Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea in Informatica

La gestione di grandi collezioni di dati su iPad: il caso di studio ProClara

Tesi di Laurea in Basi di dati

Relatore:
Chiar.mo Prof.
Danilo Montesi

Presentata da:
Bonfiglioli Martino

Sessione II
2010/2011

Indice

Capitolo 1

Introduzione

Poco più di un anno fa la Apple ha lanciato sul mercato l'iPad, il primo Tablet PC che ormai tutti conoscono. Come per molte innovazioni in campo tecnologico, in poco tempo questo strumento sembra già essere diventato indispensabile per molti e molti altri lo desidererebbero avere. In realtà il concetto di Tablet PC risale ad alcuni anni fa, infatti, già negli anni 80' e 90', molti designer e aziende avevano progettato strumenti simili ma no li avevano mai realizzati, in quanto non si riteneva che i costi di produzione sarebbero stati coperti dalle vendite. È inevitabile per me prendere l'iPad come esempio di Tablet PC, non solo per la mia indiscutibile passione per l'azienda californiana Apple ma soprattutto per il fatto che da marzo 2011 l'iPad copre oltre il 70% del mercato dei Tablet PC[1]. Tralasciando gli aspetti soggettivi relativi al design del prodotto, il Tablet PC ha avuto una diffusione da record. Il segreto di questo successo è sicuramente la versatilità del prodotto: studenti, bambini, professionisti, pensionati ecc...: la vera forza del Tablet PC risiede nella possibilità di utilizzare applicazioni che riguardano diversi campi, dallo studio al gioco, al lavoro e all'informazione. Così com'é accaduto per il computer e per il cellulare, anche il Tablet PC é diventato utilizzabile da chiunque. Il mio lavoro consisterà nel capire quali sono i modi per poter gestire i dati su questo strumento e come servirsene ogni

volta che sia necessario, come poter avere a portata di "Tablet PC" tutti i dati utili per il proprio lavoro, guardando gli aspetti di prestazione ma anche alla sicurezza e alla persistenza di questi a lungo termine. L'esempio che presenterò nel mio lavoro, ProClara, è un'applicazione per iPad che è in grado di tenere in memoria i dati relativi a tutti i pazienti che possono essere in cura da uno o più dottori. Ho dovuto affrontare i punti di usabilitá, velocità e sicurezza di tutti i dati, infatti, il Tablet PC, oltre ad essere uno strumento comodo e molto trasportabile, ha anche una potenza molto inferiore a quella di un normale personal computer ed è anche molto più soggetto a danni accidentali.

Attualmente iOS (il sistema operativo installato su iPad) supporta due modalità di salvataggio dei dati: NSUserDefault e Core Data. La prima, è una classe[1] di base del framework[2] Foundation di Cocoa, ovvero la base del sistema operativo. Questa classe serve per salvare le preferenze (impostazioni) della propria applicazione e, solitamente, non viene utilizzata per contenere grandi quantità di dati. La seconda é un framework indipendente, pensato per la creazione e la gestione di un proprio database all'interno dell'applicazione. A prima vista la decisione su quale delle due modalità scegliere sembrerebbe scontata, ma prima di approfondire questo punto preferisco andare a testare veramente quale delle due soluzioni é più adatta e a quali compromessi. Oltre a decidere come salvare i dati, mi soffermerò molto sullo studio di come garantire la sicurezza di questi dati all'interno di un Tablet PC: qual'e' la logica di persistenza pensata da Apple per questo dispositivo? come può essere sia facilmente utilizzabile che sicuro e veloce?

[1]Nella programmazione orientata ad oggetti, una classe è un costrutto che viene utilizzato come modello per creare istanze di se stesso, oggetti di classe, gli oggetti istanza o semplicemente oggetti.

[2]In ambito di programmazione, il framework è una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un framework c'è sempre una serie di librerie di codice utilizzabili, spesso corredate da una serie di strumenti di supporto allo sviluppo del software ideati per aumentare la velocità di sviluppo del prodotto finito.

Capitolo 2

Il Tablet PC

2.1 Che cos'è il Tablet PC

Il Tablet PC è un normale computer portatile caratterizzato dalla presenza di uno o più digitalizzatori che permettono all'utente di interfacciarsi con il sistema direttamente sullo schermo mediante una penna e, in alcuni modelli, anche con le dita. Nel primo caso si parla di digitalizzatori attivi, dispositivi che vengono posti dietro lo schermo o sopra di esso. Essi proiettano un debole campo magnetico che lo stilo, ovvero la penna speciale, utilizza per inviare un feedback al digitalizzatore il quale è in grado di capire la posizione esatta della punta dello stilo e la pressione esercitata dalla penna sullo schermo. Non utilizzando un vero e proprio schermo sensibile al contatto, l'utente può appoggiare la sua mano sullo schermo e scrivere con naturalezza. Nel secondo caso, invece, si parla di digitalizzatori passivi, utilizzati in particolar modo nei modelli di Tablet PC economici o destinati a lavori specializzati. Possiamo distinguere digitalizzatori di tipo resistivo che necessitano della pressione del dito o dello stilo sullo schermo, e di tipo capacitivo che si attivano al semplice sfioramento. Con l'avvento del fenomeno "multitocco", le case produttrici hanno inserito digitalizzatori capaci di riconoscere quattro punti di contatto. La forma di un Tablet PC

