Università degli studi di Modena e Reggio Emilia
Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea in Ingegneria Informatica

QUICKFAX PER MAC OS X: SVILUPPO DI UN'APPLICAZIONE SOFTWARE PER UN'AZIENDA INFORMATICA

Relatore
Prof. Sonia Bergamaschi

Candidato
Arletti Luca

Anno Accademico 2012/2013

“Il design non riguarda semplicemente l’aspetto di un oggetto.
Il design è come un oggetto funziona.”

- Steve Jobs

Indice

Introduzione **1**

Panoramica dei capitoli **2**

1. Specifiche e requisiti, funzionalità e tecnologie utilizzate **3**
- 1.1 Il progetto 3
- 1.2 Interfaccia grafica Xcode 4
- 1.3 Interface Builder 7

2. Realizzazione dell'interfaccia grafica di base **9**
- 2.1 Le viste e il pattern MVC 12
- 2.2 Realizzazione Menu Bar 15
- 2.3 Impostazione icone pulsanti 17
- 2.4 Switching di views: algoritmo per la gestione del passaggio da una vista all'altra 19

3. La view dei fax recenti **25**
- 3.1 La Table View 26
- 3.2 Popolare la tabella 28
- 3.3 Cella personalizzata 29
- 3.4 Interazioni dell'utente 30
- 3.5 La vista di dettaglio 31
  - 3.5.1 Addressbook e ricerca contatti 31
  - 3.5.2 Switching di views a scorrimento laterale 33

4. Le viste di invio fax **37**
- 4.1 La view di invio fax di solo testo 37
  - 4.1.1 Filtraggio dei contatti della rubrica 39
  - 4.1.2 Funzione di invio fax 43
- 4.2 La view di invio fax con allegato 45
  - 4.2.1 Drag & Drop 46
  - 4.2.2 Anteprima file 49
  - 4.2.3 Il framework Quick Look 50
  - 4.2.4 Funzione di invio fax 53

5. Cenni alle viste di login, registrazione e impostazioni **55**
- 5.1 La view di login 55
- 5.2 La view di registrazione 58
- 5.3 La view impostazioni 60

6. Conclusioni e sviluppi futuri **63**

Bibliografia **65**

Introduzione

Il presente elaborato è stato sviluppato a seguito di un'attività di stage presso l'azienda Command Guru s.r.l. di Carpi. L'esperienza è durata 3 mesi, dall'inizio di Giugno 2013 a inizio Settembre dello stesso anno.

L'azienda Command Guru s.r.l. ha nel suo portafoglio software il prodotto Faxator il quale offre agli utenti la possibilità di inviare e ricevere fax utilizzando semplicemente il proprio computer. E' sufficiente registrarsi sul sito dedicato per poter usufruire di tale servizio; l'utente registrato può dunque inviare e ricevere fax utilizzando la propria casella di posta elettronica o accedendo alla propria area riservata sul sito.

L'azienda ha successivamente sviluppato quickFax, un'applicazione disponibile per dispositivi mobili aventi sistema operativo Android o iOS, la quale sfrutta il servizio Faxator per consentire all'utente di inviare fax direttamente dal proprio smartphone o tablet.

L'obiettivo di questa tesi è quello di descrivere la metodologia di progetto utilizzata per la realizzazione di questa applicazione in ambiente Mac OS X la quale, sfruttando le funzionalità messe a disposizione dal servizio Faxator, consente all'utente di inviare fax attraverso un'interfaccia grafica semplice, intuitiva e notevolmente user friendly, fornendo così all'utente finale un servizio ottimizzato senza dover utilizzare strumenti intermedi (come un client di posta elettronica).

Al fine di facilitare la lettura dell'elaborato sono state utilizzate colorazioni diffferenti per evidenziare la sintassi del linguaggio come descritto in legenda:

- Direttive e parole chiave
- Classi standard
- Classi realizzate

Panoramica dei capitoli

Capitolo 1: "Specifiche e requisiti, funzionalità e tecnologie utilizzate"
In questo capitolo vengono presentate le funzionalità fondamentali che il programma applicativo dovrà fornire all'utente; vengono inoltre analizzati due strumenti fondamentali quali XCode, l'ambiente di sviluppo e InterfaceBuilder, il tool utilizzato per la realizzazione dell'interfaccia grafica.

Capitolo 2: "Realizzazione dell'interfaccia grafica di base"
Questo capitolo è dedicato alla realizzazione dell'interfaccia grafica iniziale del programma applicativo.Vengono presentate le varie fasi che consentono la realizzazione dell'interfaccia grafica che costituirà il corpo del programma applicativo; sono inoltre descritte le viste e le classi fondamentali utilizzate, l'impostazione delle icone dei pulsanti, l'aggiunta del programma applicativo alla menubar e l'algoritmo per lo switching delle viste.

Capitolo 3: "La view dei fax recenti"
Questa parte è dedicata all'analisi e alla realizzazione della vista che mostra all'utente lo storico dei fax da lui inviati; viene descritto in maniera approfondita l'oggetto TableView utilizzato, come personalizzare e popolare la tabella. E' inoltre presentata la vista di dettaglio relativa ad un fax inviato e l'algoritmo che realizza l'animazione a scorrimento laterale.

Capitolo 4: "La viste di invio fax"
In questo capitolo si analizzano le due viste che consentono all'utente di inviare rispettivamente un fax di solo testo o con allegato. Vengono presentate e descritte in maniera dettagliata diverse funzionalità quali il filtraggio dei contatti della rubrica, l'algoritmo di invio del fax, il drag & drop e l'anteprima di un file mediante l'uso del framework Quick Look.

Capitolo 5: "Cenni alle viste di login, registrazione e impostazioni"
Questo capitolo è dedicato alla descrizione delle viste che consentono l'autenticazione e la registrazione di un utente; nella parte finale del capitolo si descrive invece la view impostazioni.

Capitolo 6: "Conclusioni e sviluppi futuri"
In questo capitolo vengono riassunti i risultati ottenuti e gli eventuali sviluppi futuri.

Capitolo 1

Specifiche e requisiti, funzionalità e tecnologie utilizzate.

1.1 Il progetto

Il progetto quickFax per Mac nasce dall'idea di realizzare un'applicazione che funzioni in ambiente Mac OS X la quale consenta di utilizzare il servizio Faxator mediante un'interfaccia grafica semplice ed intuitiva.

L'idea di base era quella di realizzare tale progetto ispirandosi all'applicazione quickFax già esistente per dispositivi mobile, in modo tale che gli utenti che già la utilizzavano potessero ritrovare anche in ambiente OS X un'applicazione dal design simile e che offrisse loro le stesse funzionalità, mentre per i nuovi utenti questa applicazione risulta essere facilmente comprensibile senza presentare alcun tipo di difficoltà nell'utilizzo.

Il progetto quickFax per Mac (sebbene si ispiri all'applicazione quickFax già esistente) è iniziato da zero; le uniche classi che sono state riutilizzate sono le librerie API che consentono la gestione di tutti i servizi offerti da Faxator (invio fax, visualizzazione fax recenti e altri), ovvero una serie di classi che si occupano di trasformare i tipi di dato gestiti da Objective-C e Cocoa (linguaggio e framework utilizzati nell'ambiente di lavoro Xcode per la realizzazione di applicazioni in ambiente Mac OS X) e convertirli in tipi di dato che siano interpretabili dal servizio Faxator per poter effettuare correttamente l'interrogazione col server, ad esempio per la registrazione di un nuovo utente o la visualizzazione dei fax recenti associati ad un utente. [1,2]

Ciò che mi è stato chiesto di realizzare era un'applicazione composta da quattro pulsanti (uno per ogni funzione gestita) con cui l'utente può interagire con l'applicazione per selezionare la funzione desiderata.
Mediante la pressione di uno dei pulsanti il programma doveva mostrare la vista specifica per la funzionalità richiesta dall'utente.

Le funzionalità che il programma deve gestire sono essenzialmente:

- Invio fax di solo testo
- Invio fax con allegato
- Visualizzazione fax recenti
- Impostazioni

Ad ognuna di queste funzionalità è associata una vista; in questo modo è possibile tenere separate, sia a livello logico che di programmazione, l'implementazione delle funzionalità richieste e la risoluzione dei vari problemi.
Nello specifico ad ogni vista sarà associato un controllore (il *viewController*) il quale

conterrà i riferimenti agli oggetti grafici e i metodi specifici solamente per la vista che controlla, consentendo una notevole modularità del codice. Ogni vista risulta quindi essere "indipendente" da tutte le altre.
Il passaggio da una vista all'altra è possibile mediante la pressione di uno dei quattro pulsanti, ad ognuno dei quali è associata una e una sola vista.

Inoltre il programma, una volta eseguito, deve essere presente nella *menu bar* in alto a destra in modo tale da risultare sempre accessibile all'utente, senza essere ingombrante o troppo invasivo una volta aperto (l'idea si è ispirata alle moderne applicazioni come ad esempio Dropbox e Facebook).
Dopo questa breve introduzione vediamo come realizzare tale applicazione analizzando due strumenti fondamentali: ***Xcode e Interface Builder***.

1.2 Interfaccia grafica Xcode

Xcode è un ambiente di sviluppo integrato (*Integrated development environment, IDE),* sviluppato da Apple per agevolare lo sviluppo di software per Mac OS X e iOS. Esso è utilizzato nella scrittura di codice, nell'organizzazione delle classi, nella rilevazione e correzione di eventuali errori e nella realizzazione dell'interfaccia grafica mediante lo strumento integrato Interface Builder. [3]

Creiamo un nuovo progetto da File > NewProject e selezioniamo CocoaApplication.
Dopo aver dato un nome al nuovo progetto (chiamato semplicemente quickFax) comparirà la seguente finestra.

L'interfaccia grafica di Xcode è suddivisibile essenzialmente nelle seguenti parti fondamentali:

- Area editor: Visualizza il contenuto di un file selezionato nel menu a sinistra e consente la scrittura e modifica del codice che andrà a costituire la nostra applicazione.

• Area Navigazione: Grazie a tale menù possiamo accedere ai nostri file di progetto, creare e aggiungere nuovi file (classi, framework, immagini e altri) al nostro progetto.
I file sorgenti dove risiede il codice Objective-C sono all'interno della cartella avente lo stesso nome del progetto; inoltre è possibile visualizzare eventuali errori o warning, visualizzare i breakpoint attivi, visualizzare una gerarchia delle classi utilizzate, ricercare parole chiave all'interno del codice.

Tutte queste funzioni sono disponibili semplicemente selezionando uno dei seguenti pulsanti:

• Area Utility: Essa mostra tutte le informazioni essenziali relative al file selezionato nell'area di navigazione. Le informazioni e opzioni visualizzate cambiano a seconda del tipo di file; in particolar modo cambia radicalmente nel caso in cui venga selezionato un file grafico di tipo .xib (il quale verrà descritto in dettaglio in seguito quando sarà introdotta la funzione Interface Builder di Xcode).

• Area di Debug: In basso troviamo la console dove potremo stampare a video utili informazioni durante l'esecuzione dell'applicazione.

Nota: l'utilizzo della console, unito ad un corretto uso dei breakpoint è la base indispensabile per poter trovare e correggere gli errori!

• Toolbar: Nella toolbar sono presenti diversi pulsanti importanti che riassumiamo di seguito:

Due pulsanti fondamentali presenti nella toolbar sono i pulsanti per l'esecuzione e lo stop dell'applicazione mostrati in figura. Tramite il tasto con l'icona di play è possibile compilare ed eseguire l'applicazione mentre con il tasto dell'icona stop possiamo interrompere l'esecuzione.
Non è presente un tasto predisposto per la sola compilazione dell'applicazione; esistono tuttavia delle shortcut da tastiera che consentono di compilare eseguire e arrestare l'applicazione e sono:

- Build: shift+cmd+B
- Run: cmd+R
- Stop: shift+cmd+invio

Un altro pulsante utile è quello che si trova alla destra dei pulsanti run e stop e permette di decidere quale schema utilizzare nella fase di compilazione ed esecuzione dell'applicazione. Dato che il nostro progetto verrà sviluppato per Mac su sistema OS X 10.8 a 64bit, una volta impostato correttamente non sarà mai necessario modificare tale schema.

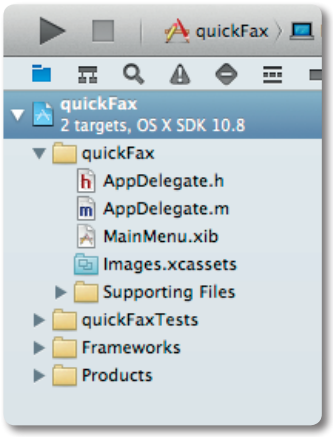
Possiamo mostrare e nascondere la console tramite il pulsante cerchiato di rosso. Quello verde serve invece per mostrare il menu di sinistra contenente la gerarchia dei file, mentre quello azzurro ci permetterà di accedere al menu di informazioni sui file (o di Interface builder se selezioniamo un file .xib).

Mediante i seguenti tre pulsanti è possibile modificare la tipologia dell'editor centrale:

- Standard Editor : il contenuto del file selezionato viene visualizzato singolarmente.

- Assistant Editor : l'editor viene diviso in due parti dove è possibile visualizzare il contenuto di due file differenti.

- Version Editor : visualizza le differenze tra un file selezionato e un'altra versione dello stesso file (molto utile per verificare le modifiche che sono state fatte ad un file sorgente).



Concentriamoci ora sull'area di navigazione, più precisamente sui tre file che sono presenti nella gerarchia dei file del nostro progetto; essi sono i file di base di ogni applicazione Cocoa e si distinguono in:

- **AppDelegate.h**: header del file delegato ai compiti che saranno eseguiti dall'applicazione.
Contiene la sezione @interface, ovvero la sezione pubblica della classe.
- **AppDelegate.m**: file di implementazione del delegato, dove sarà presente il codice.
Contiene la sezione @implementation dove vengono definiti i metodi, ovvero le azioni che la classe è in grado di compiere.

MainMenu.xib: file di supporto per la costruzione dell'interfaccia grafica dell'applicazione.

A seguire troviamo altri file a supporto:

main.m: il programma principale, manda in esecuzione il codice contenuto in AppDelegate lasciando a quest'ultimo il controllo di flusso dell'applicazione.

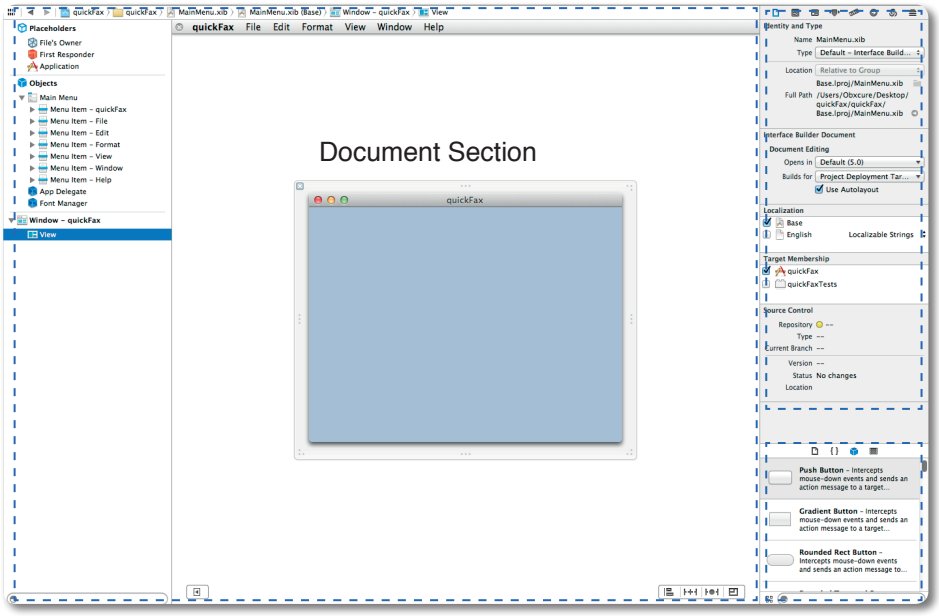
quickFax-Info.plist: questo file contiene molte informazioni ma la riga fondamentale è quella relativa a "Main nib file base name" dove nella colonna value è presente la dicitura MainMenu.

Quindi quando si crea un'applicazione Cocoa, il framework cerca il valore appena citato nel file NomeApplicazione-Info.plist e utilizza il file con quel nome per creare l'interfaccia di base dell'applicazione.
All'interno del file AppDelegate.m troviamo sempre il metodo `applicationDidFinishLaunching:` ; tale metodo rappresenta sostanzialmente il primo metodo che verrà eseguito non appena l'applicazione verrà eseguita.
Ora concentriamoci sul file MainMenu.xib.

Se selezioniamo questo file l'interfaccia grafica di XCode muterà radicalmente per lasciare spazio allo strumento InterfaceBuilder che consente la costruzione dell'interfaccia grafica della nostra applicazione.

1.3 Interface Builder

Interface Builder è un tool integrato all'interno di Xcode, che viene usato per la realizzazione delle interfacce grafiche, generando un file .xib. L'utilizzo del tool è immediato: è sufficiente trascinare all'interno dell'area di lavoro gli elementi grafici che si vogliono utilizzare (come bottoni, campi di testo, immagini) per poi passare al loro posizionamento. [5]



L'interfaccia di IB (Interface Builder) è suddivisibile in tre sezioni fondamentali:

- Library Section: questa sezione contiene al suo interno tutti gli elementi grafici raccolti per tipologie (label, bottoni, view, table view a altri). Per inserire un nuovo elemento nel nostro file .xib sarà sufficiente trascinare l'elemento desiderato all'interno dell'area di lavoro.

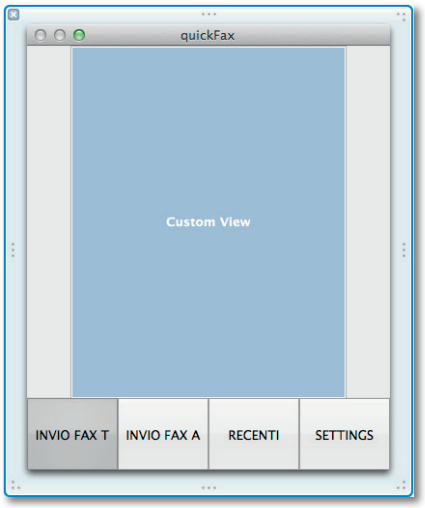
- Document Section: rappresenta l'ara di lavoro vera e propria dove sono presenti tutti gli elementi grafici che costituiscono la nostra interfaccia. Tutti gli elementi grafici presenti sono raggruppati in un diagramma ad albero che mostra (per ogni livello presente) quali sono gli elementi che appartengono a tale livello.

- Inspector Section: da questa sezione è possibile accedere e modificare tutti gli attributi che sono associati all'elemento grafico selezionato nell'area di lavoro. L'Inspector Section possiede un menù dal quale è possibile accedere agli attributi, alle connessioni, alle dimensioni dell'oggetto in esame e ad altre sue proprietà.

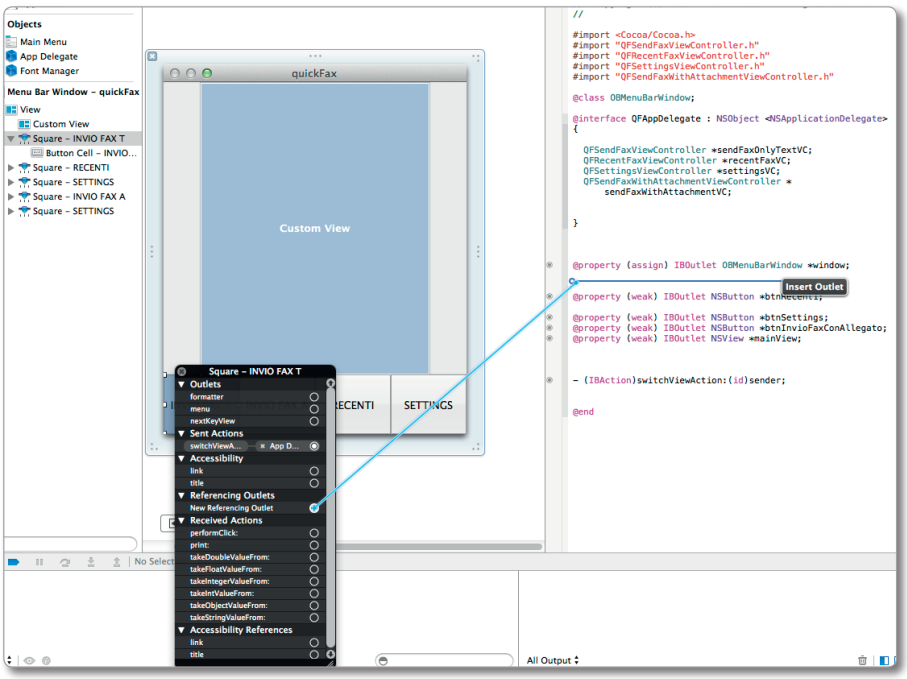
Capitolo 2

Realizzazione dell'interfaccia grafica di base

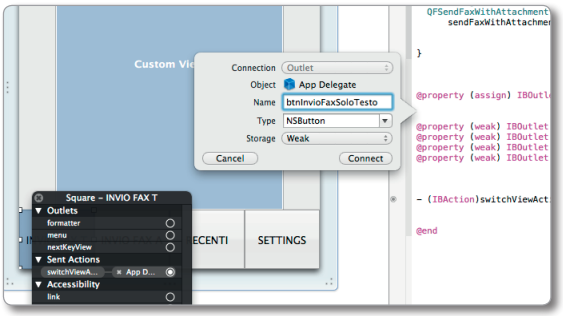
Procediamo ora alla realizzazione dell'interfaccia grafica iniziale.
Selezioniamo il file MainMenu.xib e aggiungiamo quattro pulsanti di tipo *push button* e una *custom view*; tali elementi sono presenti nella Document Section come appena descritto. Disponiamo tali elementi come in figura:



Una volta aggiunti i pulsanti dovremo cambiarne il tipo, da "*momentary push*" in "*push on push off*": questo tipo di pulsante assume un aspetto quando si trova nello stato on mentre ne assume uno diverso quando si trova nello stato off. In questo modo risulterà immediato capire quale pulsante risulta essere selezionato e quali invece deselezionati.
Per la nostra tipologia di progetto è requisito fondamentale che un solo pulsante alla volta possa essere attivo; dobbiamo dunque impostare di default lo stato del primo pulsante a on e tutti gli altri a off.
E' necessario creare un riferimento agli oggetti grafici aggiunti mediante Interface Builder per poter agire su di essi mediante codice.
Per fare ciò è necessario passare in modalità di visualizzazione affiancata (mediante il pulsante Assistant Editor descritto precedentemente) e visualizzare sia il file MainMenu.xib che il file AppDelegate.h.
Selezionando un oggetto grafico presente in MainMenu.xib, tenendo premuto il tasto "ctrl" e trascinando il cursore del mouse all'interno della sezione @interface della classe corrispondente (nel nostro caso l'AppDelegate.h), sarà possibile effettuare un collegamento dell'oggetto grafico, in modo da interagire con esso direttamente a livello di codice; un esempio di tale collegamento è mostrato in figura:



A questo punto è bene soffermarsi su alcuni concetti importanti; nel momento in cui si esegue il collegamento suddetto, apparirà una finestra nella quale è possibile selezionare la voce *Outlet* o *Action* creando di conseguenza un oggetto di tipo IBOutlet o un metodo IBAction, riferito all'oggetto grafico selezionato; vediamo cosa significano questi due termini:



IBOutlet: Gli IBOutlet (acronimo di *Interface Builder Outlet*) sono le variabili che rappresentano gli oggetti grafici all'interno del codice e permettono di modificare l'oggetto tramite codice specifico.

La sintassi tipica per un oggetto IBOutlet è la seguente:

```
@property (valore) IBOutlet NomeClasse *nomeoutlet;
```

La parola chiave IBOutlet, precede il nome della classe e indica che quell'oggetto risulta essere "collegato" al corrispondente presente nella parte grafica.
La parole chiave @property invece permette di indicare il comportamento dell'oggetto; i valori che possiamo avere sono i seguenti:

- **assign**: viene utilizzato un semplice assegnamento per il metodo set degli attributi
- **readonly**: viene richiesto solo il metodo get dell'attributo.
- **readwrite**:viene richiesto sia il metodo get che set.
- **copy**: viene utilizzata una copia dell'oggetto per l'assegnamento.
- **nonatomic** :il metodo creato avrà caratteristiche di atomicità.

