia ed è rappresentata dal username del contatto.
La classe *InfoPersonali* è rappresentata da un nome, cognome, anno di nascita e una descrizione dell'utente.
La classe *Profilo* rappresenta il profilo di un utente ed è stato scelto di rappresentarla come vector di puntatori a *Competenze*, *Esperienze*, *Formazione*, *Lingue*.

- Classe Competenze: raffigura le competenze di un utente ed è rappresentata dal nome della competenza e il voto che può essere espresso con un numero compreso tra 1 e 10, dove 1 è il minimo e 10 e il massimo;
- Classe Esperienze: raffigura le esperienze nel campo lavorativo del utente ed è rappresentata dal nome dell'azienda in cui ha lavorato e il periodo tra il quale ha lavorato;
- Classe Formazione: con formazione si intendono tutti i diplomi acquisiti in seguito a un percorso di studio. La classe è rappresentata dal nome del ramo/indirizzo/facoltà e dall'anno conseguimento diploma;

- Classe Lingua: raffigura le lingue studiate dal utente. La classe è rappresentata dal nome della lingua e il livello di conoscenza della lingua:
  - Basic;
  - Avanzato;
  - Eccelente;
  - Madre Lingua;

L'utente è caratterizzato da:

- Username: univoco;
- Informazioni Personali: rappresentato tramite un puntatore alla classe *InfoPresonali*;
- Profilo: rappresentato tramite un puntatore alla classe *Profilo*;
- Rete di contatti: rappresentato tramite un vector di puntatori alla classe *Rete*. È presente il distruttore profondo.

La classe *DB* è la classe collezione utenti ed è stato scelto di rappresentarla tramite un vector di puntatori a *Utente* sfruttando l'ereditarietà. La classe *DB* contiene anche un distruttore profondo e metodi di aggiunta e rimozione dei singoli elementi. Per gestire il contenuto del database su un file *XML* sono state utilizzate le seguenti classi offerte da Qt:

- QDomDocument;
- QXmlStreamWriter.

Come contenitori ho scelto di usare i vector in quanto l'inserimento viene fatto sempre in coda in tempo O(1). Gli oggetti dei vector vengono allocati esclusivamente sullo heap.

2.1 Ricerca

La funzionalità di ricerca cambia in funzione alla propria tipologia di account. Le tre tipologie di account descritte sopra permettono delle funzionalità di ricerca crescenti:

- Basic permette una ricerca che controlla solamente se esiste l'utente. Ritorna il nome, cognome e l'anno di nascita;

- Business permette una ricerca che ritorna tutte le informazioni di un utente meno i contatti dell'utente ricercato;
- Executive permette una ricerca completa.

La ricerca è stata implementata attraverso i funtori ed è stato utilizzato come riferimento lo scheletro logico con funtori offerto dal Prof. Ranzato.

3 Classi grafiche

La classe *MainWindow* è la finestra principale del programma. Alla sua creazione viene costruito anche l'oggetto *DB* che contiene il database degli utenti. Qui è possibile selezionare la modalità in cui si vuole accedere, utente o amministratore.

Figure 2: Finestra principale

La classe *ClientWindow* è la finestra destinata agli utenti del servizio. Qui si può modificare il proprio profilo e ricercare altri utenti. È stato scelto la derivazione da *QDialog* per fare in modo che sia una finestra bloccante e non possa interagire con altre finestre del programma finché non sarà chiusa.

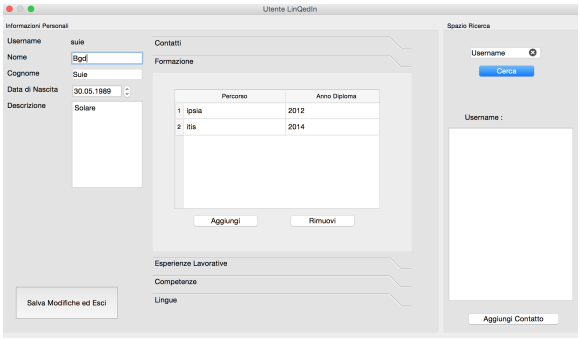


Figure 3: Finestra utente

La classe *AdminWindow* è la finestra destinata all'amministratore. Qui è possibile aggiungere un nuovo utente, selezionare un utente e cambiare la tipologia di account tra basic, executive e business cosi come la rimozione o la ricerca di un utente. È stato scelto la derivazione da *QDialog* per fare in modo che sia una finestra bloccante e non possa interagire con altre finestre del programma finché non sarà chiusa.

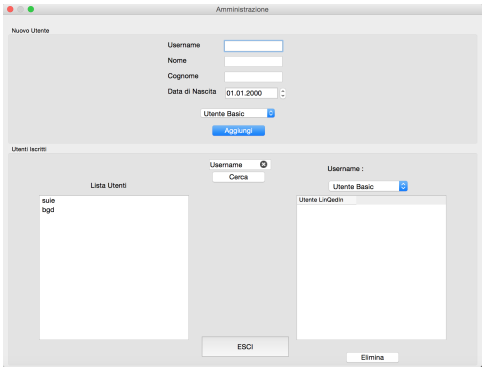


Figure 4: Finestra amministrazione

La ricerca e l'autenticazione viene fatta tramite username. Al momento del autenticazione e della ricerca viene effettuato un controllo sulla presenza o meno del username. Nel caso in cui l'username non fosse presente, viene visualizzato un messaggio di errore.

4 Gestione della memoria

Nel progetto gli oggetti di tipo *Utente* e derivati da *Utente*, *Profilo*, *InfoPersonali*, *Rete* sono allocati esclusivamente sullo heap. Sono stati ridefiniti i distruttori e quindi le classi sono fornite di un distruttore profondo che permette la deallocazione degli oggetti dallo heap nel momento in cui viene distrutto il puntatore. Se l'utente decide di chiudere tutte le finestre viene eseguita un'operazione di svuotamento dei vector che prima dealloca gli oggetti puntati dai puntatori e poi rimuove tutti gli elementi del vector.