ricorda quella di una lavagnetta da scolaro: piatto, sottile, leggero e con pochi tasti disposti su una cornice che racchiude il display LCD. In Italia il Tablet PC è stato sempre considerato un prodotto business fino all'introduzione nel 2007 da parte di HP del primo modello consumer HP Pavilion. A partire dal 2008, con la disponibilità dei processori Intel Atom e con l'esplosione del fenomeno netbook, alcune case produttrici hanno iniziato a produrre modelli convertibili in Tablet a prezzi inferiori comportando un aumento della popolarità di questa piattaforma. Negli Stati Uniti, in Germania e nel Regno Unito il Tablet PC viene utilizzato da molti studenti, professionisti e, in particolar modo, è diffuso nell'ambito ospedaliero e militare.

2.2 Evoluzione dei Tablet PC

A partire dal 2001 Microsoft presentò la sua idea di Tablet PC, un nuovo prodotto che integrava le funzionalità e la portabilità di uno smartphone nella potenza di un netbook. Da allora diverse case produttrici hanno cercato di competere nello stesso mercato. Nel 2007 Amazon propose il suo Kindle, un e-reader che utilizzava la connessione wireless per consentire agli utenti di acquistare, scaricare, sfogliare e leggere libri elettronici, giornali e riviste. Il Kindle presentava uno schermo di sei pollici, era dotato da quattro livelli di scala di grigi con 250 MB di memoria interna e fu venduto esclusivamente tramite internet dal sito statunitense Amazon. Il software era proprietario, girava su Kernel Linux ed era protetto da un sistema DRM. Alla fine del 2009 la maggior parte delle grandi società (Dell, Toshiba, Lenovo, HP tra le più attive) proposero uno o più Tablet PC destinati quasi unicamente al mercato professionale e spesso difficilmente acquistabili dalle persone comuni, come gli studenti. Al lato dei Tablet PC sono poi stati introdotti gli UMPC frutto del progetto Microsoft Origami che però non hanno mai avuto successo. Gli Ultra Mobile PC erano estremamente costosi, infatti poche persone erano disposte a spendere più di 1000 euro per un PC completo ma tascabile. Nel 2009 qualcosa cambiò: uscì Windows7, molto più "leggero" di Windows Vista e finalmente adatto ai processori

Atom; si diffusero le voci di un Tablet PC Apple, voci alimentate dall'enorme successo dell'Apple IPhone. La notizia giunse alla Microsoft e Steve Ballmer, al CES di inizio Gennaio 2010, pochi giorni prima di una conferenza Apple, presentò in modo veloce un vago Tablet PC marchiato HP e altri modelli secondari. Al seguito della presentazione fatta al CES, molti produttori iniziarono a progettare una serie di dispositivi piccoli e leggeri, con schermo resistivo che entrarono nel mercato nei primi sei mesi del 2010.

2.3 iPad: la storia

iPad è stato presentato da Steve Jobs il 27 gennaio 2010. La prima versione di iPad, dotata solo della connessione con antenna Wi-Fi, è stata venduta ad Aprile 2010, mentre la versione dotata anche dell'antenna 3G è stata introdotta nel Giugno 2010. Durante questo anno sono stati venduti oltre 14 milioni di iPad che rappresentano il 75% dei Tablet PC venduti sul mercato in quel periodo. iPad è stato presentato da Steve Jobs come un dispositivo nuovo, non è pensato per sostituire un PC; é destinato a riempire quello spazio di mercato compreso tra gli Smartphone e i Pc ma non deve sostituire né un mercato né l'altro. Tante persone all'uscita di questo prodotto hanno avuto molti dubbi riguardo all'utilizzo di questo dispositivo che non si poteva definire "nuovo", in quanto altri Tablet PC erano già presenti sul mercato. Di fatto, grazie ad Apple, il mercato dei Tablet PC ha avuto un incremento esponenziale poiché, come già era accaduto con l'IPhone, Apple ha "reinventato" un dispositivo. Il successo fu enorme grazie alle decine di migliaia di applicazioni che furono sviluppate, all'interfaccia studiata e progettata in modo perfetto e a una facilità d'uso estrema che caratterizzava il prodotto. Inoltre, quello che più colpi' le persone fu la reattività dello strumento nel rispondere ai comandi dell'utente e la sua velocità complessiva. Si rilevarono anche degli aspetti negativi legati in particolar modo alla mancanza di porte di espansione, allo schermo difficilmente visibile in ambienti esterni e alle temperature di esercizio, che ne impedivano alcuni usi non comuni. Il 25 Marzo