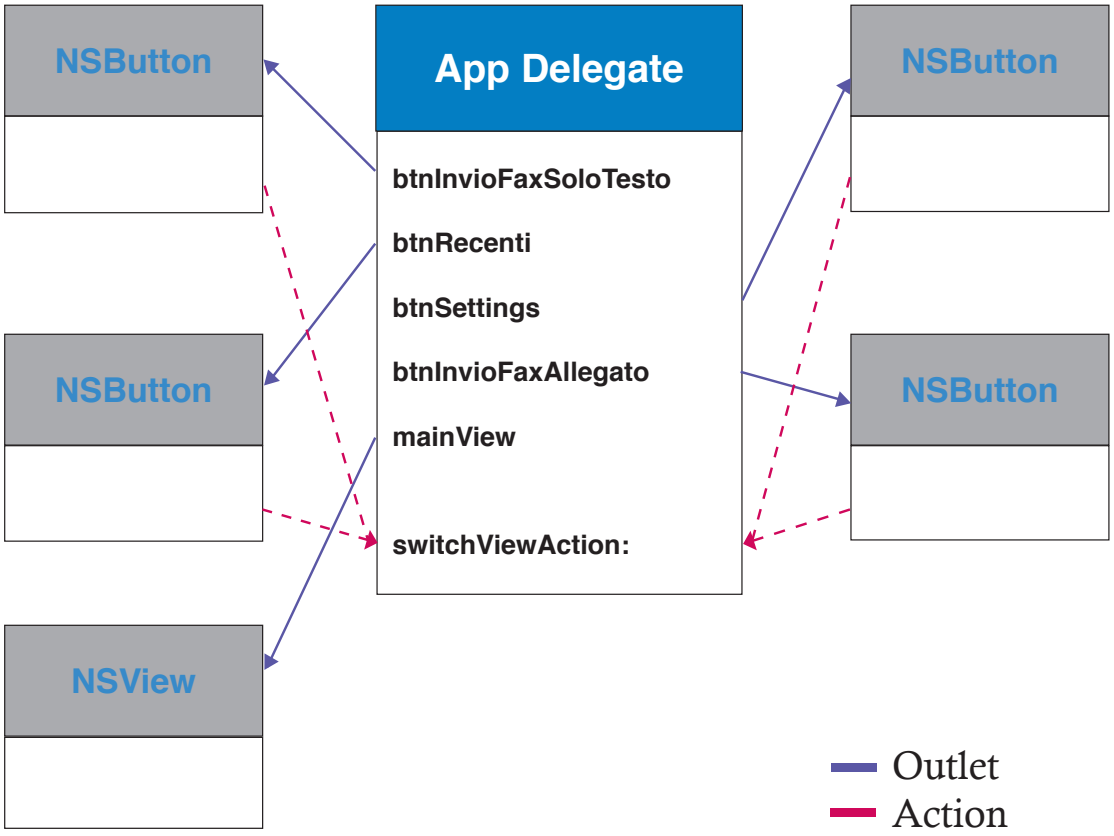
La @property viene (quasi) sempre utilizzata in coppia con la parola chiave @synthesize che troveremo nell'AppDelegate.m.
Con l'utilizzo di tale direttiva si comunica al compilatore di sintetizzare una coppia di metodi accessori (setter e getter) in accordo con la dichiarazione nella sezione @interface dell'AppDelegate.h.

Nota: se non si utilizza la direttiva @synthesize è possibile comunque accedere agli outlet creati facendo precedere al loro nome il simbolo "_"; in questo modo si potrà accedere all'outlet solamente all'interno della classe nella quale è stato dichiarato.

IBAction: Le IBAction (acronimo di *Interface Builder Action*) sono funzioni che vengono richiamate da determinate azioni; esse identificano un'azione collegata a uno o più oggetti grafici presenti nell'interfaccia dell'applicazione.
E' una funzione (più precisamente un metodo di istanza) che non restituisce alcun valore e accetta come parametro di ingresso un oggetto di tipo id, ovvero sarà assegnata la classe di appartenenza di quell'oggetto. La sintassi tipica per la dichiarazione di un metodo IBAction è la seguente:

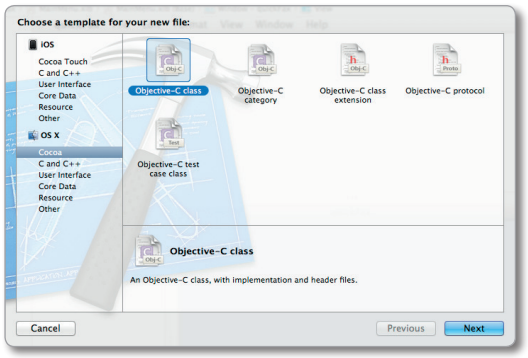
```
- (IBAction)nomeAction:(id)sender;
```

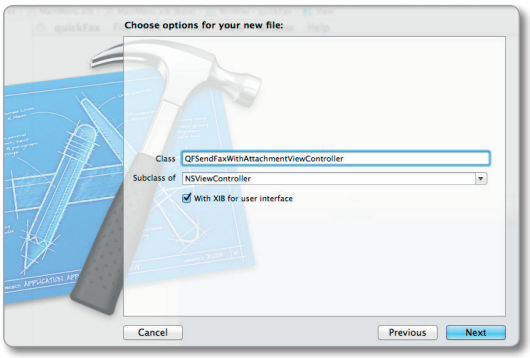
Colleghiamo tutti i pulsanti al metodo `switchViewAction:` mediante il quale verrà gestita l'azione che dovrà compiere ogni pulsante nel momento in cui verrà premuto; ciò significa che ogni volta che viene premuto uno dei quattro pulsanti viene eseguito il codice presente in `switchViewAction:` .
Si dispone ora di un IBOutlet per ogni pulsante e di un metodo IBAction chiamato `switchViewAction:` invocato ogni volta che l'utente premerà uno dei pulsanti.
Inoltre disponiamo anche di un IBOutlet per la *customView*, che chiameremo (in riferimento al codice) *mainView*. Tale IBOutlet servirà per decidere quale vista mostrare a seconda del pulsante premuto dall'utente.
Tutto ciò viene riassunto nel seguente schema:



2.1 Le viste e il pattern MVC

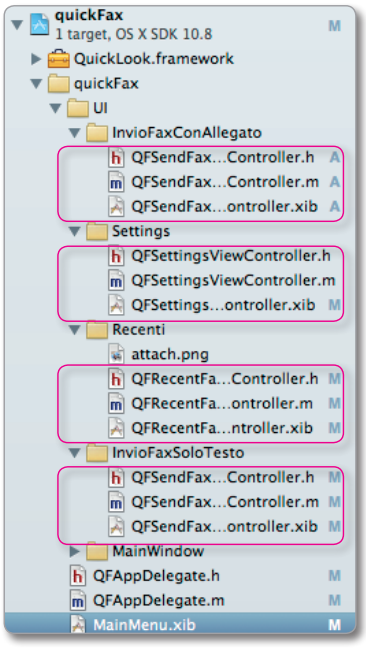
A questo punto disponiamo di un IBOutlet per ogni pulsante e di un metodo IBAction chiamato `switchViewAction:`, invocato ogni volta che l'utente premerà uno dei pulsanti presenti a schermo.
Inoltre disponiamo anche di un IBOutlet per la *customView*, che chiameremo (in riferimento al codice) *mainView*. Tale IBOutlet ci servirà per decidere quale vista mostrare a seconda del pulsante premuto dall'utente.
Ora ciò che ci serve è creare quattro viste, ognuna delle quali si occuperà di una delle funzionalità specifiche del programma e potrà essere visualizzata mediante la pressione del pulsante dedicato.
Per creare una vista è necessario creare una nuova classe; per fare ciò andiamo in File>New>File..
selezioniamo Objective-C Class e creiamo una classe che erediti dalla classe NSViewController.





La classe NSViewController fornisce il modello fondamentale di gestione delle view per le nostre applicazioni. [6]
Nel caso in cui siano presenti più NSViewController (come nel nostro caso) sarà necessario, come vedremo più avanti, implementare tecniche di swithching di views, in accordo con il flusso logico dell'applicazione.
In generale tutte le applicazioni realizzate in ambiente Mac OS X hanno almeno una NSWindow allocata dal delegato della nostra applicazione ed una o piu' NSViewController.

La situazione riportata in figura mostra che per ognuna delle quattro classi create, relative a invio fax di solo testo, invio fax con allegato, fax recenti e impostazioni, sono presenti tre file:



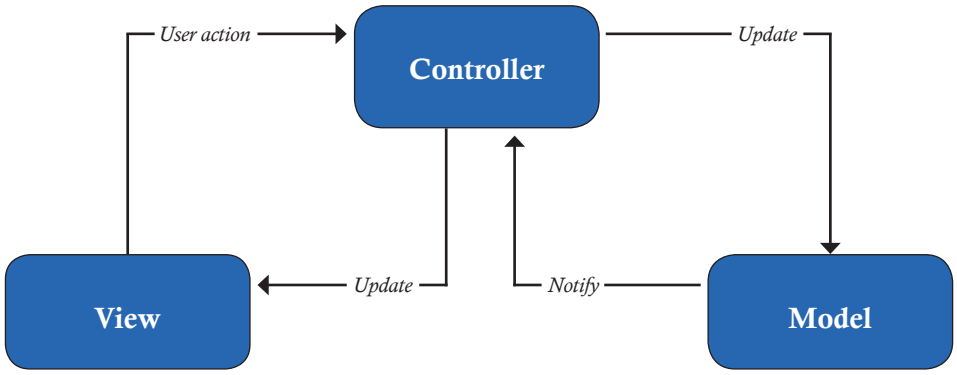
NomeClasse.h : File header della classe contenente la sezione pubblica della classe (ovvero la sezione @interface) dove verranno dichiarate le variabili, i riferimenti agli oggetti grafici presenti nella vista e i metodi pubblici.

NomeClasse.m : File di implementazione, contenente la sezione @implementation nella quale si trova l'implementazione delle azioni che tale classe è in grado di compiere, ovvero i metodi (di classe e di istanza).

NomeClasse.xib : File di Interface Builder dove verranno disposti gli oggetti grafici appartenenti a tale view.

In questo modo, tutta la logica di funzionamento (e il codice necessario) di una specifica vista sarà racchiusa nei file .m e .h che si occuperanno di gestire le interazioni dell'utente e i problemi relativi solamente alla vista che controllano.

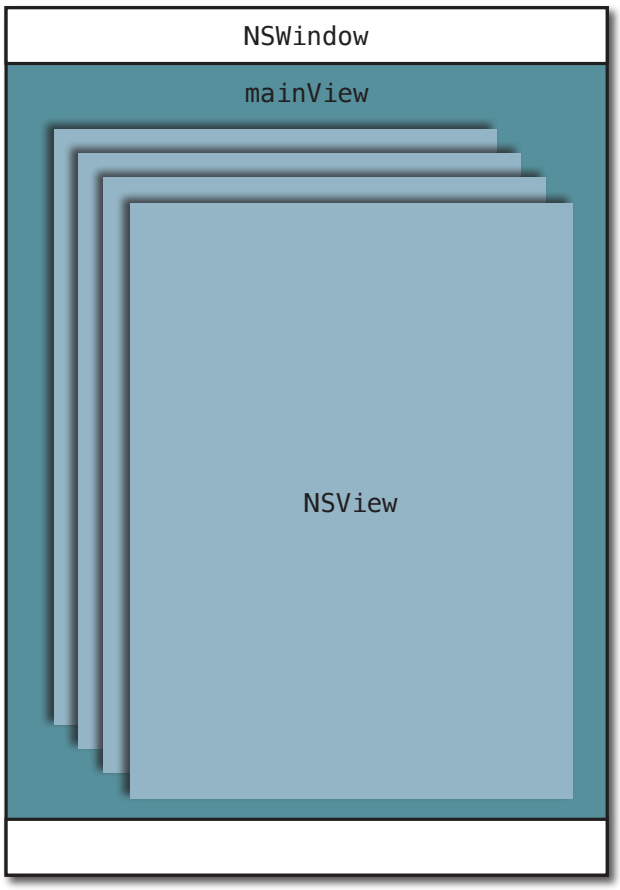
E' importante sottolineare che nel momento in cui si crea una classe che eredita da NSViewController la nuova classe presenta sia il ruolo di view (delegato al file .xib) sia il ruolo di controller (mediante i file .m e .h descritti precedentemente).
Tale osservazione permette di introdurre un concetto importante: il framework Cocoa, così come tutta la logica di programmazione in ambiente Mac OS X e iOS, si basa sul pattern Model-View-Controller, il quale consente di organizzare l'intero sistema in tre principali sottosistemi (Model, View, Controller) caratterizzati da ruoli ben precisi.
Il sottosistema View visualizza i dati, il sottosistema Model incapsula i dati relativi ad un determinato problema mentre il sottosistema Controller fa da mediatore tra View e Model.



Lo schema precedente rappresenta l'interazione fra i tre sottosistema definiti dal pattern MVC; nel momento in cui l'utente interagisce con l'interfaccia grafica (ad esempio facendo click su un pulsante), la View inoltra l'informazione relativa al suo Controller. Il Controller riceve l'evento e applica una strategia per modificare i dati incapsulati all'interno del Model. Nel caso in cui il Model sia stato modificato, esso invia la notifica di avvenuta modifica al Controller il quale si occuperà di aggiornare la View di conseguenza. Il Controller agisce dunque come mediatore, comunicando alla View le modifiche allo stato del Model in modo che la View possa mostrare visivamente le conseguenze di tali modifiche.

Tornando alla classe AppDelegate è necessario, al suo interno, allocare un'istanza di ognuno dei *viewController* appena creati e aggiungere tali view alla *mainView* principale, definita nell'AppDelegate e rappresentata nel file MainMenu.xib dalla custom view precedentemente citata.

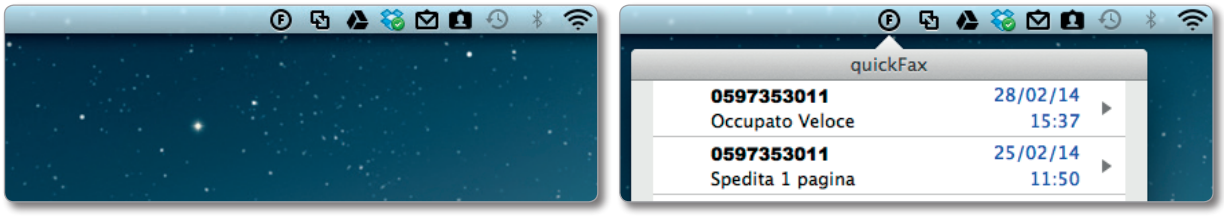
L'idea è quella di aggiungere tutte le view appena create alla mainView (come se fossero sovrapposte), nasconderle tutte rendendole invisibili e mostrare di default solamente la view relativa all'invio fax di solo testo.



Le istruzioni che consentono l'allocazione e l'istanza di ognuno dei *viewController* sono state inserite all'interno del metodo `applicationDidFinishLaunching:` che, come detto precedentemente, è presente nell'AppDelegate.m ed è il primo metodo che verrà eseguito non appena l'applicazione viene avviata. Di conseguenza all'interno di tale metodo saranno inserite tutte le istruzioni necessarie alla corretta inizializzazione dell'applicazione, come ad esempio l'allocazione e l'istanza dei *viewController* sopra citati, l'aggiunta delle view (controllate dai rispettivi viewController) alla *mainView* dell'AppDelegate, l'impostazione del parametro di invisibilità e altre.
A questo punto, una volta che l'utente preme un pulsante a video, viene invocato il metodo da me definito `switchViewAction:` presente nell'AppDelegate, all'interno del quale si identifica il pulsante premuto dall'utente. Ciò è possibile dato che ad ognuno dei pulsanti viene assegnato un *tag*, ovvero un numero identificativo univoco per ognuno di essi. A seconda del pulsante premuto dall'utente viene mostrata la view richiesta rendendola visibile e vengono nascoste tutte le altre view che diventano invece invisibili.
Tale metodo si occupa inoltre di cambiare gli stati dei pulsanti da on a off (in accordo con la view selezionata), in modo da rispettare la seguente specifica: solo un pulsante alla volta deve trovarsi nello stato on e solo una view alla volta deve essere mostrata all'utente.
L'assegnazione dei *tag* ai pulsanti e l'impostazione della view di default vengono eseguiti all'interno del metodo `setupWindow:`, (il quale viene richiamato all'interno del metodo `applicationDidFinishLaunching:` presente nell'AppDelegate).

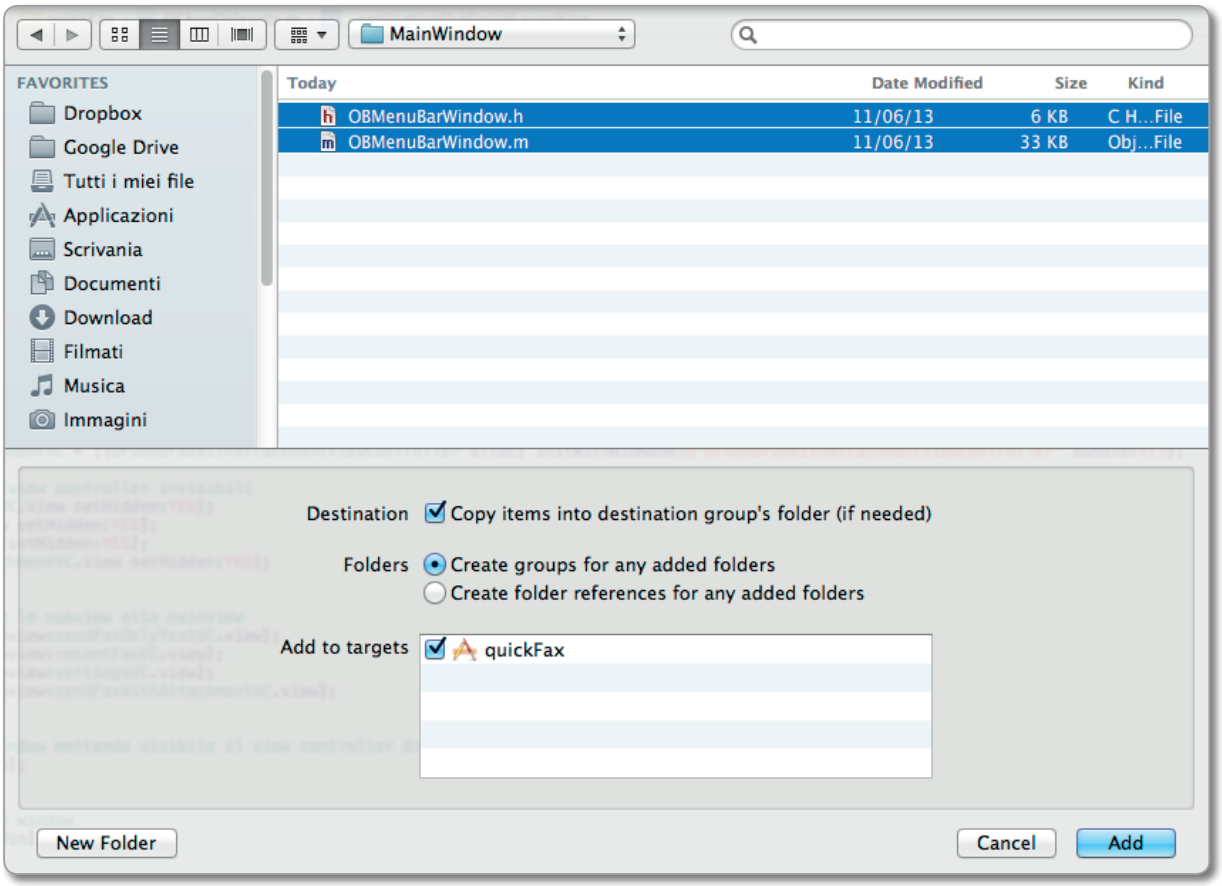
2.2 Realizzazione Menu Bar

Una delle ultime specifiche richieste dal progetto è che il programma sia presente nella *menu bar* in alto a destra, in modo da essere sempre accessibile all'utente in ogni momento. Per rispondere a questa specifica, dopo diverse ricerche relative alla realizzazione di un'applicazione nella *menu bar*, si è deciso di utilizzare una classe già esistente in grado di risolvere questo genere di problema.
La classe in questione prende il nome di OBMenuBarWindow e permette di eseguire un'applicazione nella *menu bar* del Mac; l'interfaccia grafica dell'applicazione viene mostrata solamente quando l'utente preme sull'icona e viene nascosta non appena si fa un click in una porzione di schermo esterna all'interfaccia grafica del programma.



Esempio di menu bar

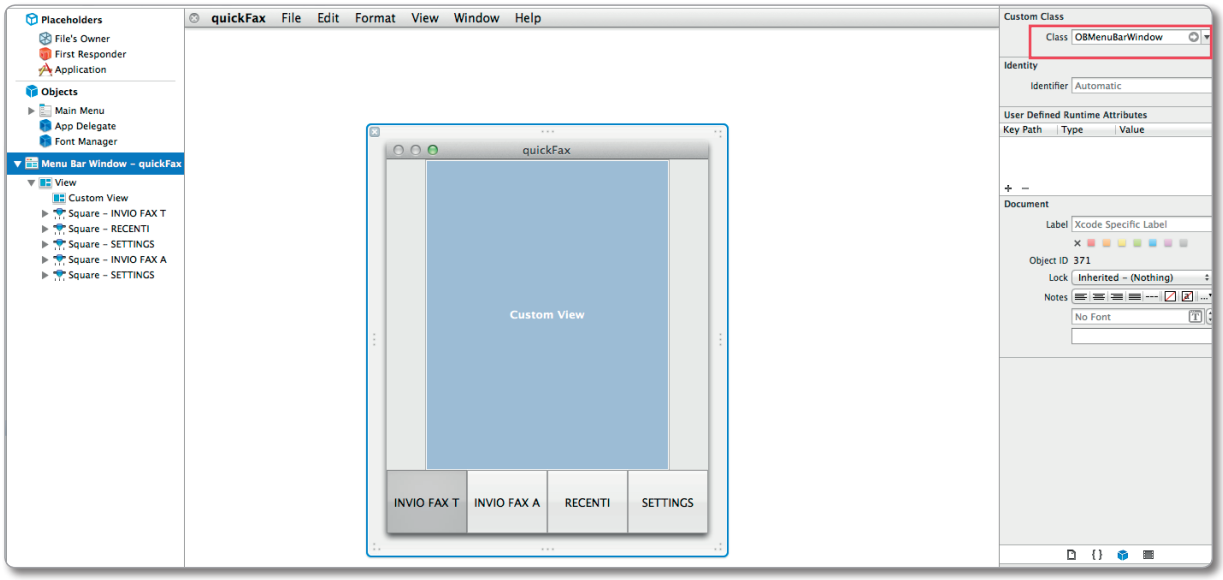
Per utilizzare la classe OBMenuBarWindow all'interno dell'applicazione è necessario importarla all'interno del progetto quickFax. Per fare ciò è necessario fare clic su File->Add File to "quickFax" e selezionare la classe desiderata.



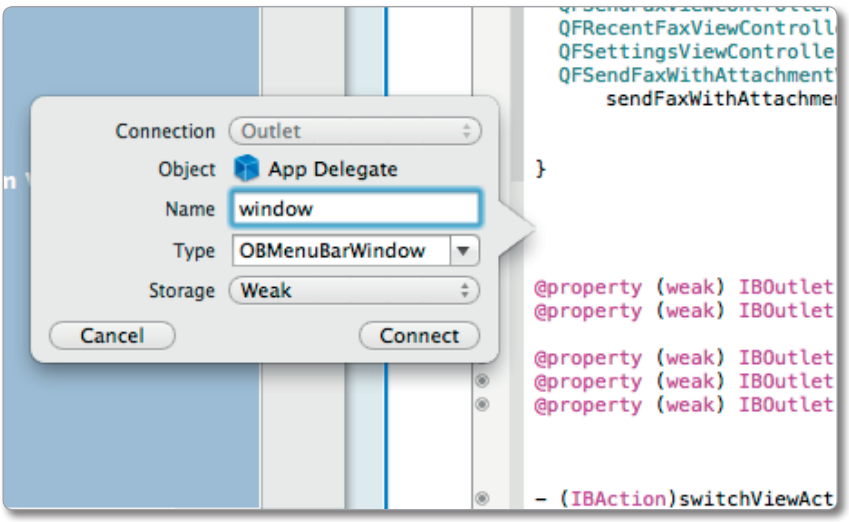
Ora è possibile definire che la classe che gestisce la window principale all'interno del file MainMenu.xib non è quella standard, ovvero NSWindow, ma è la classe OBMenuBarWindow appena importata.

Per fare ciò all'interno del file MainMenu.xib è necessario cliccare sulla dicitura Window a sinistra dell'Interface Builder e selezionale in alto a destra la barra Identity Inspector.
Alla voce class specifichiamo la nostra classe OBMenuBarWindow.

In questo modo specifichiamo esplicitamente che l'oggetto window della nostra applicazione non è un oggetto di tipo NSWindow (classe standard per la gestione delle finestre) bensì è di tipo OBMenuBarWindow (classe custom che mi permette di gestire la finestra come una menu bar).



A questo punto non resta che importare all'interno dell'App Delegate la classe OBMenuBarWindow (mediante la direttiva #import) e creare un IBOutlet relativo alla window per poter accedere ai metodi pubblici messi a disposizione da tale classe (con i quali ad esempio cambiare l'icona nella menu bar o definire se la finestra deve essere sempre in primo piano) come possiamo vedere in figura:



Creazione IBOutlet relativo alla window

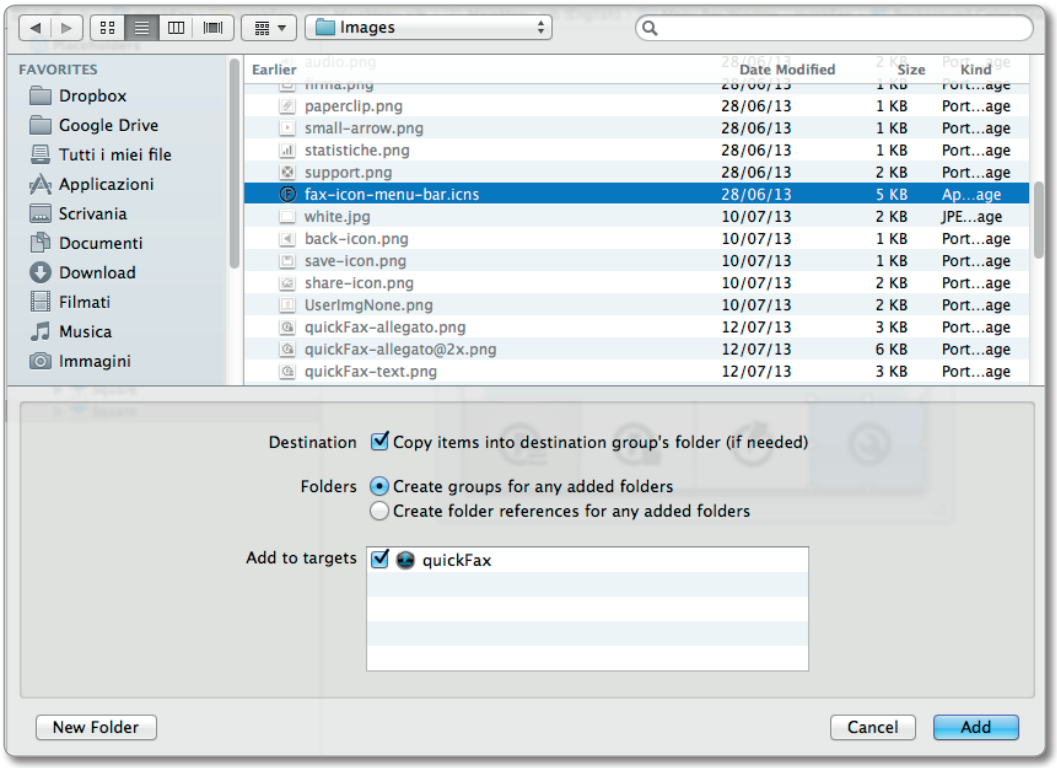
In seguito è stato definito un metodo `launchMainWin:` all'interno dell'AppDelegate, per impostare l'icona che comparirà nella *menu bar* e fare in modo che la finestra risulti "agganciata" all'icona.

2.3 Impostazione icone pulsanti

Design icone pulsanti

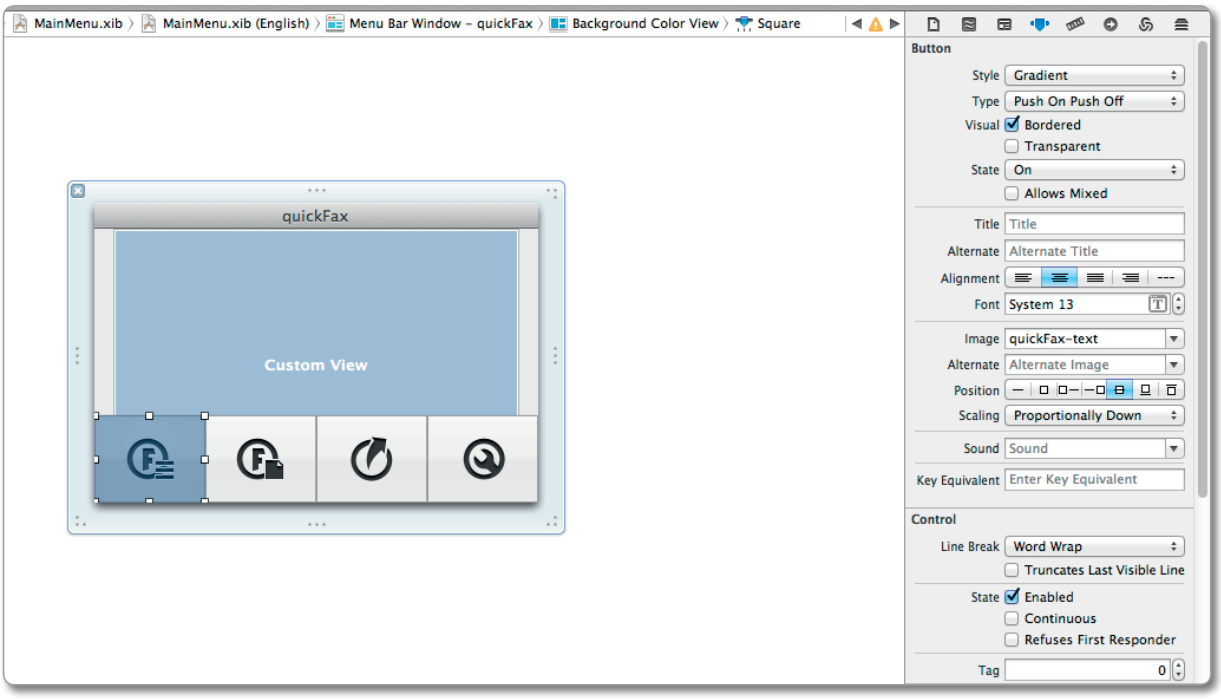
Il progetto quickFax per Mac prevede che i pulsanti (precedentemente aggiunti) per gestire il passaggio da una view all'altra visualizzino un'immagine al loro interno anziché un testo.
Ad ogni pulsante sono associate due immagini: una relativa allo stato attivo del pulsante e un'altra che viene visualizzata quando il pulsante risulta essere deselezionato.
Per visualizzare un'immagine all'interno di un pulsante è necessario prima di tutto importare le immagini all'interno del progetto mediante il percorso File > Add File to "quickFax".
A questo punto si apre una finestra di dialogo con la quale è possibile selezionare i file da aggiungere al nostro progetto.
Scegliendo l'opzione "Copy items into destination group's folder" verrà eseguita una copia locale dei file selezionati all'interno della cartella di progetto di XCode (definita al momento di creazione del progetto).

Nota: per una maggiore organizzazione è consigliabile raggruppare file dello stesso tipo (come ad esempio le immagini) all'interno di gruppi in modo tale da evitare la presenza di file "sparsi" all'interno del progetto.



Per associare un'immagine ad un pulsante è necessario selezionare il file MainMenu.xib e di conseguenza il pulsante desiderato.
Nell'Inspector section, selezionando Attributes inspector, è possibile modificare molti attributi e proprietà legate all'oggetto grafico selezionato (nel caso specifico un pulsante).
Alla voce Image basta scrivere il nome dell'immagine precedentemente importata nel progetto ed essa verrà visualizzata all'interno del pulsante.

In figura sono mostrate le immagini di default dei pulsanti, relative allo stato off (deselezionato).



In questo modo l'immagine visualizzata all'interno del pulsante sarà sempre la stessa mentre ai fini dei requisiti del progetto è necessario che l'immagine associata ad un pulsante cambi nel momento in cui l'utente preme su di esso.
Per fare ciò è necessario agire sulle immagini associate ai pulsanti a livello di codice.
E' stato creato un nuovo metodo all'interno della classe QFAppDelegate chiamato `toggleButtonImage:` il quale riceve come parametro un pulsante di tipo NSButton.
Nel momento in cui l'utente preme su un pulsante viene richiamato il metodo `switchViewAction:` che si occupa, come precedentemente descritto, di visualizzare la view associata al pulsante selezionato. All'interno di tale metodo viene richiamato `toggleButtonImage:` al quale viene passato come parametro il pulsante premuto.
Il metodo `toggleButtonImage:` imposta l'immagine relativa allo stato on (abilitato) per il pulsante selezionato, mentre tutti gli altri pulsanti mostrano l'immagine di default, quella relativa allo stato off del pulsante.

Esempio selezione pulsante

In modo analogo viene impostata l'immagine che viene visualizzata dall'applicazione nella *menu bar*. Una volta importata l'immagine desiderata all'interno del progetto, la classe OBMenuBarWindow (utilizzata per gestire la finestra dell'applicazione come una menu bar descritta precedentemente), mette a disposizione la proprietà *menuBarIcon* (dichiarata come @property nella sezione @interface della classe OBMenuBarWindow) che serve per definire l'immagine dell'icona che dovrà essere visualizzata nella *menu bar* quando l'applicazione viene eseguita.

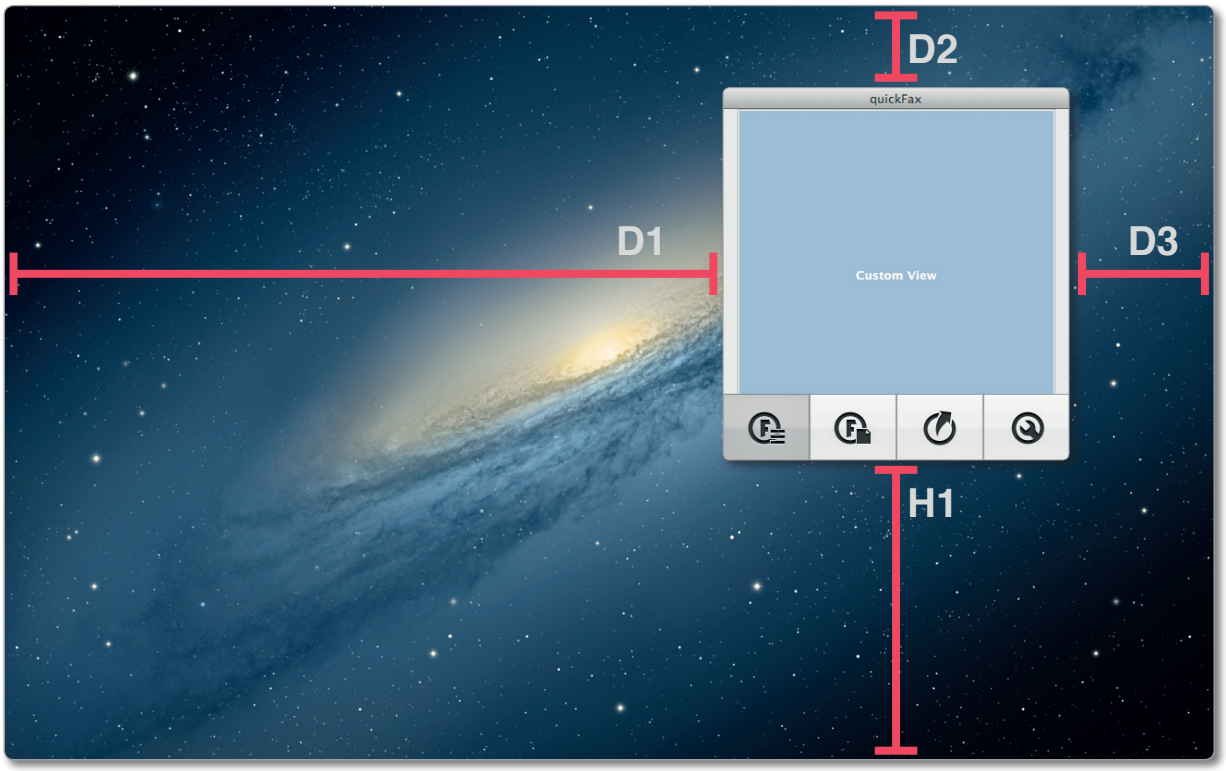
2.4 Switching di views: algoritmo per la gestione del passaggio da una vista all'altra

Prima di procedere alla spiegazione della realizzazione delle varie viste è necessario approfondire il tema riguardante il passaggio da una view all'altra. E' stata realizzata una prima applicazione di base nella quale sono presenti quattro pulsanti, nel momento in cui un utente preme un pulsante viene visualizzata la view desiderata (ovvero quella associata a livello logico al pulsante selezionato).
Questo semplice sistema consente il passaggio da una view all'altra solo nell'ipotesi in cui tutte le viste abbiano la stessa dimensione.
Nel caso specifico però ogni view ha la stessa larghezza ma altezza variabile, pertanto è necessario elaborare un algoritmo che consenta il passaggio da una view all'altra, ridimensionando automaticamente la finestra e la *mainView* presente nel MainMenu.xib, a seconda della view che si deve mostrare.

Le fasi che l'algoritmo dovrà gestire sono dunque le seguenti:

- Nascondere la view attualmente visualizzata
- Calcolare, sulla base della dimensione della view che deve essere visualizzata, se l'altezza della finestra deve aumentare o diminuire
- Ridimensionare la finestra e la *mainView* sulla base del calcolo precedente
- Visualizzare la nuova view all'interno della *mainView*

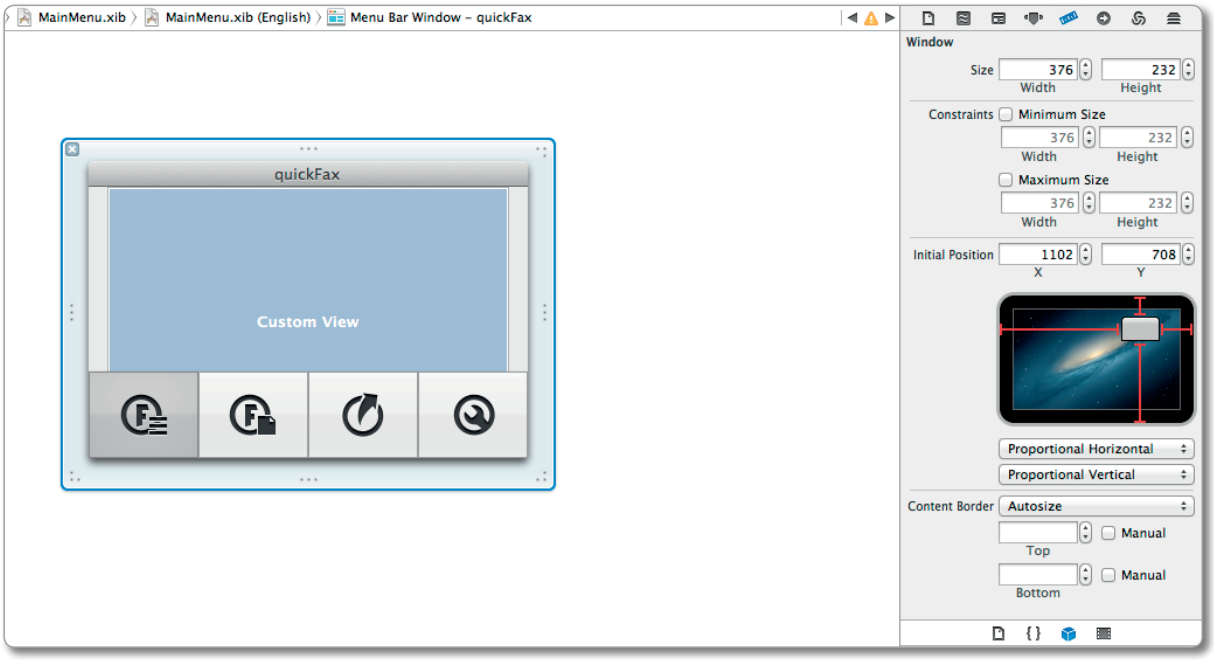
Innanzitutto è necessario definire le dimensioni che devono rimanere fisse.
Dalla figura possiamo vedere che le dimensioni D1, D2 e D3 devono rimanere fisse, in modo da assicurare che la posizione della finestra resti fissa all'interno dello schermo. Solo H1 deve essere modificata, in modo da aumentare o diminuire l'altezza della finestra per adattarsi alla nuova vista.



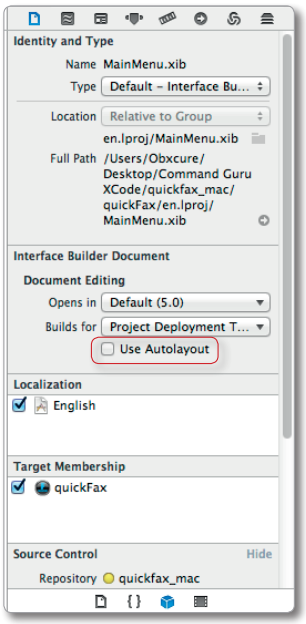
Per capire come realizzare tali specifiche è utile approfondire le modalità con cui vengono disegnate le finestre su Mac, in relazione con lo strumento Interface Builder.
In generale in Mac OS X tutte le finestre appartengono alla classe NSWindow (o ad una classe che eredita da essa come nel caso di OBMenuBarWindow); tale classe infatti definisce un oggetto che controlla le dimensioni della finestra, le sue coordinate e permette di mostrarla sullo schermo. Le due principali funzioni di un oggetto NSWindow sono:

- fornire un'area all'interno della quale un oggetto NSView può essere disegnato e in cui inserire altri oggetti
- controllare gli eventi che si verificano durante il ciclo di vita di una applicazione.

Un oggetto NSWindow è definito da un rettangolo chiamato *frame* che include tutta la finestra, compresa la barra del titolo e i bordi.
Tramite lo strumento Interface Builder, nella sezione Size inspector è possibile modificare le dimensioni di altezza e larghezza, definire dimensioni minime e massime per la finestra oltre che alla posizione iniziale della finestra stessa.

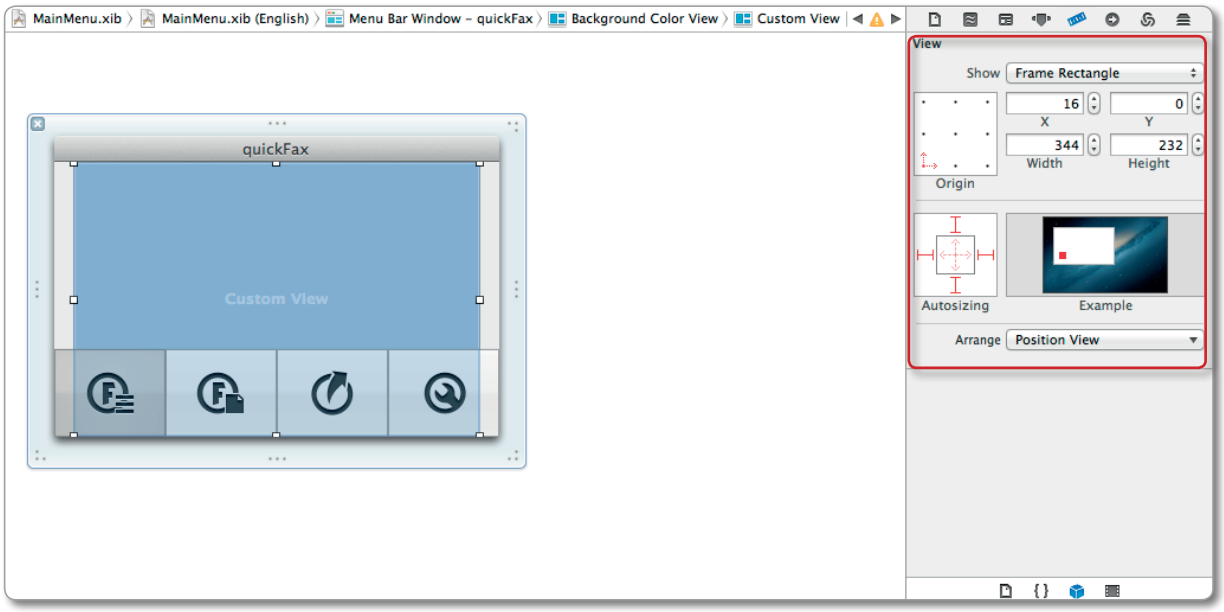


E' importante evidenziare che di default XCode gestisce in modo automatico la posizione e il layout di tutti gli oggetti grafici definiti mediante Interface Builder siano essi finestre, viste bottoni o altro.



Dato che dovremo modificare a livello di codice l'altezza della finestra e della *mainView* presenti nel file MainMenu.xib, è necessario innanzitutto disattivare la funzione di AutoLayout; ciò è possibile selezionando il file MainMenu.xib e nella Inspector section di Interface Builder, alla voce File inspector, disabilitare tale caratteristica, come mostrato a fianco.

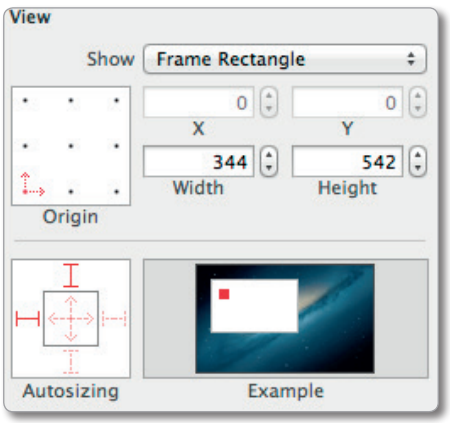
Ora selezionando la *mainView* nel file MainMenu.xib, la sezione Size inspector sarà simile a quella presente in figura:



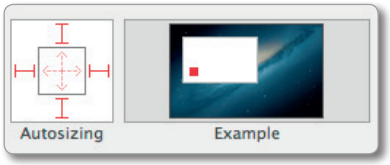
Il riquadro presente in figura permette di definire quali dimensioni (relative all'oggetto selezionato, in questo caso la mainView) devono rimanere fisse e quali possono essere modificate, quando la finestra viene ridimensionata.
Le frecce all'interno del riquadro sono utilizzate per definire il modo in cui un oggetto sarà ridimensionato; gli ancoraggi più esterni definiscono le dimensioni che devono rimanere fisse fra l'oggetto e uno dei bordi.

E' possibile vedere come l'elemento selezionato si sposterà rispetto al suo riquadro che lo contiene mediante la finestra di anteprima; portando il mouse su di essa viene visualizzato come avviene il ridimensionamento dell'elemento selezionato (evidenziato in rosso) quando il riquadro che lo contiene o *superview* (evidenziato in bianco) viene ridimensionato.

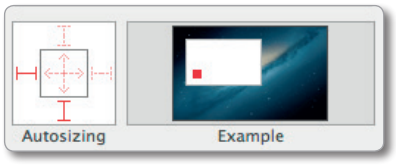
E' inoltre presente un riquadro dove è possibile definire la posizione dell'origine a partire dalla quale verrà disegnato l'oggetto grafico in questione; di default l'origine è fissata a partire dall'angolo in basso a sinistra.
In modo tale da standardizzare la posizione dell'origine delle viste e il modo in cui esse vengono ridimensionate quando variano le dimensioni della window si è deciso, per tutte le viste create (ovvero quelle relative a invio fax di solo testo, invio fax con allegato, fax recenti e impostazioni) di fissare gli ancoraggi superiore e sinistro (in accordo con la seguente figura) e di definire l'origine nell'angolo in basso a sinistra.



Gli ancoraggi che devono essere fissati per la *mainView* nel file MainMenu.xib risultano essere differenti; in questo caso tutti gli ancoraggi devono essere fissati in modo tale da definire univocamente la posizione della *mainView* rispetto alla window, nel momento in cui quest'ultima sarà ridimensionata.



Un controllo diverso deve essere fatto per i pulsanti presenti nella view del MainMenu.xib per i quali il Size Inspector si presenta come quello in figura:



Fissando gli ancoraggi sinistro e basso si definisce che nel momento in cui la window verrà ridimensionata in altezza i pulsanti resteranno fissi nella parte inferiore della window; senza questo controllo l'algoritmo avrebbe dovuto gestire oltre al ridimensionamento della window e visualizzazione della nuova view anche il riposizionamento dei pulsanti; in questo modo invece il loro posizionamento è gestito in modo automatico.

Entrando nel merito della realizzazione dell'algoritmo la prima azione che l'algoritmo deve eseguire è nascondere la view selezionata.
Ciò è già gestito dal precedente metodo `switchViewAction:` il quale rende visibile la view e nasconde tutte le altre sulla base del pulsante premuto dall'utente.
Ora è necessario gestire il ridimensionamento della window e della *mainView*; in primo luogo si crea un nuovo metodo chiamato `resizeWindow:` al quale viene passato un oggetto di tipo NSRect chiamato newFrameSize che conterrà le dimensioni della nuova view che deve essere mostrata a video.
La classe NSRect contiene tutte le informazioni fondamentali (altezza, larghezza e posizioni x,y dell'origine) per poter disegnare un rettangolo a video; è possibile salvare e impostare il *frame* di un qualsiasi oggetto grafico (comprese le view) mediante un oggetto di tipo NSRect.
Si deve dunque definire un oggetto di tipo NSRect chiamato *actualFrameSize*, il quale conterrà la dimensione della view attualmente visualizzata (definito nella sezione @interface della classe QFAppDelegate).Tale oggetto è necessario per poter confrontare l'altezza della view attualmente visualizzata con quella che deve essere visualizzata successivamente.

Nota: di default si inizializza l'oggetto actualFrameSize con le dimensioni relative alla view che si presenta all'utente nel momento in cui esegue l'applicazione, quindi quelle relative alla view di Invio Fax di Solo Testo.

Il metodo `resizeWindow:` definisce di quanto deve aumentare o diminuire l'altezza della finestra; ciò è possibile proprio confrontando l'altezza dell'oggetto *actualFrameSize* (contente le dimensioni della view attualmente visualizzata) con l'altezza dell'oggetto *newFrameSize* (contente le dimensioni della view che deve essere mostrata a video).
Se l'altezza della view attuale è minore dell'altezza della view che deve essere mostrata allora l'altezza della finestra deve aumentare, in caso contrario l'altezza della finestra

dovrà essere ridotta.
Ora è necessario calcolare l'offset, ovvero il modulo della differenza fra le altezze delle due view; l'offset definisce esattamente la quantità numerica che dovrà essere sommata o sottratta all'altezza della finestra.
Infine si calcolano le nuove dimensioni della finestra a seconda che l'altezza della finestra debba aumentare o diminuire; nel caso in cui l'altezza della finestra debba aumentare si trasla in basso la coordinata y di una quantità pari all'offset e si somma all'altezza della finestra l'offset stesso; nel caso invece in cui l'altezza della finestra debba diminuire si trasla in alto la coordinata y di una quantità pari all'offset e si sottrae l'offset all'altezza della finestra.
Una volta effettuato il calcolo delle nuove dimensioni della finestra, si esegue l'animazione che mostra il ridimensionamento della finestra stessa.
Terminato il ridimensionamento è necessario ridefinire la *mainView* in modo che le sue dimensioni siano coerenti con quelle della view visualizzata.
Si definisce un metodo `resizeMainView:` al quale viene passato un oggetto di tipo NSRect chiamato *newFrameSize*; come per il metodo precedente l'oggetto *newFrameSize* conterrà esattamente le dimensioni della view che dovrà essere visualizzata a video.Tale metodo si occupa esclusivamente di assegnare le dimensioni dell'oggetto *newFrameSize* alla *mainView* e di eseguire un refresh della view in modo tale da ridisegnarla a video con le dimensioni aggiornate.

Nota: è necessario aggiornare le dimensioni dell'oggetto actualFrameSize, in tale modo esso conterrà sempre le dimensioni della view attualmente visualizzata a video.

Infine deve essere presentata a video la nuova view; si è deciso di procedere mediante un'animazione di *fade-in*, la quale consiste in una dissolvenza in entrata della view stessa. Per eseguire tale animazione è necessario agire sul parametro Alpha della view, il quale definisce la trasparenza o l'opacità della stessa.
Se si imposta il valore Alpha di una view pari a 0 la view risulterà completamente trasparente mentre impostandone il valore a 1 la view risulterà completamente visibile; agendo su tali parametri è possibile ottenere l'effetto desiderato.

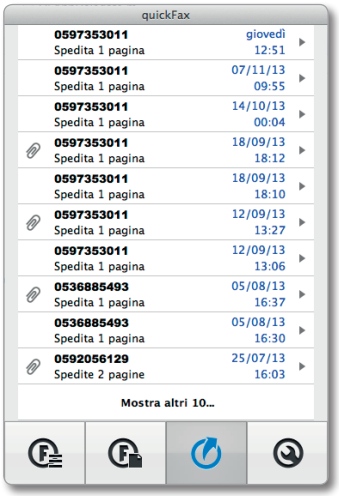
Nota: è all'interno del metodo switchViewAction: che, nel momento in cui si definisce quale view dovrà essere presentata a video, si imposta a 0 il valore Aplha di tale view.

Si definisce un metodo `fadeInNewView:` il quale, sulla base della view che deve essere presentata a video, si effettua un'animazione durante la quale il valore Alpha della view viene fatto passare dal valore 0 al valore 1 in un certo tempo t.
Le figure seguenti mostrano le varie fasi che si susseguono nel passaggio da una view all'altra nel caso in cui l'altezza della finestra debba aumentare.

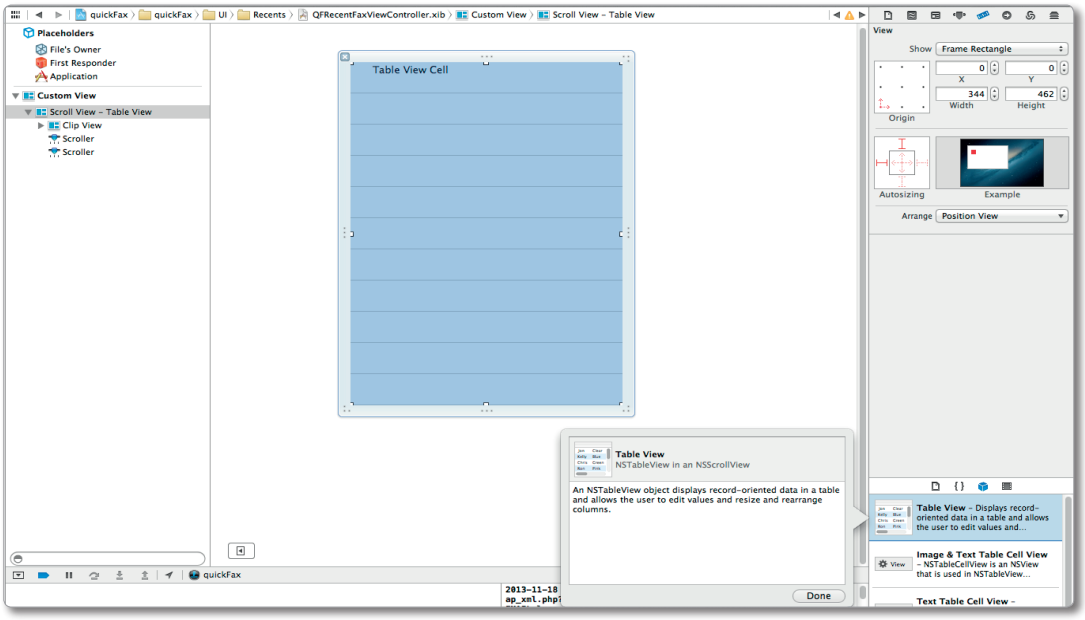
Capitolo 3

La view dei fax recenti

La view dei fax recenti permette all'utente di visionare lo storico dei fax da lui inviati, ordinati in ordine cronologico inverso, partendo dal più recente; la vista mostra i dieci fax più recenti inviati organizzati in forma tabellare: ad ogni riga della tabella corrisponde un fax inviato nella quale vengono mostrate le informazioni fondamentali quali numero di fax, riscontro sull'invio del fax, data, ora e se è presente o meno un file allegato.
Mediante la pressione del pulsante "Mostra altri 10…" l'utente è in grado di scorrere all'indietro lo storico dei fax inviati, a gruppi di dieci.



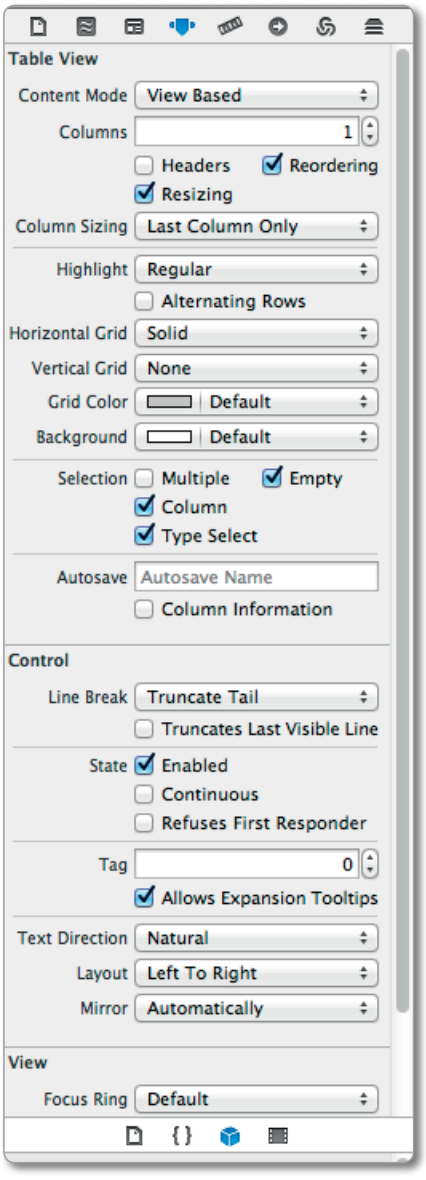
Se l'utente effettua un click su uno dei fax recenti corrispondente ad un riga della tabella, viene mostrata una nuova vista nella quale sono presenti i dettagli relativi al fax inviato; in essa vengono mostrate informazioni aggiuntive sul destinatario (nome, cognome, numero di fax e foto) e l'anteprima del fax inviato con la possibilità di salvare o inviare per mail tale documento.
Per realizzare tale vista è necessario agire sulla classe QFRecentFaxViewController sottoclasse di NSViewController (tale classe è già stata creata durante la realizzazione dell'interfaccia grafica di base); innanzitutto si dispone nel file .xib della classe l'elemento grafico fondamentale che costituisce il fulcro della vista: la *Table View*.



Esempio di utilizzo dell'oggetto TableView in Interface Builder

3.1 La Table View

Le tabelle sono uno degli elementi fondamentali nella realizzazione di interfacce grafiche e con esse è possibile mostrare all'utente in maniera ordinata dei record. [4]
Ogni riga della tabella solitamente rappresenta un record, mentre le colonne rappresentano gli attributi.
Come esempio si può pensare ad una tabella la quale raccolga dati relativi a differenti modelli di auto; in tale caso ad ogni riga della tabella sarà associato un modello di un'auto, mentre le colonne rappresentano le sue caratteristiche (ad esempio marca, cilindrata e prezzo).
In questo caso il metodo più semplice per rappresentare i dati relativi ai fax inviati da un utente è quello di raggruppare i dati in una tabella; ad ogni riga della tabella corrisponderà uno ed un solo fax inviato.
E' necessario ora analizzare il file .xib, il quale si presenta come nella figura precedente.
Di default, quando inseriamo una *Table View* all'interno di un file .xib, vengono create due colonne e in ognuna di esse vengono create delle celle di tipo testuale.
Ora è necessario modificare le impostazioni relative alla tabella in modo tale da adattarla al caso in esame; si selezioni la *Table View* e si visualizzi l'Attributes inspector, nella barra destra di XCode.



La prima opzione Content Mode permette di scegliere tra Cell Based e View Based; l'opzione View Based consente di disegnare le celle personalizzate direttamente da Interface Builder, un notevole vantaggio in termini di flessibilità per lo sviluppatore.
L'opzione Columns consente di scegliere il numero di colonne della tabella.
Sono poi presenti tre scelte:

- Headers: mostra le informazioni di intestazione per ogni cella
- Reordering: permette di riordinare le colonne
- Resizing: permette di scegliere dinamicamente la larghezza delle colonne

L'opzione Column Sizing definisce come deve essere impostata la larghezza delle colonne; Highlight definisce se è possibile o meno da parte dell'utente evidenziare il contenuto della cella.
L'opzione Alternating Rows, se abilitata, colora le celle in maniera sfalsata.

Nota: l'opzione Alternating Rows risulta molto utile nel caso di tabelle di grandi dimensioni per permettere una lettura più agevole.

Horizontal Grid e Vertical Grid permettono di definire i bordi rispettivamente delle righe e delle colonne;è inoltre possibile specificare il colore dei bordi mediante l'opzione Grid Color.

Le opzioni relative alla selezione (Selection) permettono di scegliere in quali modi diversi l'utente può selezionare parti della tabella. Nel caso specifico la tabella deve presentare una sola colonna, con possibilità di selezionare solamente una riga per volta e le righe devono essere visualizzate in modalità non alternata. Definite tutte le opzioni della tabella è necessario creare l'IBOutlet per la classe a cui essa appartiene, ovvero NSTableView; mediante tale outlet, chiamato in riferimento al codice *recentFaxTableView*, è ora possibile modificare la tabella tramite codice.

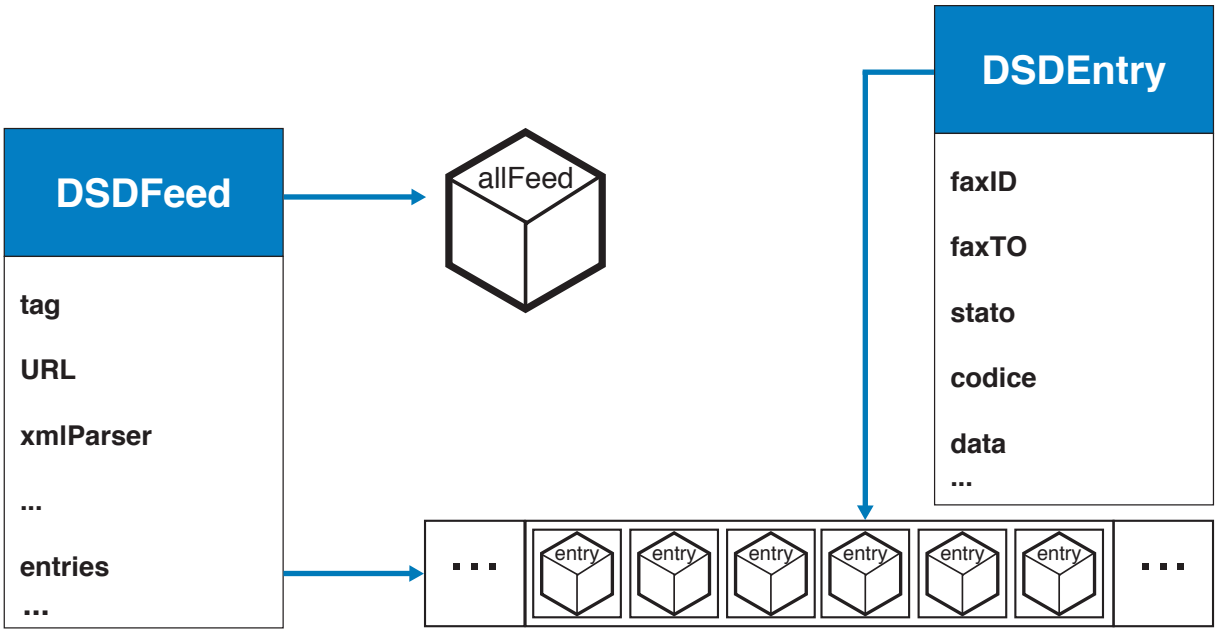
Nota: E' importante assicurarsi di aver creato l'outlet dell'oggetto appartenente alla classe NSTableView e non NSScrollView.

Le funzionalità che la tabella dovrà presentare sono:

- Visualizzazione dei dati relativi ai fax recenti
- Mostrare inizialmente un numero di fax pari a dieci
- Possibilità per l'utente di premere su uno dei fax recenti, mostrando una vista di dettaglio
- Possibilità per l'utente di premere su un pulsante "Mostra altri 10…" a seguito del quale vengono aggiunte dieci righe alla tabella corrispondenti ad altri dieci fax recenti.

Nota: il pulsante "Mostra altri 10…" è in realtà una particolare riga della tabella e non un pulsante di tipo NSButton.

Prima di poter popolare la tabella è necessario definire quali sia la struttura dati dalla quale verranno prese le informazioni da visualizzare in tabella. In questo caso la struttura dati è un'istanza della classe DSDFeed (la quale fa parte delle librerie API fornitemi dall'azienda per poter comunicare correttamente con il servizio Faxator), chiamata in riferimento al codice *allFeed*. All'interno della classe DSDFeed viene definito un array ordinato, denominato *entries*, i cui elementi sono istanze della classe DSDEntry, la quale contiene tutte le informazioni fondamentali associate ad un fax (come ad esempio ID, data, ora stato, numero di pagine inviate e descrizione).
In breve ogni elemento dell'array [allFeed entries] corrisponde ad un fax inviato dall'utente e quindi ad una riga della *TableView*, la quale sarà popolata con le informazioni contenute all'interno di tale elemento (istanza della classe DSDFeed).

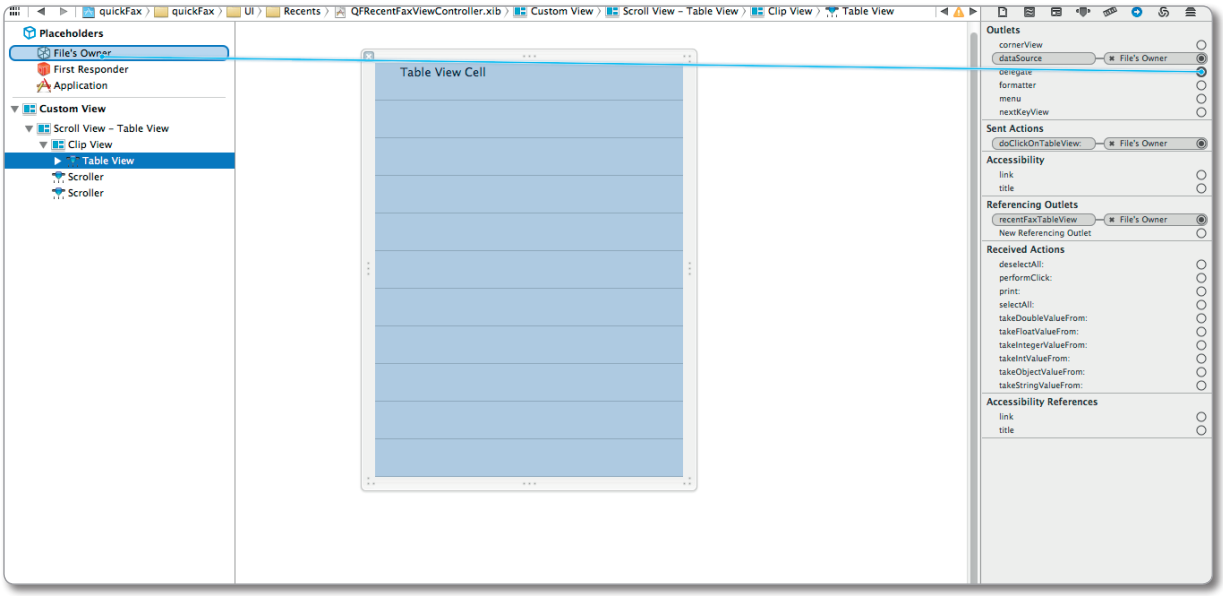


Compito fondamentale è stato quello di richiamare correttamente i metodi appartenenti alle librerie API sopra citate in modo da definire il numero di fax inviati dall'utente da richiedere al server (ovvero il numero di elementi dell'array [allFeed entries]).
All'interno del metodo inizializzatore della classe viene impostato il numero di fax recenti da richiedere al server (definito dalla variabile downloadCount) e viene richiamato il metodo `loadRequest:`, il quale si occupa di popolare correttamente la struttura suddetta richiedendo al server un numero di fax pari a *downloadCount*.

Nota: di default viene impostato downloadCount pari a dieci in quanto si è deciso in fase di progetto di visualizzare un numero di fax recenti multiplo di dieci. La variabile downloadCount viene incrementata di un valore pari a dieci nel momento in cui l'utente preme sul pulsante "Mostra altri 10…" .

3.2 Popolare la tabella

Per popolare la tabella è innanzitutto necessario impostare quale è la classe che fornisce i dati (*dataSource*) e quale risponde alle chiamate (*delegate*) della tabella.
Per fare ciò è sufficiente selezionare la *Table View* nel file . , visualizzare il Connections inspector e creare un collegamento (come mostrato in figura) fra *delegate* e File's Owner e *dataSource* e File's Owner.



In questo modo si definisce che sarà la classe stessa (ovvero QFRecentFaxViewController) a rispondere del *delegate* e del *dataSource,* fornendo i dati con i quali popolare la tabella e implementando i metodi fondamentali per il suo corretto funzionamento. Ora è necessario implementare i due metodi fondamentali che consentono di popolare la tabella.
Il primo metodo da definire è `numberOfRowsInTableView:` il quale ritorna il numero di righe della tabella; si imposta come valore di ritorno il numero di elementi dell'array [allFeed entries] più uno.

Nota: la riga in più è quella che consente all'utente di mostrare altri 10 fax recenti, se l'utente effettua un click su di essa.

Il secondo metodo è `tableView:viewForTableColumn:row:` il quale si occupa di gestire il contenuto della cella. Tale metodo ritorna un oggetto di tipo NSView il quale andrà a costituire la vista della cella, ovvero ciò che verrà rappresentato all'interno della cella della tabella.

Il metodo si occupa in principio di istanziare le due classi fondamentali, CellRowViewController e CellRowView, per l'assegnazione di una view personalizzata ad ogni riga della tabella; successivamente, utilizzando l'indice di riga della tabella, si accede all'elemento corrispondente nell'array [allFeed entries] e si visualizzano le informazioni relative al fax inviato dall'utente in apposite etichette (*label*).
Le informazioni che vengono visualizzate sono:

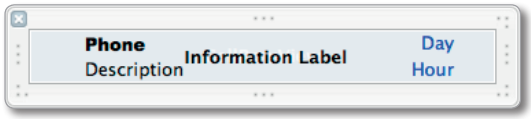
- Numero di fax
- Data
- Ora
- Descrizione sullo stato del fax

L'algoritmo gestisce inoltre i casi particolari in cui non ci siano fax recenti da visualizzare (ad esempio se l'utente si è appena registrato e non ha ancora inviato nessun fax) oppure il caso in cui i fax recenti da visualizzare siano minori di downloadCount.

Nota: quando l'indice della riga coincide con il valore della variabile downloadCount allora è necessario mostrare la riga "Mostra altri 10…".

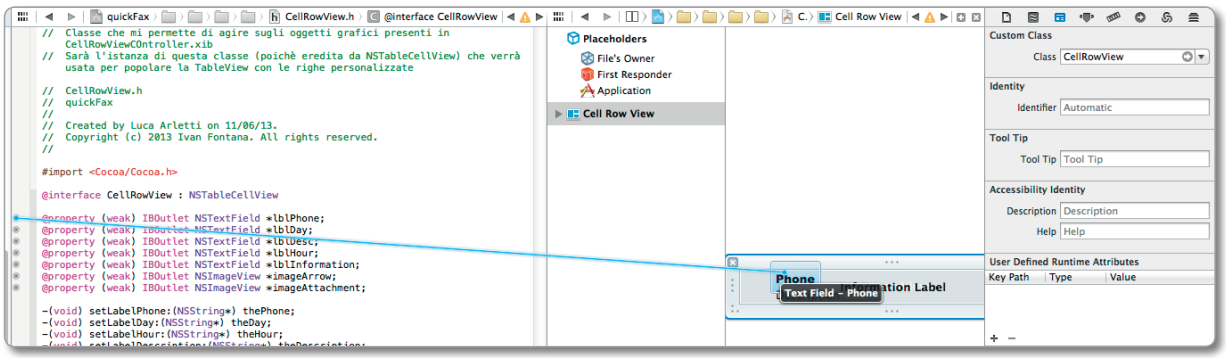
3.3 Cella personalizzata

A questo punto si deve descrivere come realizzare una cella personalizzata della tabella: è necessario creare una nuova classe che eredita da NSViewController, chiamata CellRowViewController e disporre una serie di oggetti label come in figura:



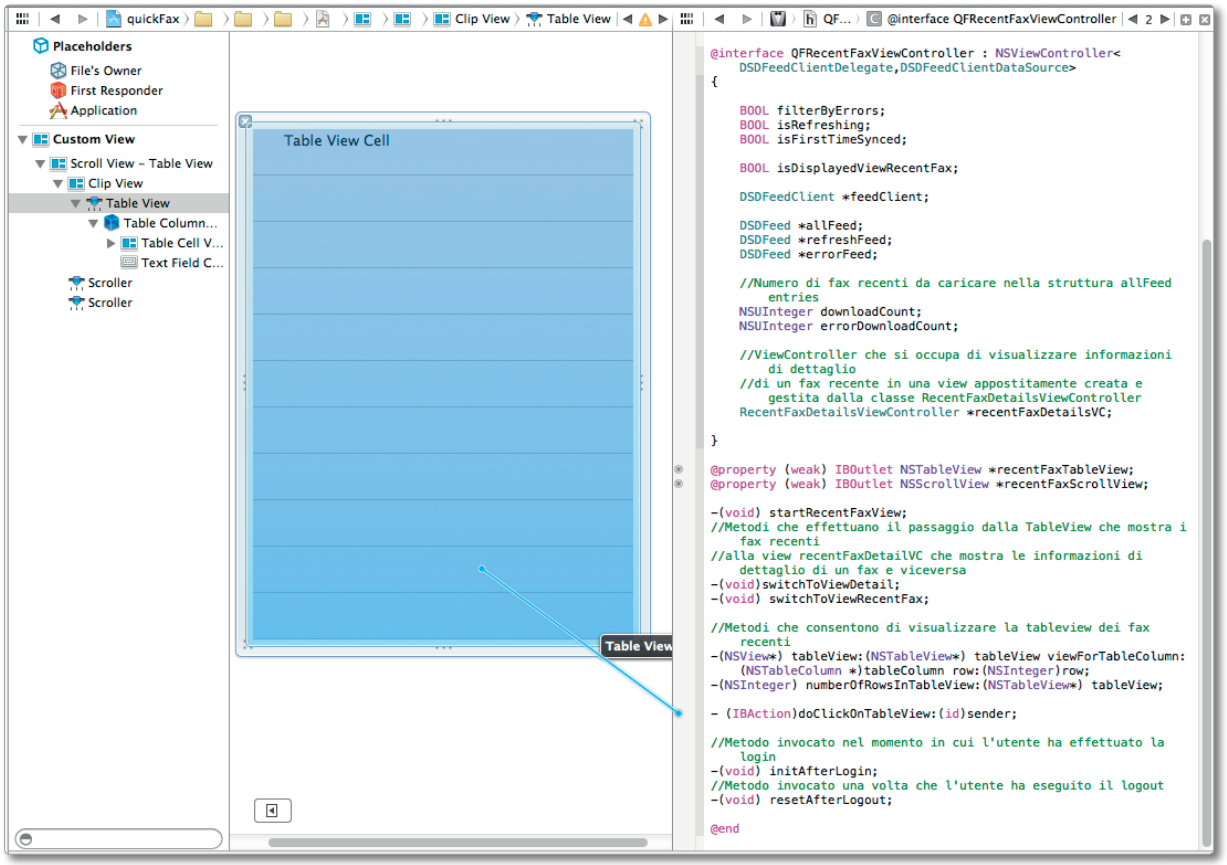
Dato che la vista creata graficamente nel file .xib della classe suddetta deve rappresentare una cella della tabella, si crea una nuova classe chiamata CellRowView la quale eredita da NSTableCellView, classe fondamentale per la gestione delle celle personalizzate di una *Table View*.
Per far sì che la vista disegnata nel file CellRowViewController.xib vada a costituire la vista di ognuna delle celle della *Table View* è necessario definire che la classe che gestisce la view presente nel file CellRowViewController.xib non è la classe standard NSView ma è la classe CellRowView appena creata, la quale eredita da NSTableCellView.

Per poter modificare a livello di codice il contenuto delle label precedentemente inserite creiamo i rispettivi IBOutlet, i quali devono essere definiti nel file header della classe CellRowView come riportato in figura:



3.4 Interazioni dell'utente

Le tabelle sono spesso utilizzate per mostrare informazioni all'utente, ma è importante anche permettere all'utente di interagire con esse.
Nel caso specifico è importante sapere quando un utente esegue un click su una riga della tabella e in tal caso sapere quale sia l'indice della riga sulla quale ha eseguito il click. E' necessario dunque creare un metodo IBAction, chiamato `doClickOnTableView:`, associato alla *Table View* come mostrato in figura:



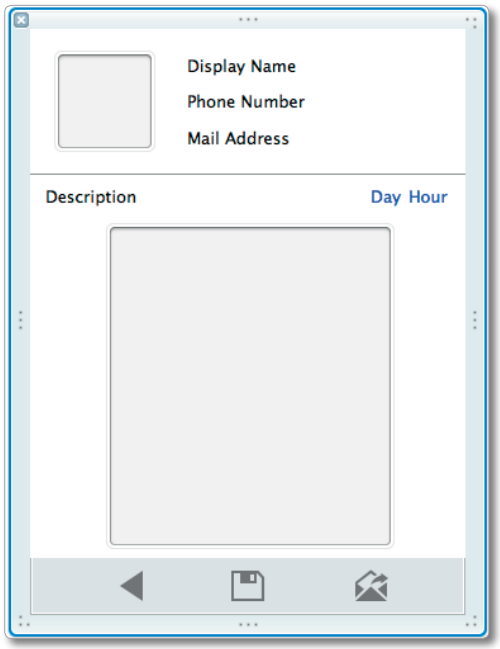
Tale metodo verrà invocato ogni qual volta l'utente effettua un click su una riga della tabella.
Al suo interno viene memorizzato l'indice della riga sulla quale l'utente ha premuto, e sulla base del valore di tale indice confrontato con il valore della variabile *downloadCount*, si eseguono due azioni differenti; distinguiamo perciò due casi:

1. Nel caso in cui l'utente abbia fatto click sula riga "Mostra altri 10…" (ciò significa che l'indice di riga risulta uguale al valore della variabile downloadCount) è necessario incrementare la variabile *downloadCount* di un valore pari a dieci; successivamente viene eseguita nuovamente la sincronizzazione con il server in modo da aggiungere alla *Table View* dieci righe, corrispondenti a dieci fax recenti inviati dall'utente

2. Nel caso in cui l'utente esegua un click su una delle righe della tabella corrispondenti ad un fax inviato, viene presentata una nuova vista che mostra ulteriori informazioni riguardanti il fax selezionato.

3.5 La vista di dettaglio

La vista di dettaglio mostra all'utente informazioni aggiuntive riguardanti il fax selezionato. In tale vista vengono riportate informazioni aggiuntive riguardanti il destinatario del fax quali nome, cognome, numero di fax e indirizzo email (se presenti).
Viene inoltre mostrata un'anteprima del fax inviato, la quale viene visualizzata in un apposito riquadro, ovvero un oggetto *ImageWell* (presente nella Library Section di Interface Builder) appartenente alla classe NSImage la quale fornisce i metodi fondamentali per la rappresentazione di un'immagine; è inoltre possibile salvare o inviare per email tale documento mediante l'utilizzo di appositi pulsanti situati nella parte inferiore della vista.
La vista di dettaglio è stata realizzata mediante la creazione di una sottoclasse della classe NSViewController chiamata RecentFaxDetailsViewController (in riferimento al codice).
In figura viene riportata la struttura del file .xib utilizzato per realizzare tale vista.



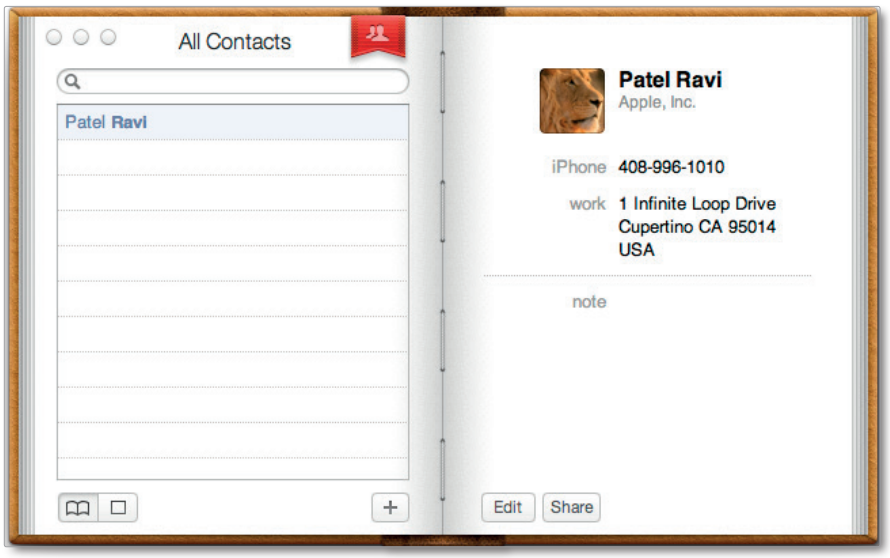
Passiamo ora ad analizzare le varie funzionalità implementate nella classe suddetta.

3.5.1 Addressbook e ricerca contatti

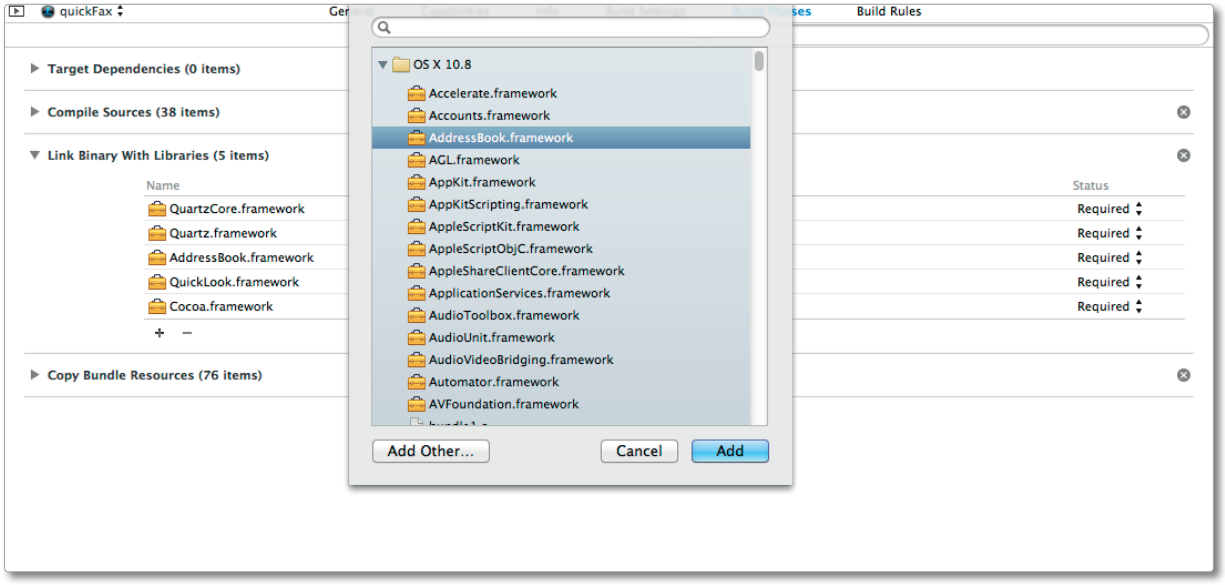
Dalla precedente analisi della struttura [allFeed entries] risulta che l'unico dato memorizzato riguardante il destinatario è il solo il numero di fax; risulta dunque necessario reperire le informazioni aggiuntive dalla rubrica "*Contatti*" presente nel sistema operativo Mac OS X, effettuando una ricerca sulla base del numero di fax.
Nel caso in cui la ricerca vada a buon fine vengono visualizzate le informazioni aggiuntive disponibili quali:

- Nome
- Cognome
- Indirizzo e-mail
- Foto associata al contatto

Nel caso invece in cui la ricerca non ottenga risultati viene visualizzato solamente il numero di fax del destinatario.



Per implementare tale funzionalità è necessario poter accedere alla rubrica "Contatti" del sistema operativo; risulta necessario l'utilizzo di uno specifico framework chiamato Address book, il quale contiene tutti i metodi e le istruzioni fondamentali per poter accedere e manipolare i contatti presenti nella rubrica. [7]
Per importare tale framework all'interno del progetto è necessario selezionare il progetto nel menu a sinistra "Project Navigator" e "Build Phases"; alla voce "Link Binary With Libraries" vengono mostrati i framework attualmente utilizzati nel progetto.
Mediante l'apposita icona "+" è possibile aggiungere framework al progetto selezionando fra quelli esistenti messi a disposizione dal sistema operativo Mac OS X.



E' ora possibile realizzare un algoritmo che sulla base del numero di fax del destinatario, ricerchi tale numero all'interno dei contatti presenti in rubrica.
Tale funzionalità viene implementata dal metodo `performSearch:` presente all'interno della classe RecentFaxDetailsViewController.
Nel momento in cui l'utente effettua un click su uno dei fax da lui inviati visualizzati nella *Table View*, prima di mostrare la view di dettaglio, viene invocato il metodo suddetto al quale viene passato il numero di fax del destinatario. Il metodo confronta tale numero con tutti i numeri presenti in rubrica: nel momento in cui la ricerca fornisce un esito positivo (ovvero il numero di fax del destinatario risulta essere uguale ad un numero memorizzato in rubrica) la ricerca viene interrotta (impostando un'opportuna variabile booleana) e vengono memorizzati i dati interessati associati al contatto trovato.

Le informazioni vengono memorizzate in un'istanza della classe Contact (sottoclasse di NSObject) il cui scopo è proprio quello di contenere le informazioni che dovranno essere visualizzate a video.
Nel caso in cui la ricerca abbia esito negativo allora l'istanza della classe Contact memorizzerà solamente il numero di fax del destinatario.
Al termine della ricerca viene comunque invocato il metodo displayContactInformation: , al quale viene passata l'istanza della classe Contact; tale metodo si occuperà di visualizzare opportunamente a video le informazioni memorizzate all'interno di tale istanza.

Nota: al termine della ricerca (qualunque sia il suo esito) vi è la certezza che l'istanza della classe Contact sarà stata creata e che conterrà almeno il numero di fax del destinatario.

Due metodi molto importanti che sono stati implementati all'interno della classe RecentFaxDetailsViewController sono:

```
- (IBAction)clickSaveButton:(id)sender;

- (IBAction)clickMailButton:(id)sender;
```

Il primo metodo viene invocato mediante la pressione del pulsante  e consente all'utente di salvare il fax inviato come documento pdf. Viene aperto un opportuno pannello di salvataggio mediante il quale l'utente può decidere il nome del file e il percorso dove salvarlo. Al termine del salvataggio per confermare che l'operazione è andata a buon fine il file viene aperto e mostrato all'utente.
Il secondo metodo invocato mediante la pressione di  consente invece all'utente di inviare il documento di fax direttamente come allegato di posta elettronica. Il metodo si occupa di aprire l'applicazione Mail (integrata in Mac OS X) e comporre un messaggio vuoto al quale viene allegato il documento suddetto.

3.5.2 Switching di views a scorrimento laterale

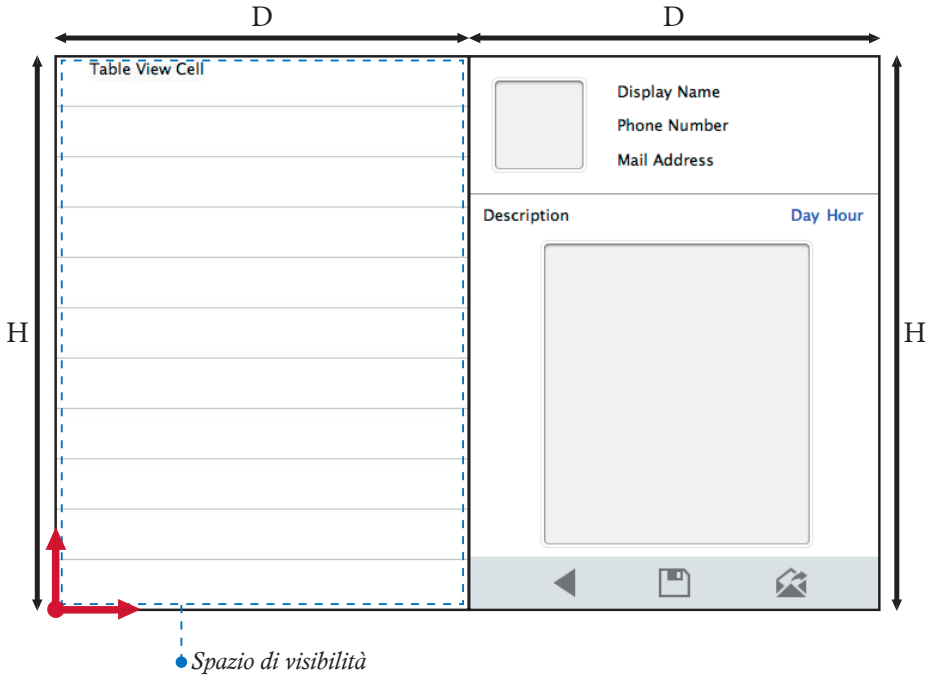
Nel momento in cui l'utente effettua un click sul fax inviato, il programma deve passare dalla visualizzazione della view dei fax recenti in tabella alla view che mostra i dettagli relativi al fax selezionato dall'utente; il modo migliore per effettuare tale passaggio, affinché risulti efficiente dal punto di vista visivo e di conseguenza gradito dall'utente, è quello di utilizzare un'animazione.
Un esempio di passaggio da una view all'altra mediante animazione è stato già trattato nel paragrafo 2.4, nel quale si è visto come modificare l'altezza della finestra dell'applicazione affinché si adatti alle dimensioni della nuova vista che deve essere mostrata all'utente.
Nel caso in esame, invece, le due viste hanno dimensioni che risultano essere fisse e uguali fra loro; in fase di progetto si è infatti deciso di realizzare la vista di dettaglio con le stesse dimensioni della vista dei fax recenti in quanto le dimensioni definite sono risultate sufficienti per costruire correttamente la view di dettaglio, così come mostrata nel paragrafo 3.5 (in modo tale che ogni oggetto grafico appartenente a tale view risulti sufficientemente visibile all'utente, senza ridurne troppo le dimensioni).
Inoltre l'interfaccia grafica risulta essere notevolmente pesante dal punto di vista visivo nel caso in cui l'applicazione esegua molto spesso lo stesso tipo di animazione, per tale motivo l'animazione è stata variata. Per quanto riguarda il tipo di animazione, si è scelto di utilizzare lo scorrimento orizzontale per il passaggio da una view all'altra, il quale risulta essere molto efficace nel caso in cui le viste abbiano la stessa dimensione.

Vediamo ora come realizzare nel dettaglio lo scorrimento orizzontale delle due viste.

Le fasi che l'algoritmo dovrà gestire sono le seguenti:

- Disporre la vista di dettaglio alla destra della view che mostra i fax recenti
- Traslare orizzontalmente le viste verso destra o verso sinistra a seconda che venga richiesto di mostrare l'una o l'altra vista.

Per meglio comprendere come l'algoritmo opera consideriamo un caso pratico: supponiamo che l'utente abbia avviato l'applicazione e abbia premuto sul pulsante per mostrare la view che mostra i suoi fax recentemente inviati.
Inizialmente dovrà risultare visibile solo tale vista, mentre la vista di dettaglio deve risultare nascosta. Affinché lo scorrimento avvenga in maniera corretta le due viste devono essere disposte una di fianco all'altra, come in figura successiva.

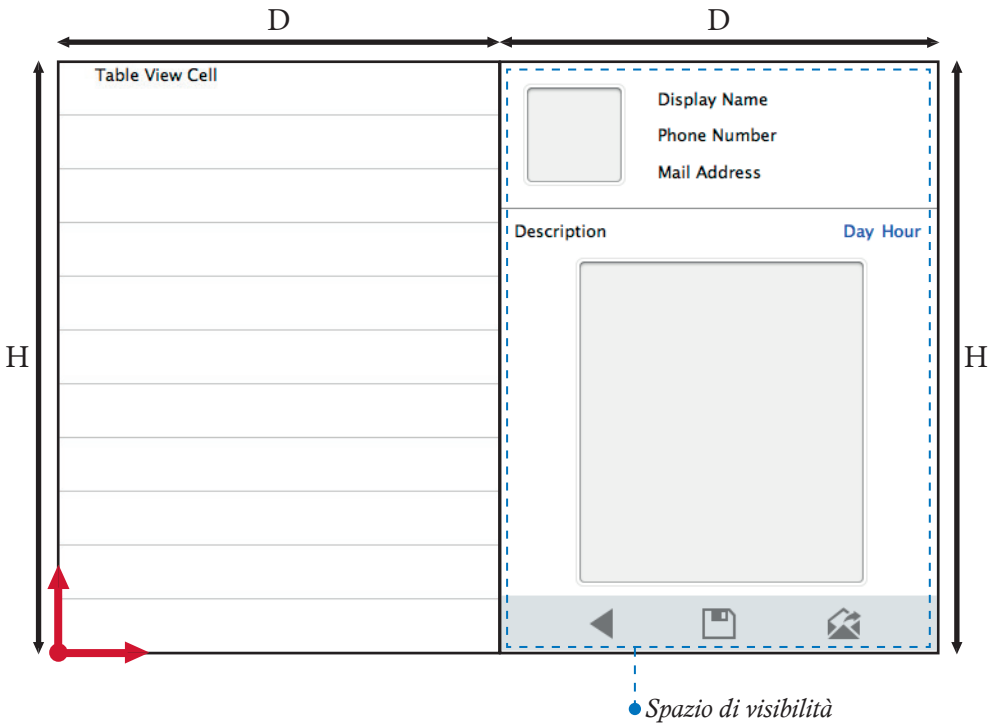


Dato che lo spazio visibile all'utente è solo DxH, la vista di dettaglio inizialmente disposta affiancata non risulta essere visibile.

Per fare ciò all'interno del metodo inizializzatore della classe QFRecentFaxViewController si crea l'istanza della classe RecentFaxDetailsViewController (chiamata in riferimento al codice recentFaxDetailsVC) e si impone che la coordinata x dell'origine della view *recentFaxDetailsVC* coincida con la larghezza D della view dei fax recenti.
In questo modo la view di dettaglio verrà disegnata alla destra della view dei fax recenti (come da figura sopra riportata).
A questo punto vengono definiti due metodi i quali si occupano di effettuare il passaggio da una view all'altra mediante scorrimento orizzontale:

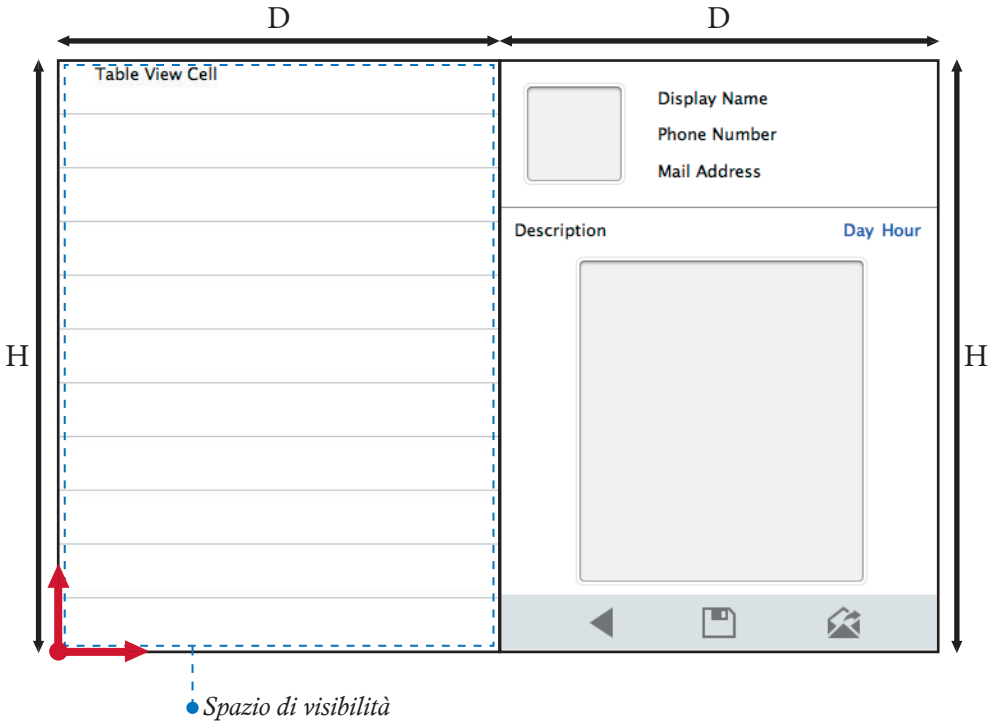
```
-(void) switchToViewDetail;
-(void) switchToViewRecentFax;
```

Nel momento in cui l'utente effettua un click su uno dei fax recenti visualizzati nella *Table View* viene eseguito il primo metodo il quale si occupa di traslare verso sinistra il rettangolo 2DxH di una quantità pari a D (creato ponendo le due viste una accanto all'altra come descritto sopra) rendendo così visibile solo la view di dettaglio mentre la view dei fax recenti esce dallo spazio di visibilità dell'utente.
La situazione sopra descritta è rappresentata in figura successiva:



Il rettangolo 2DxH viene traslato verso sinistra sottraendo alla coordinata x dell'origine globale una quantità pari a D.

Nel momento in cui l'utente preme sul pulsante  presente nella view di dettaglio, viene richiamato il secondo metodo che si occupa di traslare il rettangolo 2DxH verso destra di una quantità pari a D rendendo così visibile la view dei fax recenti e facendo uscire dallo spazio di visibilità la view di dettaglio, come rappresentato in figura:



Capitolo 4

Le viste di invio fax

In questo capitolo verranno analizzate le funzionalità di invio fax messe a disposizione dall'applicazione.

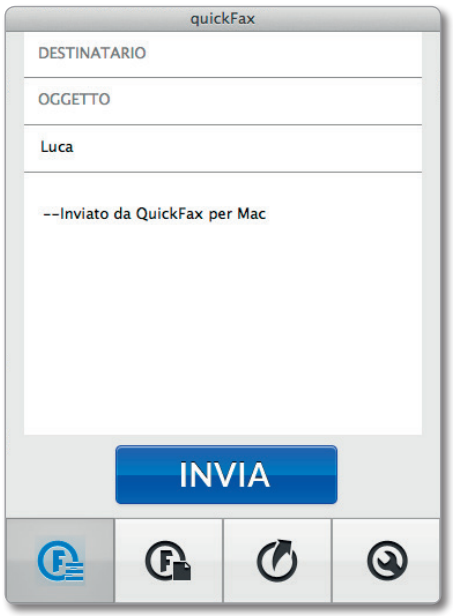
Vi sono due tipologie fondamentali di fax che l'utente può inviare:

- Fax contente un messaggio di solo testo
- Fax contente un allegato

Il primo tipo consente all'utente di inviare come fax un messaggio puramente testuale mentre il secondo tipo mette a disposizione dell'utente la funzionalità più avanzata di inviare come fax un documento, come ad esempio un'immagine. L'utente può selezionare un file presente nel proprio computer e, nel caso in cui il formato del file sia supportato dall'applicazione, può inviarlo come fax.
Al fine di realizzare le due funzionalità suddette sono state create due viste separate, una per l'invio di fax testuali e una che consente l'invio di fax con allegato.
Procediamo ora ad analizzare nel dettaglio il funzionamento delle due viste.

4.1 La view di invio fax di solo testo

La view di invio fax di solo testo è essenzialmente composta da quattro campi testuali, i quali devono essere opportunamente compilati dall'utente per poter inviare correttamente il fax.



I campi di testo sono i seguenti:

- **Destinatario**: il numero di fax del destinatario
- **Oggetto**: l'oggetto del fax
- **Mittente**: nome del mittente
- **Messaggio**: il corpo del fax

Nota: il campo mittente è compilato automaticamente con il nome dell'utente che ha effettuato il login sull'applicazione e non modificabile.
Il nome che compare è quello fornito dall'utente al momento della registrazione.

Nota: nel caso in cui nella vista di impostazioni l'utente abbia specificato una firma, il campo messaggio è compilato automaticamente con quest'ultima.

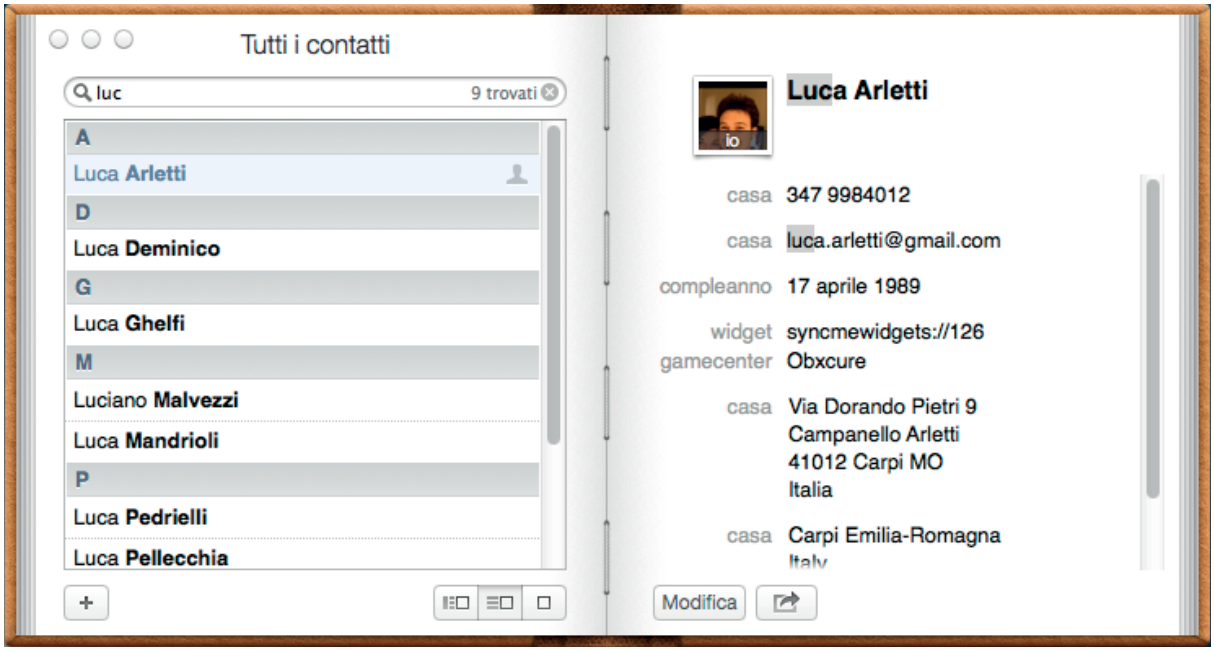
Una volta che l'utente ha compilato opportunamente tutti i campi può inviare il fax mediante la pressione sul pulsante "INVIA".
Nel momento in cui l'utente preme tale pulsante, l'applicazione controlla che tutti i campi siano stati correttamente compilati; in caso affermativo viene eseguita un'animazione durante la quale l'applicazione informa l'utente sullo stato di invio del fax (in elaborazione, trasmesso, non trasmesso). Nel caso in cui non siano riscontrati errori un messaggio informativo comunica all'utente che la richiesta di invio del fax è stata trasmessa con successo; l'utente riceverà una mail al proprio indirizzo di posta elettronica con la conferma di avvenuta spedizione del fax da parte del servizio Faxator.
Nel caso in cui invece si riscontri un errore durante la trasmissione del fax, un messaggio opportuno comunica all'utente la tipologia di errore riscontrato.

In figura è riporta la struttura del file .xib della classe QFSendFaxOnlyTextViewController (già creata durante la realizzazione dell'interfaccia grafica di base) la quale si occupa di gestire tutte le funzionalità messe a disposizione da tale vista.

Come già anticipato, la vista è stata realizzata inserendo nel file .xib quattro oggetti di tipo *Text Field* (per la realizzazione dei relativi campi di testo compilabili dall'utente). Sono presenti anche tre *label* necessarie per informare l'utente di possibili errori avvenuti durante la compilazione dei campi di testo.
Vediamo ora nel dettaglio le varie funzionalità implementate.

4.1.1 Filtraggio dei contatti della rubrica

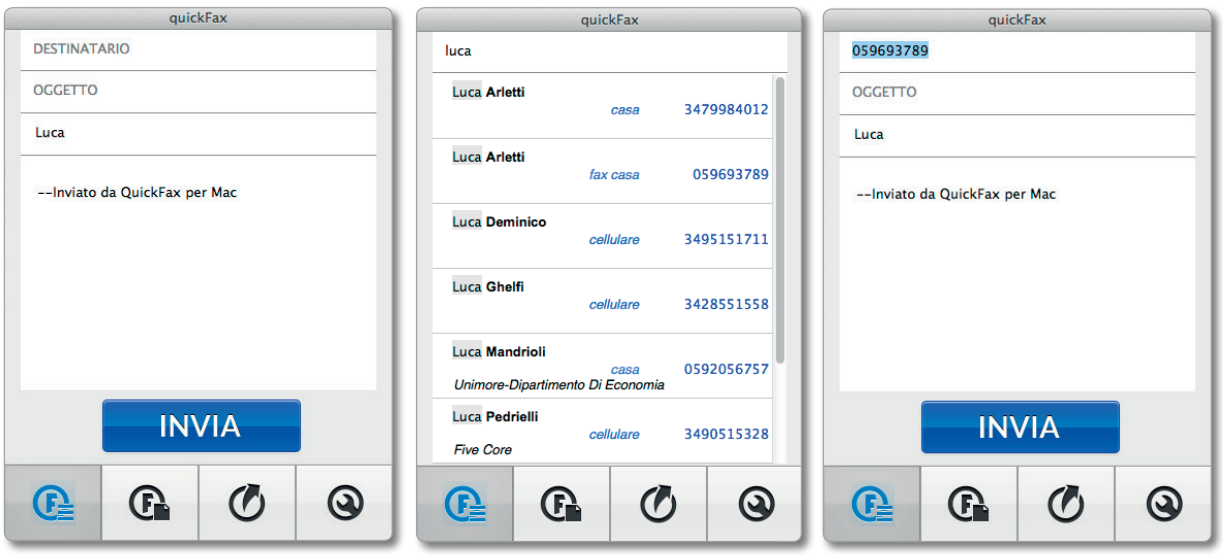
Una delle funzionalità più importanti presenti in tale vista riguarda l'autocompletamento che filtra i contatti presenti sulla rubrica dell'utente.
Si è tratta ispirazione dal modo con il quale la rubrica "*Contatti*" presente su Mac OS X filtra i record memorizzati sulla base dei caratteri digitati dall'utente, di conseguenza si è deciso di implementare una funzionalità analoga anche all'interno di tale vista.



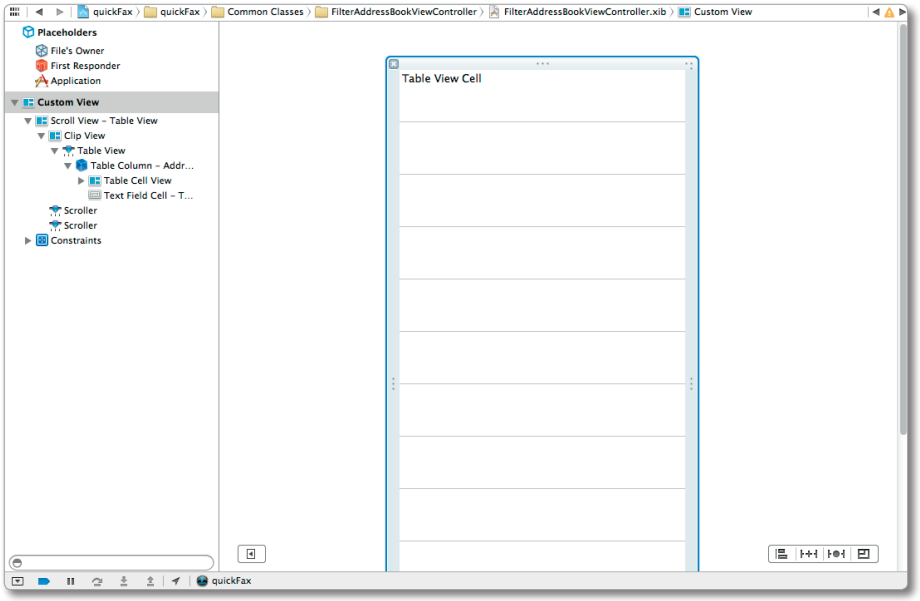
Esempio filtro contatti della rubrica

Nel momento in cui l'utente inserisce una stringa di caratteri all'interno del campo di testo "*Destinatario*", viene visualizzata una *Table View* i cui elementi sono tutti e soli i contatti presenti in rubrica che contengono all'interno dei campi nome, cognome, azienda, numero di telefono/fax almeno un'occorrenza della stringa digitata dall'utente nel campo destinatario.
Se l'utente effettua un click su una riga della tabella, il numero corrispondente viene inserito all'interno del campo destinatario.

Nota: Il campo destinatario deve essere compilato con uno ed un solo numero di fax.



Per la realizzazione della *Table View* contenente i record filtrati dalla rubrica, si è deciso di creare una nuova classe chiamata FilterAddressBookViewController (sottoclasse di NSViewController).
E' stata creata una nuova classe, anziché implementare tale funzionalità direttamente all'interno della classe QFSendFaxOnlyTextViewController, al fine di rendere il codice più modulare e riutilizzabile. Infatti in questo modo tale funzionalità potrà essere facilmente implementata anche all'interno della classe QFSendFaxWithAttachmentViewController (che sarà analizzata in dettaglio in seguito) mentre sarebbe risultato molto più difficile e complesso senza la disponibilità di una classe dedicata.
Il file .xib della classe FilterAddressBookViewController è essenzialmente composto da una unica *Table View* come mostrato in figura.



Le dimensioni della *Table View* così come l'altezza delle singole celle sono state fissate in fase di progetto in modo che la TableView visualizzi al massimo sei righe. Nel caso in cui la tabella debba mostrare più di sei record viene visualizzata una barra di scorrimento che consente all'utente di visualizzare tutti i record della tabella.

Nota: le dimensioni della Table View (altezza e larghezza) sono fisse e indipendenti dal numero di record che la tabella deve presentare all'utente.

Analizziamo ora l'algoritmo in dettaglio. Innanzitutto, all'interno del metodo inizializzatore della classe QFSendFaxOnlyTextViewController, viene allocata e istanziata la classe FilterAddressBookViewController: tale istanza prende il nome di *displayFilterAddressBookViewController*.

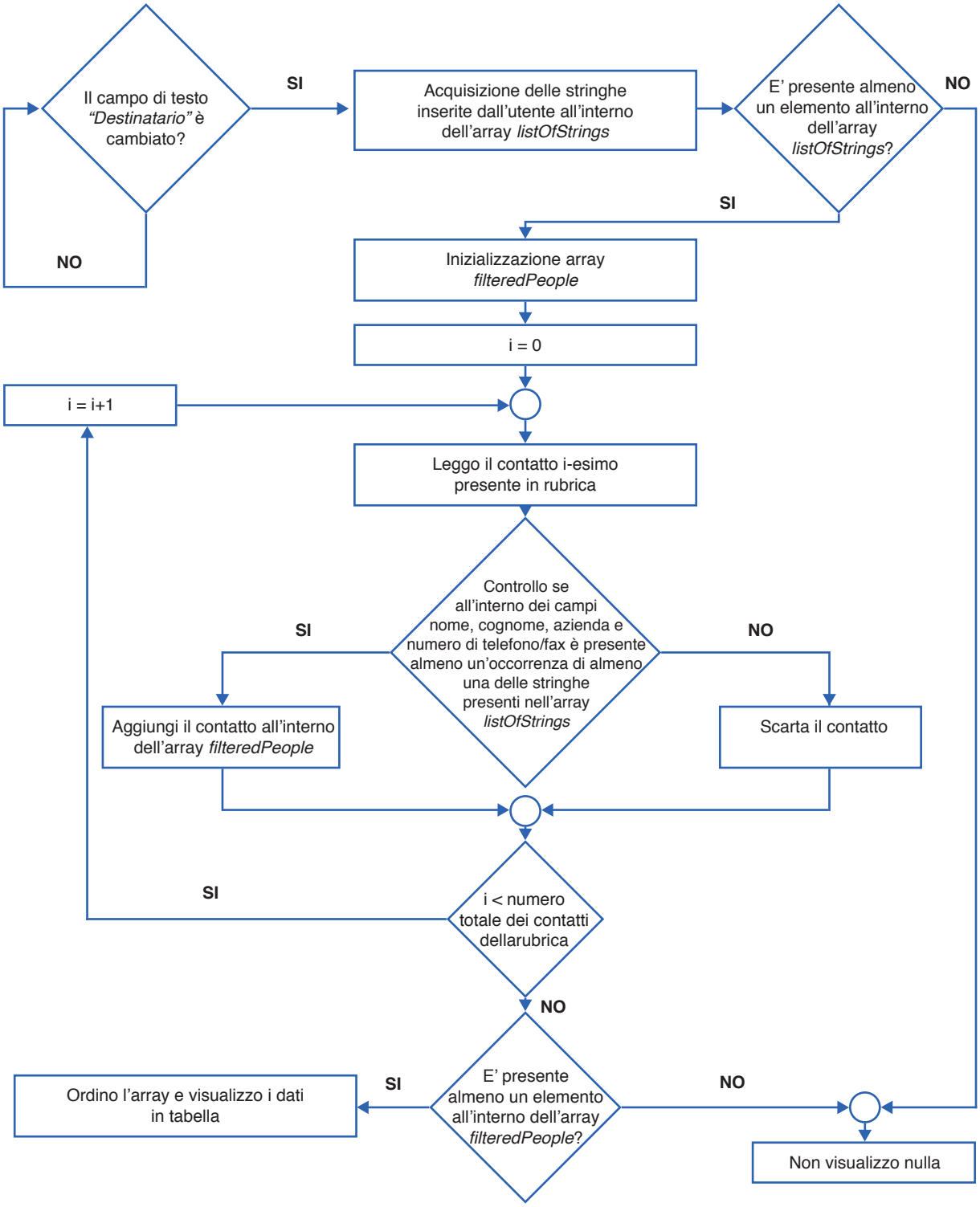
Nota: tale istanza è necessaria al fine di accedere ai metodi messi a disposizione dalla classe.

Il metodo inizializzatore si occupa anche di disporre correttamente la *Table View,* la quale viene disegnata in modo che il lato superiore della tabella coincida con il lato inferiore del campo di testo destinatario.
Al fine di intercettare una qualsiasi modifica effettuata dall'utente al campo "*Destinatario*", è necessario implementare all'interno della classe QFSendFaxOnlyTextViewController il metodo `controlTextDidChange:` conforme al protocollo NSTextFieldDelegate; tale metodo verrà invocato ogni qual volta l'utente inserisce, cancella o modifica un qualunque carattere presente nel campo di testo "*Destinatario*".

Nota: si può definire brevemente un protocollo come una collezione di metodi; una classe si dice aderire al protocollo se definisce tutti (o parte) dei metodi che fanno parte del protocollo.

All'interno di tale metodo viene invocato il metodo computeFilter (che costituisce il fulcro della classe FilterAddressBookViewController) al quale viene passato un array di stringhe. Vi è una corrispondenza uno a uno fra le stringhe memorizzate all'interno dell'array e le stringhe inserite dall'utente all'interno del campo destinatario.

Nota: le stringhe sono separate dal carattere blankspace.



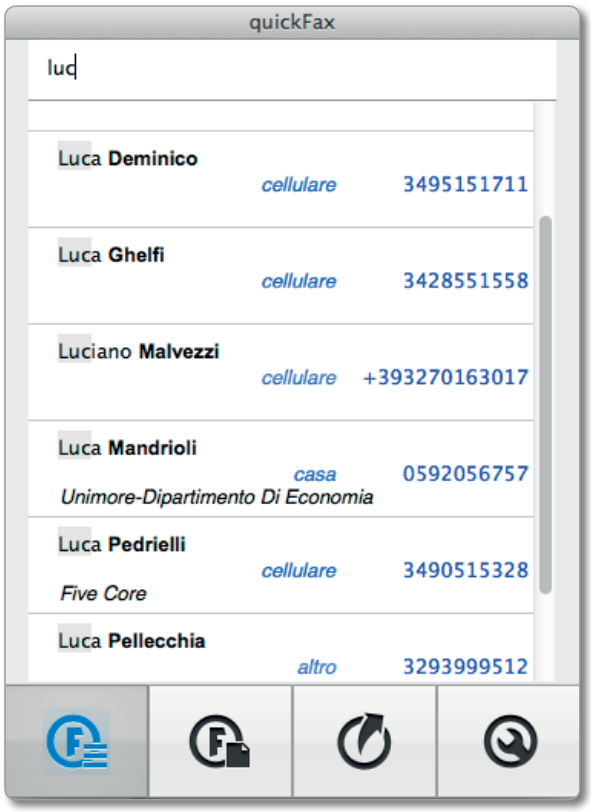
Il diagramma di flusso presente in figura descrive l'algoritmo risolutivo per il filtro dei contatti della rubrica

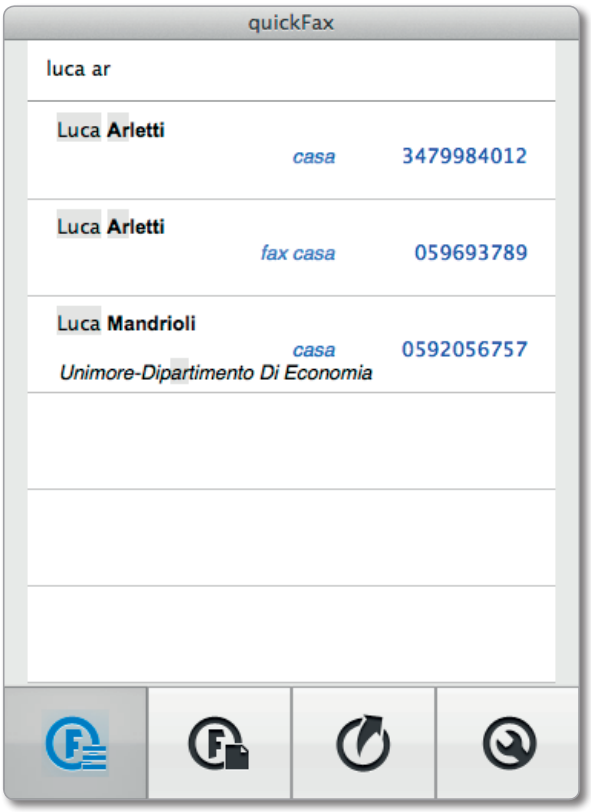
Ora ragioniamo sul problema che si deve risolvere:nell'ipotesi più generale in cui l'utente abbia inserito N stringhe, l'algoritmo dovrà mostrare all'interno della *Table View* tutti e soli i contatti che contengono all'interno dei campi almeno un'occorrenza di almeno una delle N stringhe inserite dall'utente. Tale funzionalità è implementata proprio all'interno del metodo `computeFilter:` che potremmo riassumere nei seguenti passaggi:

- Acquisizione dell'array di stringhe inserite dall'utente (*listOfStrings)*
- Ricerca per ogni contatto presente in rubrica, se all'interno dei campi nome, cognome, azienda e numero di telefono/fax è presente almeno un'occorrenza di almeno una delle stringhe presenti nell'array *listOfStrings*.
- In caso affermativo il contatto viene memorizzato all'interno di un opportuno array chiamato, in riferimento al codice, *filteredPeople*.
- Visualizzazione dei dati nella *Table View*, opportunamente ordinati.

Per concludere, al termine dell'esecuzione viene infine invocato il metodo `tableView:viewForTableColumn:row` il quale per ogni elemento memorizzato all'interno dell'array *filteredPeople* costruisce una riga della tabella visualizzando opportunamente le seguenti informazioni (non tutte le informazioni potrebbero essere presenti):

- Nome
- Cognome
- Azienda
- Numero associato





4.1.2 Funzione di invio fax

Nel momento in cui l'utente effettua un click sul pulsante "INVIA", viene invocato il corrispondente metodo associato al pulsante:

```
- (IBAction)clickSendFax:(id)sender
```

L'algoritmo implementato all'interno di tale metodo si compone di tre fasi fondamentali:

- Controllo campi di testo compilati dall'utente
- Esecuzione dell'animazione dedicata
- Invio del Fax

Analizziamo ora le varie fasi sopra riportate.
La prima fase consiste nel controllare che tutti i campi di testo siano stati compilati in modo corretto dall'utente; nel caso in cui uno o più campi siano stati compilati in modo sbagliato, il relativo errore viene mostrato a video attraverso la comparsa di una o più label temporizzate.
I possibili errori che possono essere riscontrati in questa fase sono:

- Uno o più campi non sono compilati (*tutti i campi di testo sono OBBLIGATORI.*)
- Numero di telefono non corretto (*si controlla solamente che i caratteri inseriti dall'utente siano una combinazione dei seguenti caratteri "0123456789+". Il controllo dell'effettiva correttezza del numero di fax inserito è delegato all'ultima fase.*)
- Il messaggio contiene solo la firma

Di seguito possiamo vedere alcuni esempi di errori:

Se non vengono riscontrati errori nella compilazione dei campi di testo si passa alla fase successiva.
Nella seconda fase viene invocato un metodo dedicato il quale si occupa di eseguire un'animazione che trasla verso l'alto la vista attuale e mostra all'utente una nuova vista, mediante animazione di tipo *fade-in*, il cui compito è quello di informare l'utente sullo stato di trasmissione del fax.

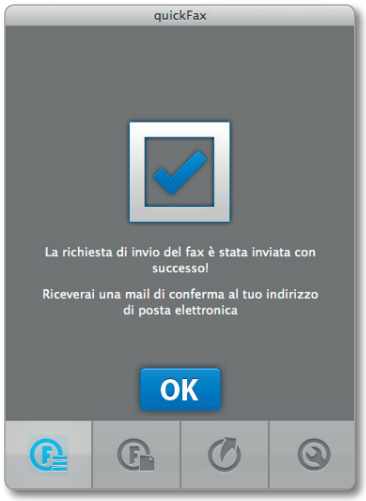
Infine si passa all'ultima fase che consente l'invio della richiesta di trasmissione del fax. Il metodo fondamentale che consente tale trasmissione è il seguente:

```
-(NSInteger) sendFaxOnlyText:(NSString*) recipientPhone
andObject:(NSString*) object andRecipientName:(NSString*)
recipientName andMessage:(NSString*) message
```

Tale metodo richiede come parametri di ingresso rispettivamente: numero di fax, oggetto, mittente e messaggio e ritorna un codice di controllo intero; se il codice ritornato dal metodo è uguale a 0 allora significa che la richiesta di invio del fax è stata inoltrata correttamente ai server di Faxator. L'utente riceverà per posta elettronica una mail di conferma dell'avvenuta trasmissione del fax nel momento in cui il servizio Faxator effettua l'inoltro del fax.
Nel caso in cui invece il codice ritornato dal metodo suddetto sia diverso da 0 significa che si è verificato un errore durante la trasmissione della richiesta di invio fax; a seconda del numero del codice possiamo distinguere vari tipi di errori riassunti di seguito:

- codice {-1,1,2} Si è verificato un problema durante la connessione con il server
- codice {4} Il numero di Fax inserito non è un numero di fax valido
- codice {5} Il servizio non è momentaneamente disponibile

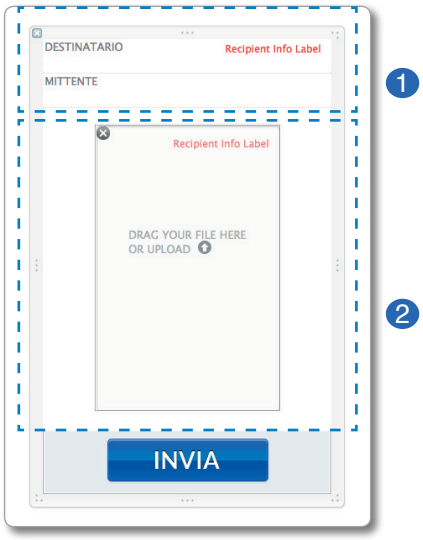
Nel caso in cui il fax sia stato trasmesso con successo viene mostrata la vista presente nella seguente figura.



L'utente effettuando un click sul pulsante "OK" conferma di aver preso visione dell'avvenuta trasmissione (o del relativo errore nel caso in cui la trasmissione non sia andata a buon fine) e viene riportato alla view di invio fax di solo testo mediante esecuzione di apposita animazione.

4.2 La view di invio fax con allegato

La view di invio fax con allegato rende l'applicazione molto più versatile in quanto consente all'utente di interagire direttamente con i file memorizzati all'interno del computer, potendo così inviare tramite fax un qualunque file (verificando che il formato sia compatibile) in modo del tutto simile all'invio di messaggi di posta elettronica contenti file allegati.
A differenza dei messaggi di posta elettronica la view suddetta non consente di inviare un numero multiplo di file nell'ambito dello stesso fax; l'utente può dunque inviare un solo file alla volta.
Nel caso in cui l'utente abbia necessità di inviare N file allo stesso destinatario, dovrà necessariamente comporre N fax (uno per ogni file). Nella figura successiva è riportata la struttura del file .xib della classe QFSendFaxWithAttachmentViewController (già creata durante la realizzazione dell'interfaccia grafica) la quale si occupa di gestire tutte le funzionalità messe a disposizione da tale vista; essa si può suddividere a livello logico in due sezioni:



1. La prima sezione è composta da due campi di testo ovvero Destinatario e Mittente; tali campi sono del tutto analoghi a quelli presenti nella view di invio fax di solo testo (per ulteriore approfondimento consultare il paragrafo 4.1).

2. La seconda sezione consente all'utente di selezionare quale file inviare al numero di fax destinatario. La view mette a disposizione due modalità che consentono all'utente di selezionare il file desiderato: ***Drag & Drop*** o ***Panello di selezione***.

- **Drag & Drop**: l'utente può trascinare il file desiderato all'interno di un'area specifica della view, definita nel file .xib da un oggetto opportuno (verrà spiegato in dettaglio successivamente). Nel caso in cui il formato del file selezionato non sia compatibile con l'applicazione, il drag & drop viene rifiutato.

- **Pannello di selezione**: mediante la pressione del pulsante viene visualizzato un pannello standard che consente all'utente di navigare all'interno della gerarchia del file system del proprio computer, in modo da poter selezionare il file desiderato. Il pannello consente di selezionare solo i tipi di file che siano compatibili con il servizio Faxator (quindi che possono essere inoltrati tramite fax).

Una volta che l'utente ha selezionato un file il cui formato è supportato dall'applicazione, viene presentata un'anteprima di esso in modo che l'utente possa verificare che il file selezionato sia corretto.

Nel caso in cui l'utente voglia inviare un file diverso da quello selezionato dovrà premere sul pulsante il quale consente di cancellare a livello logico il file selezionato, eliminando l'anteprima e riabilitando di conseguenza le due modalità sopra citate per la selezione del file.
Nel momento in cui l'utente preme sul pulsante "INVIA", l'applicazione controlla che i campi di testo siano compilati correttamente e che sia stato selezionato un file; in caso affermativo viene eseguita un'animazione apposita per informare l'utente sullo stato di trasmissione del fax. Nel caso in cui non siano riscontrati errori un messaggio informativo comunica all'utente che la richiesta di invio del fax è stata inoltrata con successo; in caso contrario l'errore riscontrato verrà comunicato all'utente mediante apposito messaggio.
Vediamo ora nel dettaglio le varie funzionalità implementate.

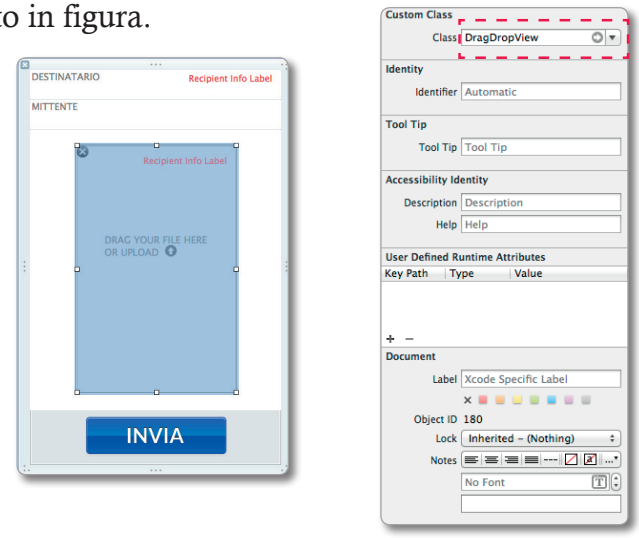
4.2.1 Drag & Drop

Il drag & drop è sicuramente il metodo più semplice ed intuitivo con il quale l'utente può selezionare un file da inviare mediante fax; l'utente deve semplicemente trascinare l'icona relativa al file desiderato all'interno dell'apposita area presente nella view per far si che l'applicazione memorizzi tale file a livello logico (viene memorizzato il path assoluto del file). [8]
Vediamo ora nel dettaglio come si è realizzata tale funzionalità.
Innanzitutto si è definita l'area della view che risulta essere ricettiva al drag & drop; per definire tale area è stato aggiunto al file .xib una custom view le cui dimensioni delimitano l'area che risulta essere ricettiva alle operazioni di drag & drop effettuate dall'utente.

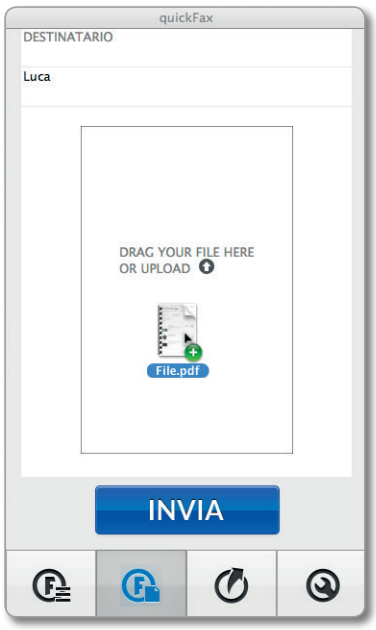
Si noti che è possibile utilizzare anche altre tipologie di oggetti per definire l'area sopra citata, l'importante è che la classe dell'oggetto in questione sia necessariamente sottoclasse di NSView (o NSWindow) in quanto solamente gli oggetti che ereditano da tale classe possono implementare i metodi relativi al drag & drop.

Nota: in Mac OS X solamente finestre e view possono essere utilizzate come sorgenti o destinazioni per il drag & drop.

Una volta definita la view sopra citata si è creata la classe DragDropView, sottoclasse di NSView, all'interno della quale sono stati implementati i metodi fondamentali per effettuare l'operazione di drag & drop. Infine si è definito che l'oggetto di tipo NSView appartenga alla classe DragDropView specificandola all'interno della barra Identity inspector come mostrato in figura.



I metodi fondamentali per il drag & drop sono definiti all'interno del protocollo NSDraggingDestination per cui è necessario che la classe DragDropView sia conforme al protocollo citato.
Si può definire brevemente un protocollo come una collezione di metodi; una classe si dice aderire al protocollo se definisce tutti (o parte) dei metodi che fanno parte del protocollo.
Vediamo ora in che ordine vengono invocati i sei metodi del protocollo NSDraggingDestination nel momento in cui l'utente tenta di effettuare il drag & drop di uno specifico file.
Nel momento in cui l'utente trascina un file all'interno dell'area ricettiva al drag & drop viene richiamato il metodo `draggingEntered:` il quale si occupa di controllare che il file selezionato dall'utente sia uno solo e che il tipo di file sia supportato dall'applicazione; tale verifica viene effettuata controllando che l'estensione del file selezionato sia inclusa nella lista di tutte le estensioni supportate dal servizio Faxator. In caso affermativo si può procedere con l'operazione altrimenti l'operazione di drag & drop viene bloccata.
L'icona assume un aspetto diverso nel caso in cui il file trascinato dall'utente risulti essere corretto in modo da comunicare immediatamente all'utente in maniera visiva se l'operazione può essere eseguita correttamente.



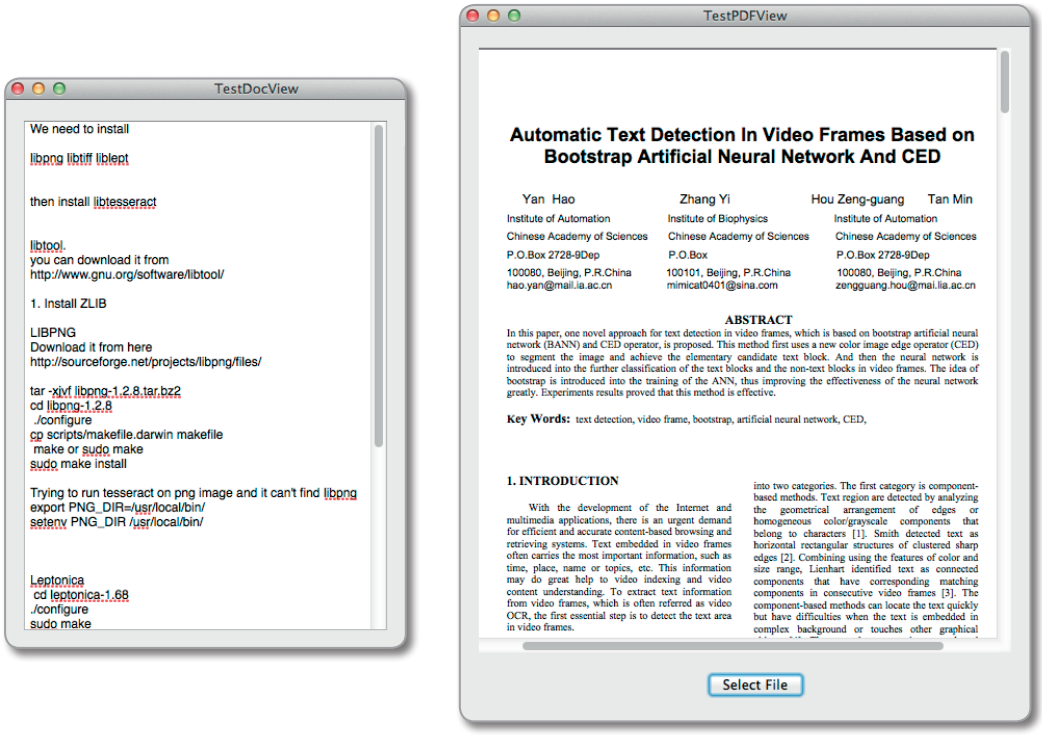
Finché l'utente mantiene il file all'interno dell'area, viene invocato periodicamente il metodo `draggingUpdated:` .
Se il file trascinato dall'utente esce dall'area relativa al drag & drop viene richiamato il metodo `draggingExited:` e nessun metodo viene più invocato finché il file non rientra all'interno dell'area sopra citata.
Se il file si trova all'interno dell'area e il tipo di file trascinato dall'utente è compatibile con l'applicazione, nel momento in cui il file viene rilasciato viene invocato il metodo `prepareForDragOperation:`.
Nel caso in cui il metodo sopra citato ritorni YES viene invocato il metodo `performDragOperation:` il quale si occupa di memorizzare il percorso assoluto del file trascinato dall'utente ed invocare la visualizzazione dell'anteprima.
Infine se il metodo `performDragOperation:` ritorna YES viene invocato il metodo `concludeDragOperation:`.
La sequenza di immagini che segue mostra ciò che accade quando l'utente effettua il drag & drop di un file compatibile con l'applicazione.

4.2.2 Anteprima file

La view di invio fax con allegato permette all'utente di selezionare un file presente sul proprio computer, attraverso il drag & drop (come descritto nel paragrafo precedente) oppure mediante il pannello standard di selezione. Una volta che l'utente ha selezionato un file viene visualizzata un'anteprima del contenuto del file in modo che l'utente possa verificare che il file da lui scelto sia corretto.
Vediamo ora come è stato possibile realizzare tale funzionalità.
Come per il drag & drop anche in questo caso si è innanzitutto definita l'area della view adibita alla visualizzazione dell'anteprima del file; essa è stata realizzata ponendo all'interno del file .xib corrispondente, l'oggetto *ImageWell*, sottoclasse di NSImageView, il quale permette di visualizzare un documento o una immagine all'interno del riquadro dedicato.

E' importante sottolineare che l'oggetto *ImageWell* era stato precedentemente utilizzato all'interno della vista di dettaglio della view dei fax recenti, per mostrare l'anteprima del fax inviato dall'utente (vedi paragrafo 3.5).
Il primo problema che si è presentato riguardava il fatto che la classe NSImageView a cui appartiene l'oggetto *ImageWell* è in grado di processare correttamente solamente i vari tipi di immagini e i file in formato PDF, i quali sono solamente un sottoinsieme dei tipi di file che l'utente può selezionare e che risultano essere compatibili con l'applicazione;ad esempio non è in grado di rappresentare i vari formati dei file di testo (rtf, doc, docx) che invece risultano essere compatibili con l'applicazione e il servizio Faxator e che possono essere inoltrati correttamente come fax.
Inoltre, per quanto riguarda la visualizzazione dei file in formato PDF, viene visualizzata solamente la prima pagina con impossibilità dunque da parte dell'utente di scorrere le pagine nel caso in cui il file PDF presenti un numero multiplo di pagine. E' dunque evidente che, per i motivi sopra citati, la soluzione adottata (giù utilizzata in precedenza) risulta essere inadeguata per il problema attualmente in esame.
Nel caso della vista di dettaglio della view dei fax recenti infatti si doveva solo fornire una visualizzazione qualitativa del resoconto del fax inviato dall'utente, il formato del file è sempre lo stesso (ovvero un file PDF) e tutte le informazioni fondamentali sono presenti nella prima pagina(quindi non si presentava il problema di file PDF a pagine multiple).
Una possibile soluzione ipotizzata tratta in modo differente la visualizzazione del contenuto sulla base del tipo di file selezionato dall'utente. Una volta nota la tipologia di file si utilizza una classe specifica che è in grado di rappresentare correttamente il contenuto del documento.

Le immagini sottostanti mostrano due esempi dove vengono utilizzate rispettivamente la classe PDFView e la classe NSTextView per visualizzare file in formato PDF e rtf.



Tale approccio presenta però due problemi fondamentali:

1. Dato che è necessario utilizzare una classe differente per ogni tipo di file di cui si vuole visualizzare il contenuto, l'algoritmo rischia di diventare eccessivamente esoso in termini di complessità.

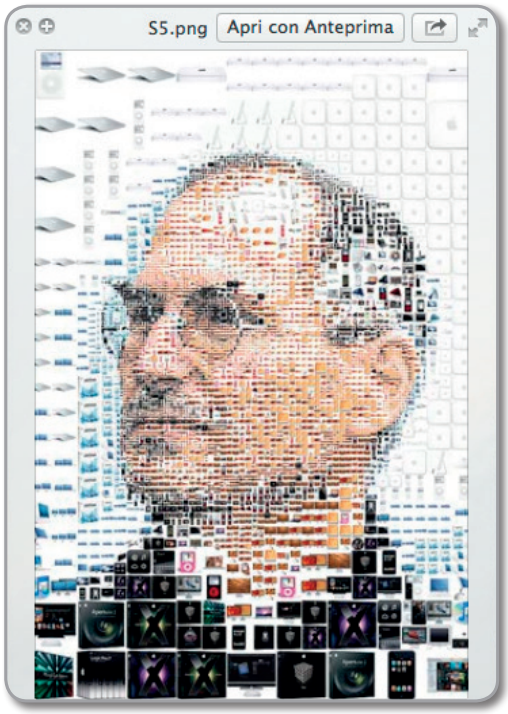
2. Ogni volta che il servizio Faxator modifica le tipologie dei file compatibili (ad esempio aggiungendo nuovi tipi di file) risulta necessario modificare l'algoritmo di conseguenza.

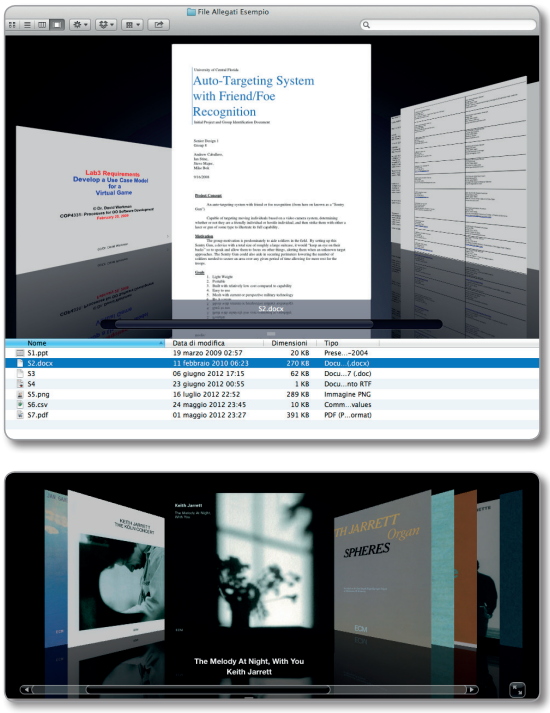
A causa di tali problematiche, dopo aver condotto alcune ricerche, si è deciso di utilizzare un particolare framework chiamato *Quick Look*, integrato all'interno del sistema operativo Mac OS X, in grado di mostrare l'anteprima di un insieme notevolmente vasto di tipologie di file (molti di più rispetto a quelli realmente compatibili con il servizio Faxator).

4.2.3 Il framework Quick Look

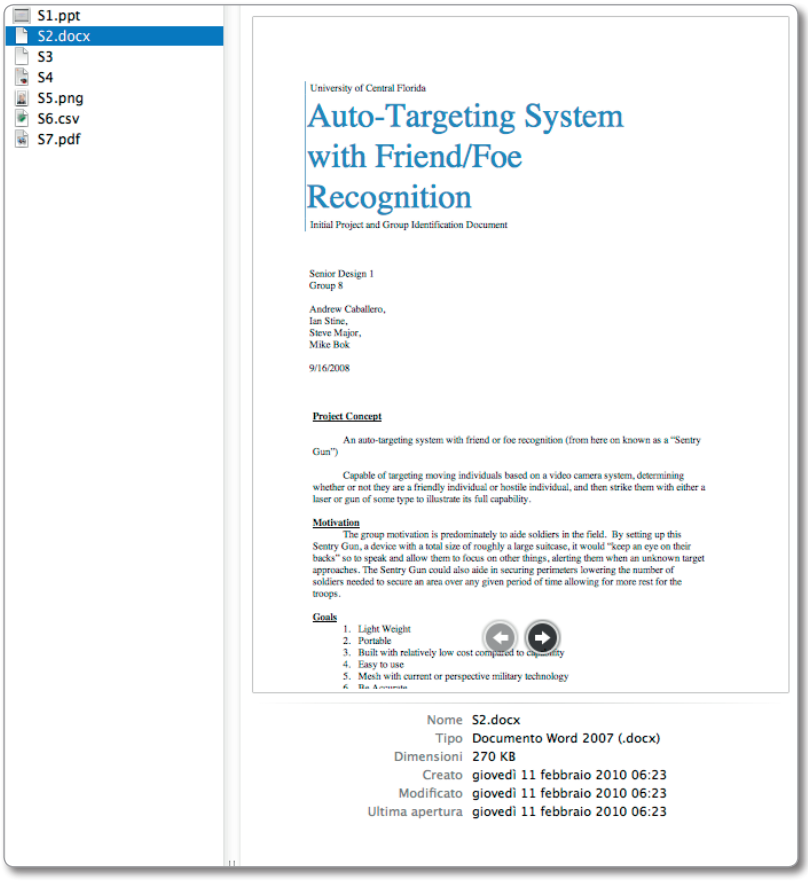
Incluso all'interno del sistema operativo Mac OS X a partire dalla versione 10.5, il framework *Quick Look* permette di visualizzare il contenuto di un file, senza dover necessariamente eseguire l'applicazione che lo ha generato, in quanto supporta in maniera nativa un numero molto ampio (e in continua espansione) di tipologie di file. [9]
Quick Look risulta essere notevolmente utile ed efficace in applicazioni che presentano all'utente una serie di file sotto forma di lista (come ad esempio Finder, Spotlight e Time Machine) in quanto consentono di identificare un particolare file, oltre che sulla base del nome, dell'icona e di relativi metadati, anche sulla base del suo contenuto generando un'anteprima che l'utente può consultare in modo molto più veloce rispetto all'apertura della corrispondente applicazione.

Nota: la modalità di visualizzazione di tipo CoverFlow utilizzata in iTunes sfrutta proprio il framework Quick Look.

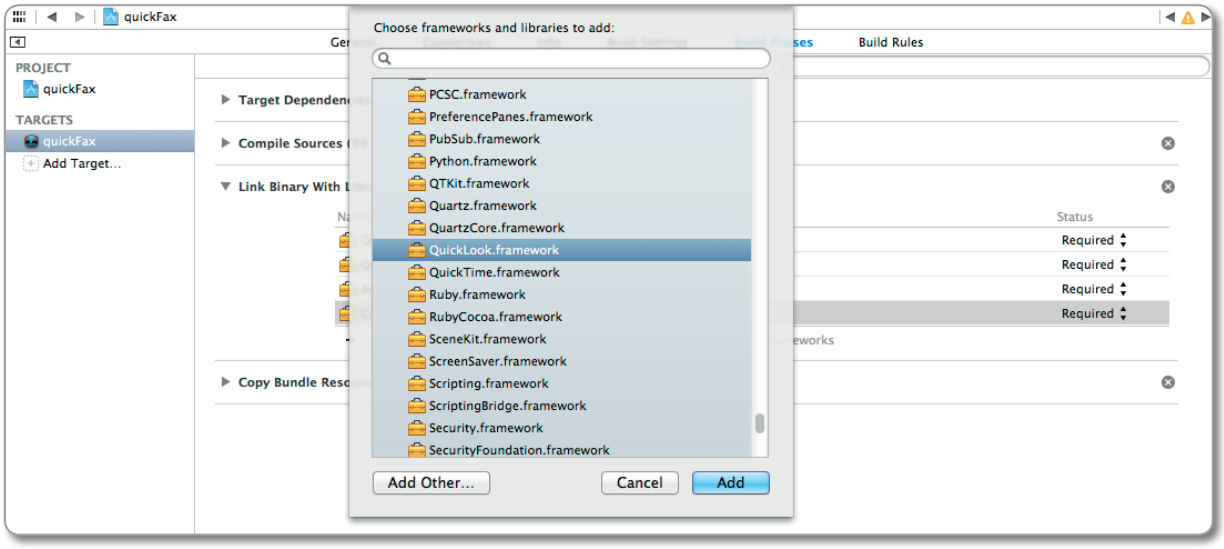




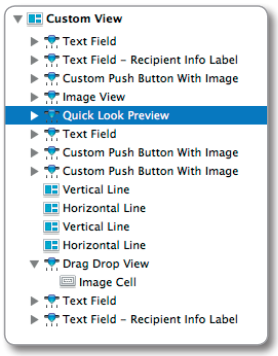
Inoltre, *Quick Look* è in grado di rappresentare documenti costituiti da un numero multiplo di pagine, consentendo all'utente di navigare fra esse utilizzando le apposite frecce, come mostrato nella figura seguente:

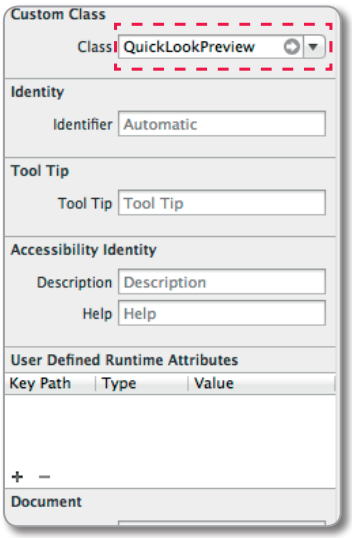


E' proprio quest'ultima tipologia di rappresentazione che è stata integrata all'interno dell'applicazione.
Per utilizzare correttamente il framework *Quick Look* è innanzitutto necessario importarlo all'interno del progetto (il procedimento necessario per importare un framework è stato trattato al paragrafo 3.5.1).



Una volta aggiunto il framework è possibile utilizzare i metodi messi a disposizione da esso; successivamente si è impostato che la classe di appartenenza per l'oggetto *ImageWell* non fosse la classe standard NSImageView, ma una nuova classe denominata QuickLookPreview, contenete un metodo fondamentale il quale permette di visualizzare l'anteprima di un file, noto il suo percorso assoluto.





E' importante osservare che la classe QuickLookPreview non implementa alcun controllo sul tipo di file selezionato dall'utente. Tale controllo viene eseguito nel momento in cui il file è effettivamente scelto o mediante pannello di selezione o mediante drag & drop; superato tale controllo si suppone che il file selezionato sia compatibile con il servizio Faxator.

4.2.4 Funzione di invio fax

Nel momento in cui l'utente Nel momento in cui l'utente effettua un click sul pulsante "INVIA", viene invocato il corrispondente metodo associato al pulsante:

```
- (IBAction)clickSendFax:(id)sender
```

L'algoritmo implementato all'interno di tale metodo risulta essere lo stesso utilizzato per la funzione di invio fax di solo testo, già descritto in precedenza (vedi paragrafo 4.1.2, al quale si rimanda per approfondimento); per completezza riportiamo le tre fasi fondamentali da cui è composto l'algoritmo implementate all'interno del suddetto metodo:

- Controllo campi di testo compilati dall'utente
- Esecuzione dell'animazione dedicata
- Invio del Fax

Ciò che cambia è il metodo utilizzato nell'ultima fase, il quale consente l'invio del file selezionato dall'utente sotto forma di fax; il metodo fondamentale che consente tale trasmissione è il seguente:

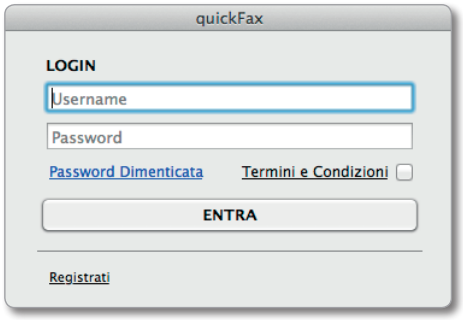
```
-(NSInteger) sendFaxOnlyText:(NSString*) recipientPhone
andObject:(NSString*) object andRecipientName:(NSString*)
recipientName andMessage:(NSString*) message
```

Esso richiede come parametri d'ingresso rispettivamente: numero di fax, nome del mittente e il path assoluto del file selezionato. E' importante sottolineare che prima di poter inoltrare la richiesta di invio del fax ai server di Faxator, il metodo si deve occupare di codificare in maniera opportuna il file selezionato dall'utente; la codifica utilizzata è il formato in *Base 64*.
La codifica in *Base 64*, utilizzata in diversi ambiti applicativi,consente di rappresentare un insieme di dati binari (il file selezionato) in forma testuale utilizzando solamente caratteri ASCII; tale codifica si rende necessaria ogni qual volta e richiesto l'invio di un file ad un Web Server.
Una volta codificato e inoltrata la richiesta di invio ai server di Faxator, il metodo ritorna un codice di controllo intero. I valori ritornati e il relativo significato sono già stati analizzati nel paragrafo 4.1.2 al quale si rimanda per approfondimento.

Capitolo 5

Cenni alle viste di login, registrazione e impostazioni

5.1 La view di login



La view di login consente all'utente di inserire le proprie credenziali al fine di identificare in modo univoco la propria identità e avere accesso ai vari servizi messi a disposizione dall'applicazione.
E' requisito fondamentale verificare la correttezza delle credenziali fornite dall'utente e, solamente in caso affermativo, egli può avere accesso alle funzionalità dell'applicazione.
All'avvio dell'applicazione viene dunque presentata all'utente la view di login nella quale sono presenti due campi di testo; l'utente dovrà fornire il proprio indirizzo mail e la password definiti al momento della registrazione.
Entrambi i campi di testo sono obbligatori e devono necessariamente essere compilati dall'utente; al fine di garantire una maggiore sicurezza i caratteri inseriti all'interno del campo password non sono visualizzati in chiaro ma vengono oscurati.
Oltre a fornire indirizzo mail e password l'utente deve anche accettare i termini e le condizioni imposti dal servizio Faxator, apportando il segno di spunta sulla checkbox relativa. Tali termini sono visionabili effettuando clik sul link ipertestuale "*Termini e Condizioni*".
La procedura di login non viene avviata finché l'utente non accetta i termini del servizio.
La view di login è gestita dalla classe QFLoginViewController; la struttura del relativo file .xib è rappresentata nella seguente figura:

Nel momento in cui l'utente effettua un click sul pulsante "ENTRA" viene invocato il corrispondente metodo associato al pulsante:

```
- (IBAction)clickLoginButton:(id)sender
```

L'algoritmo implementato all'interno di tale metodo si compone di due fasi fondamentali:

- Controllo campi di testo compilati dall'utente e accettazione termini
- Esecuzione della procedura di login

La prima fase consiste nel controllare che tutti i campi di testo siano stati compilati in modo corretto e che l'utente abbia accettato i termini e le condizioni apportando il segno di spunta sulla checkbox; la procedura di login non viene avviata finché tali condizioni non sono verificate.
Di seguito possiamo vedere alcuni possibili errori:

Nel caso in cui non siano riscontrati errori si passa alla fase successiva durante la quale viene eseguita la procedura di login vera e propria. Le informazioni inserite dall'utente vengono inoltrate al server per verificare che l'indirizzo mail e la password fornite siano corrette e corrispondano ad un utente registrato.
Il metodo che si occupa di effettuare la verifica delle credenziali fornite è il seguente:

```
-(NSInteger)userLogIn:(NSString*) username andPassword:(NSString*)
password
```

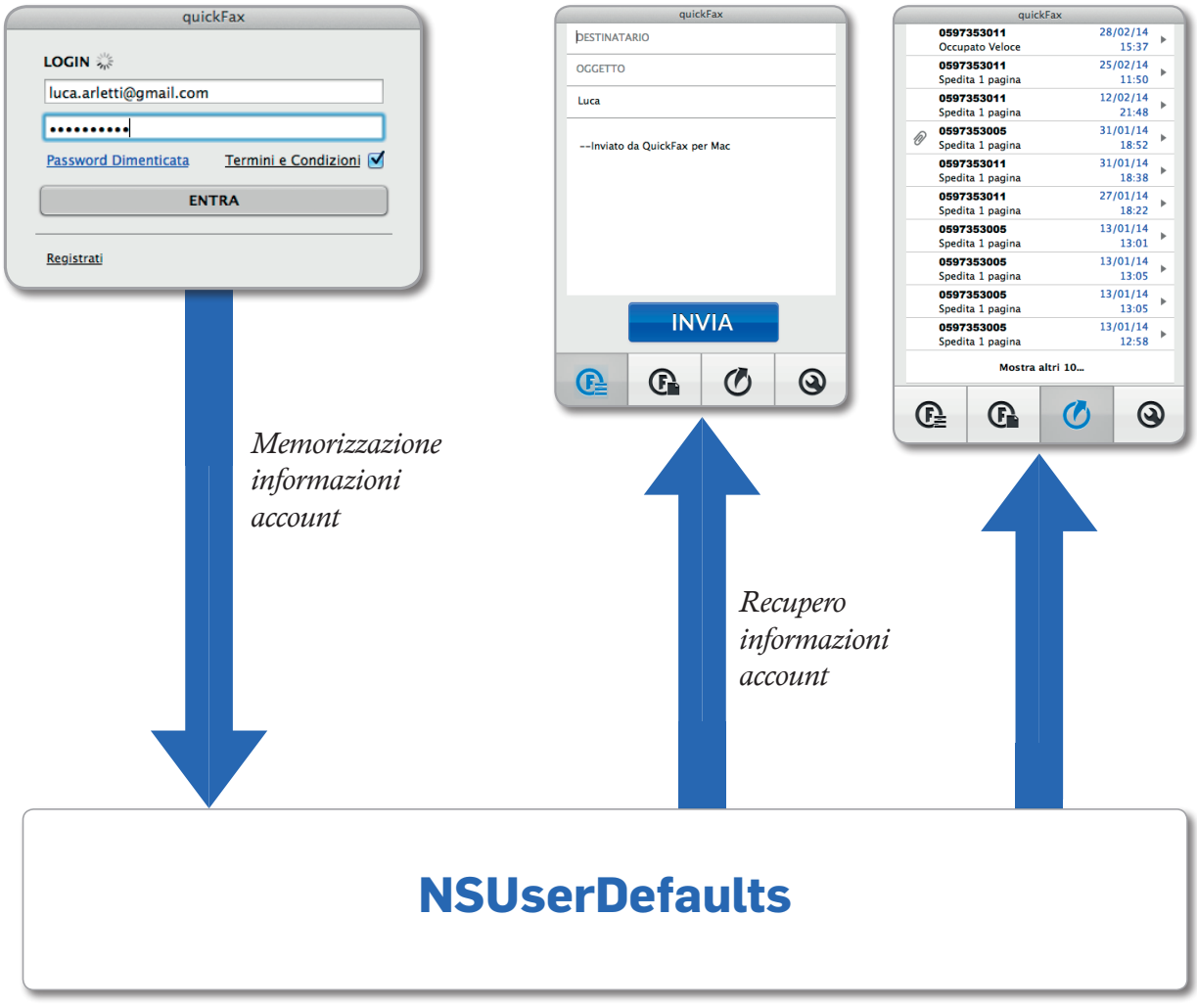
In accordo con la logica di programmazione già utilizzata in precedenza per i metodi che gestiscono l'invio di un fax di solo testo o con allegato, il metodo suddetto ritorna un codice di controllo intero il cui valore è uguale a 0 solamente se le credenziali fornite dall'utente risultano essere corrette;in caso contrario (ovvero viene ritornato un valore diverso da 0) significa che si è verificato un errore.
Le varie tipologie di errori sono riassunte di seguito:

- codice {-1} Problema di connessione con il server.
- codice {2} Indirizzo mail o password non corretti.
- codice {3} Indirizzo mail e password corretti ma l'account registrato non è stato ancora verificato.

Nel caso in cui le credenziali siano state verificate e risultino essere corrette, viene eseguita un'animazione che presenta la view di invio fax di solo testo. A questo punto l'utente ha accesso a tutte le funzionalità messe a disposizione dall'applicazione.
E' fondamentale sottolineare che nel momento in cui l'operazione di login viene eseguita con successo, le informazioni fondamentali riguardanti l'account (come ad esempio indirizzo mail, password, nome mittente) vengono memorizzate utilizzando la classe NSUserDefaults, in modo da renderle immediatamente disponibili quando necessario.
La classe NSUserDeafults fornisce un'interfaccia di programmazione per interagire con il sistema default, all'interno del quale è possibile salvare informazioni in modo da recuperarle agevolmente in futuro. La classe NSUserDefaults utilizza il metodo di programmazione *Key-Value Coding* per memorizzare gli oggetti.

Nota: il Key-Value Coding (o KVC) è un meccanismo che permette di impostare e ottenere un oggetto utilizzando il suo nome (ovvero la chiave associata all'oggetto).

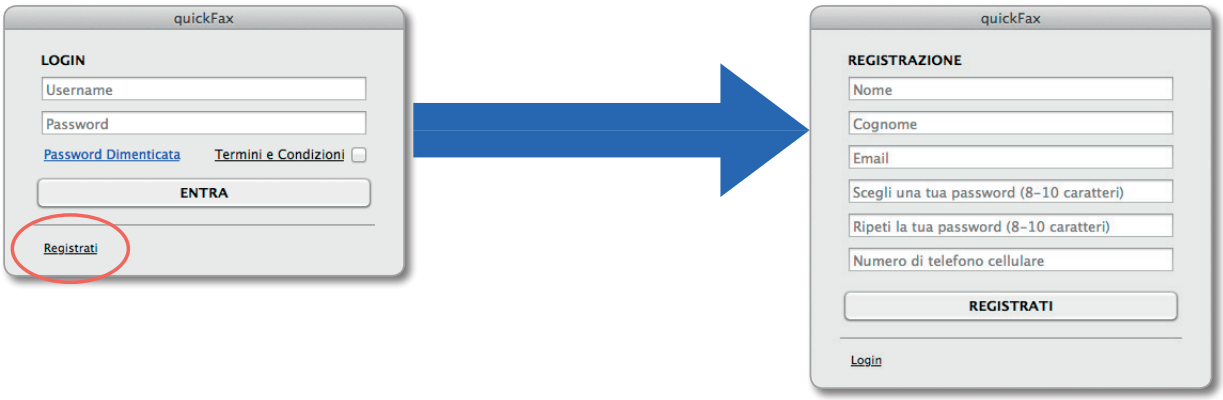
Il seguente esempio rappresenta in maniera schematica come le informazioni memorizzate nella fase di login vengano utilizzate successivamente per visualizzare i fax recenti associati ad un preciso account o per visualizzare il nome del mittente che verrà usato nell'invio del fax. Tali informazioni sono utilizzate anche durante il processo di invio fax di un fax di solo testo o con allegato.



5.2 La view di registrazione

La view di registrazione permette ai nuovi utenti di registrarsi presso il servizio Faxator in modo da poter accedere alle funzionalità messe a disposizione dell'applicazione; come già anticipato nel paragrafo precedente solamente gli utenti registrati possano accedere all'applicazione, in quanto vengono verificate le credenziali fornite dall'utente all'avvio.
E' possibile accedere alla view di registrazione direttamente dalla view di login mediante click sul pulsante dedicato.

Nota: nel passaggio dalla view di login alla view di registrazione viene utilizzata l'animazione a scorrimento analizzata nel paragrafo 3.6 a cui si rimanda per approfondimento.



La view di registrazione è essenzialmente composta da una serie di campi di testo, i quali devono tutti essere compilati in modo opportuno dall'utente al fine di consentirne la corretta registrazione.

I campi di testo sono i seguenti:

- Nome
- Cognome
- Indirizzo mail
- Password
- Numero di telefono cellulare

La view di registrazione è gestita dalla classe QFRegisterNewUserViewController; la struttura del file .xib è rappresentata nella seguente figura:

Nel momento in cui l'utente effettua un click sul pulsante "REGISTRATI" viene invocato il corrispondente metodo associato al pulsante:

```
- (IBAction)clickRegister:(id)sender
```

In accordo con la logica di programmazione più volte utilizzata in precedenza (vedi funzionalità di login e invio fax) l'algoritmo implementato all'interno di tale metodo si compone di due fasi fondamentali:

- Controllo campi di testo compilati dall'utente
- Esecuzione della procedura di registrazione

La prima fase consiste nel controllare che tutti i campi di testo siano stati compilati in modo corretto dall'utente. Tutti i campi di testo risultano essere obbligatori; pertanto la procedura di registrazione non viene avviata finché i campi non sono compilati in modo opportuno.

Di seguito si riassumono i controlli effettuati:

- Nessun campo deve essere vuoto
- La lunghezza della password deve essere compresa fra un minimo di otto e un massimo di dieci caratteri
- Le stringhe inserite nei due campi password risultano essere uguali
- Il numero di cellulare contiene solamente caratteri numerici
- La password è composta solamente da caratteri alfanumerici
- Il formato dell'indirizzo mail fornito è valido

Nel caso in cui non siano riscontrati errori si passa alla fase successiva durante la quale viene eseguita la procedura di registrazione vera e propria. Viene richiamato un metodo specifico, il quale prende in ingresso le informazioni inserite dall'utente nei relativi campi di testo e ritorna un codice di controllo. Il metodo che si occupa di effettuare la registrazione dall'utente è il seguente:

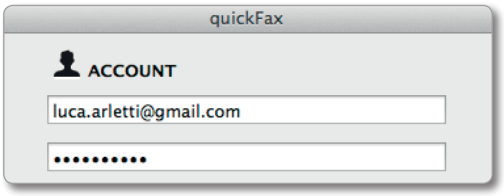
```
-(NSInteger) registerNewUser:(NSString*) name andPassword:(NSString*)
password andMail:(NSString*) email andPhoneNUmber:(NSString*) phone
```

5.3 La view impostazioni

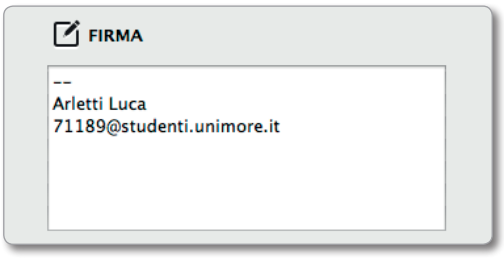
La view impostazioni è stata realizzata al fine di raggruppare una serie di funzionalità sia riguardanti la personalizzazione di alcuni parametri da parte dell'utente (come ad esempio la firma aggiunta in coda al fax di testo), sia a scopo puramente informativo (ad esempio l'account attualmente utilizzato).
In figura è riporta la struttura del file .xib della classe QFSettingsViewController (già creata durante la realizzazione dell'interfaccia grafica di base) la quale si occupa di gestire tutte le funzionalità messe a disposizione da tale vista.
Vediamo ora a quali funzionalità l'utente può avere accesso attraverso tale vista.
A livello logico è possibile suddividere la vista in tre sezioni quali:

- Account
- Firma
- Generico

La sezione account consente all'utente di verificare quale utente risulta essere l'utilizzatore dell'applicazione. Essa è composta da due campi di testo, non modificabili, che presentano l'indirizzo mail e la password dell'account attualmente in uso. In fase di analisi si era ipotizzato di migliorare tale funzionalità aggiungendo la possibilità da parte dell'utente di modificare la password attualmente in uso per quell'account; a causa del fatto che all'interno delle librerie API fornite dall'azienda non vi è un metodo specifico che consenta tale modifica, la funzionalità aggiuntiva non è stata implementata.



La sezione firma consente all'utente di impostare e modificare la firma associata all'account in uso.
Essa viene aggiunta in modo automatico alla fine di ogni fax di testo inviato.
E' possibile aggiungere o modificare la firma agendo direttamente sul campo di testo presente in tale sezione. La firma dell'utente viene aggiornata in tempo reale senza che vi sia la necessità da parte dell'utente di riavviare l'applicazione; la firma associata all'account viene memorizzata utilizzando la classe NSUserDefaults (come già descritto in precedenza nella view di login) in modo da poter essere recuperata e visualizzata nel momento in cui l'utente passa alla view di invio fax di solo testo.



L'ultima sezione raggruppa una serie di funzionalità generiche descritte di seguito:

- Accesso da parte dell'utente alla pagina di supporto del servizio Faxator.

- Accesso alle informazioni fondamentali riguardanti il prodotto, mediante la visualizzazione della view di about dedicata.

- Attivazione o disattivazione delle funzioni relative a suoni e statistiche; è importante sottolineare che tali funzionalità sono state definite in fase di analisi solamente dal punto di vista grafico (mediante utilizzo delle checkbox) lasciando la loro implementazione a sviluppi futuri.

- Funzionalità di logout la quale consente all'utente di cambiare l'account attualmente in uso. Nel momento in cui l'utente effettua un click sul pulsante "ESCI" viene presentata una vista il cui unico scopo è quello di avvisare l'utente dell'imminente logout e consentirgli di abortire l'operazione (ad esempio nel caso in cui abbia inavvertitamente premuto sul pulsante); se il logout viene confermato, l'applicazione presenta la view di login descritta in precedenza.

Nota: l'operazione di logout cancella tutte le impostazioni salvate per quell'account, come ad esempio la firma, la quale dovrà dunque essere impostata nuovamente nel momento in cui si effettua il login con lo stesso account.

Capitolo 6

Conclusioni e sviluppi futuri.

Il progetto realizzato mi ha permesso di acquisire una conoscenza approfondita nell'ambito dello sviluppo di applicativi software in ambiente Mac OS X.

Nella prime due settimane è stato effettuato uno studio personale al fine di acquisire le conoscenze di base in merito al linguaggio di programmazione Objective-C e al framework Cocoa. Inoltre è stata effettuata un'analisi che ha permesso di evidenziare le funzionalità fondamentali del programma applicativo e conseguentemente suddividere, in una serie di fasi, il lavoro di sviluppo del software.

Nelle settimane successive e per tutta la durata del tirocinio è stata fondamentale la ricerca individuale al fine di trovare la migliore soluzione per l'implementazione delle funzionalità del programma applicativo; ciò mi ha permesso di acquisire una buona metodologia di problem solving, fondamentale per affrontare i problemi di difficoltà crescente che si sono presentati durante le varie fasi di implementazione delle funzionalità.

Il progetto svolto mi ha permesso di avere una panoramica completa e approfondita della varie fasi che compongono lo sviluppo di un software, dalla progettazione fino alla realizzazione, consentendomi di mettere in pratica tutte le conoscenze acquisite in ambito universitario.

Il programma applicativo realizzato può essere ulteriormente migliorato come descritto di seguito:

- Implementazione delle funzionalità relative a suoni e statistiche presenti nella view Impostazioni
- Aggiungere la possibilità da parte dell'utente di inviare un fax ad un numero multiplo di destinatari
- Introdurre una nuova view che mostri all'utente i fax ricevuti
- Effettuare uno studio più approfondito dell'interfaccia grafica in modo da migliorare ulteriormente l'interazione con l'utente.

Bibliografia

[1] Picchi A., *Objective-C 2.0 per iOS e OS X*, Milano, Edizioni FAG Milano, 2013

[2] Kochan Stephen G., Programming in Objective-C 2.0, Boston, Addison-Wesley, 2012

[3] Isted T., *Sviluppare applicazioni con Objective-C e Cocoa*, Milano, Apogeo, 2010

[4] Novelli F., *Programmare applicazioni per Mac OS X*, Assago, Edizioni FAG, 2013

[5] Garfinkel S., *Mahoney M., Building Cocoa Applications. A Step by Step Guide*, Sebastopol, O'Reilly Media, 2012

[6] Iacubino A., *Objective-C & Cocoa. Il manuale di programmazione per Mac OS X*, Milano, Hoepli, 2013

[7] AA.VV, *Address Book Programming Guide for Mac*,Cupertino,Apple,2013

[8] AA.VV, *Drag and Drop Programming Topics*, Cupertino, Apple, 2012

[9] AA.VV, *Quick Look Programming Guide*, Cupertino, Apple, 2013

Ringraziamenti

Desidero ringraziare calorosamente la prof.ssa Bergamaschi, relatore di questa tesi, per la grandissima disponibilità e cortesia dimostrata durante la stesura di questo elaborato e per tutto l'aiuto e il supporto fornitomi durante i momenti difficili della mia carriera universitaria.

Desidero inoltre ringraziare la ditta Command Guru s.r.l,tutti i ragazzi dello staff, in particolare il CEO Alessio Zito Rossi e il mio Tutor Ivan Fontana per avermi dato la possibilità di realizzare questo progetto e per tutto quello che hanno fatto per me durante il periodo di stage.

Un sentito ringraziamento ai miei genitori, che, con il loro incrollabile sostegno morale, mi hanno permesso di raggiungere questo traguardo incoraggiandomi sempre ad andare avanti per la mia strada.

I miei ringraziamenti vanno anche a tutta la mia famiglia, in particolare i miei zii Catia e Vanni e le nonne Livia ed Elvira, che mi hanno sempre sostenuto anche quando le cose sembravano andare male.

Un ringraziamento particolare va alla mia ragazza Marina, per essermi stata sempre tanto vicina e perché nei momenti di maggiore difficoltà ha saputo essere un aiuto indispensabile.

Voglio ringraziare tutti i miei amici e compagni di corso con i quali ho passato splendide giornate e condiviso esperienze che non potrò dimenticare.

Infine vorrei dedicare questo traguardo a due persone molto speciali, veri maestri che nei momenti più difficili mi hanno dato, con i loro insegnamenti ed il loro esempio,non solo la forza necessaria a perseguire il mio obiettivo ma anche una chiave di lettura dei problemi che mi servirà nella vita futura; grazie a Giorgio e Maria.