2011 sbarcò in Italia l'iPad 2 , il nuovo prodotto era caratterizzato da maggiori prestazioni, un design migliorato per renderlo più maneggevole all'uso e un prezzo identico rispetto al modello precedente. Con questa strategia legata al prezzo, Apple diede un'altra forte scossa al mercato dei Tablet PC: i concorrenti promuovevano i loro prodotti sottolineando le caratteristiche tecniche migliori rispetto ad iPad ma, in questo modo, essi sono stati costretti a rivedere la progettazione di tali prodotti ed i relativi prezzi sul mercato. Apple é riuscita a mantenere i prezzi invariati rispetto ai precedenti modelli di prodotto investendo cinque miliardi di dollari in partnership con i principali produttori asiatici di schermi LCD e memorie, ottenendo così prezzi nettamente inferiori a quelli di mercato, passando il risparmio sul consumatore e mantenendo tuttavia un margine di quasi il 30%.

2.4 iPad: specifiche tecniche

	IPad1	iPad2
CPU	Apple A4 1 GHz	Apple A5 1 GHz Dual Core
Memoria RAM	256 MB DRAM	512 MB DRAM
Display	9,7" Led risoluzione: 1024x768	9,7" Led risoluzione: 1024x768
Capienza	16, 32 o 64 GB	16, 32 o 64 GB
Dimensioni	24,3 x 19,0 x 1,3 cm	24,12 x 18,57 x 0,88 cm

2.5 iPad: architettura

A un anno di distanza dall'uscita del primo iPad, non abbiamo ancora la certezza della reale architettura utilizzata al suo interno. L'iPad é stato aperto e smontato per poter studiare la sua struttura e i componenti utilizzati. Apple non espone questo tipo d'informazioni perché la sua politica e forza aziendale gioca sull'avere un oggetto unico sul mercato, inimitabile. Ultimamente infatti, molte aziende hanno

cercato d'imitare i suoi prodotti, iPhone in primis, la recente diatriba aziendale tra Apple e Samsung ne é un esempio. Ad oggi esistono due modelli di iPad: iPad 1 e iPad 2; in entrambi i modelli sono stati utilizzati materiali naturali tentando di limitare l'uso di materiali chimici. Il telaio é costituito da un blocco unico in alluminio che gli fornisce un'alta resistenza e poco peso, lo schermo é un pannello LCD a colori da 9.7" e risoluzione 1024x768, il tutto ricoperto da una lastra unica in vetro. Per quanto riguarda la componentistica interna, i due modelli sono differenti. [2]Per quanto riguarda il processore e la memoria RAM, per entrambi i modelli di iPad, non abbiamo informazioni certe. Il sito più conosciuto e affidabile in questo tipo di operazioni si chiama: www.ifixit.com, che dopo aver smontato in ogni componente questi dispositivi ci svela alcune interessanti informazioni. Per quanto riguarda il primo modello iPad 1, ci riporta che: Il processore utilizzato si chiama A4, è un Packet-on-Package (PoP), con almeno tre strati di circuiti uno sopra l'altro, un microprocessore da 1GHz e due moduli di memoria da 128Mb tutti inseriti insieme in un sottile PoP". Questo processore rappresenta un'importante innovazione, infatti Apple invece di comprare un ARM off-the-shelf (che equivale a modelli system-on-a-chip come Tegra di Nvidia e Qualcomm Snapdragon, ampiamente usati in smartphone, come il Google Nexus One), é andata oltre e ha sviluppato un proprio chip chiamandolo A4 con frequenza di 1GHz. Presumibilmente, questo sviluppo è stato aiutato dall'acquisto, da parte di Apple dell'azienda PA Semiconductor, una piccola azienda californiana di progettazione chip, per $ 278M nel 2008.

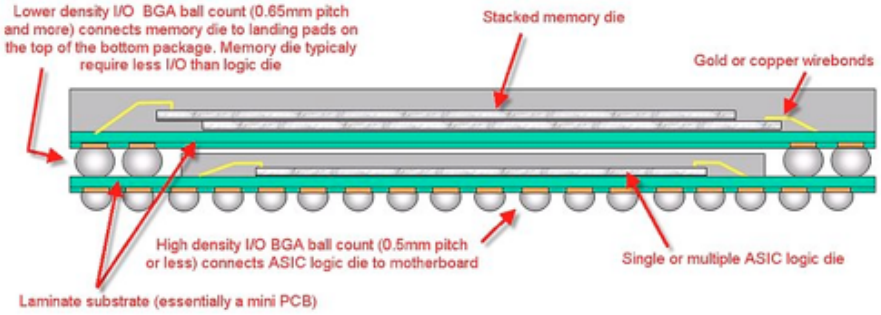
La memoria RAM di iPad si trova all'interno del package del processore A4, la conferma di questo viene da due fonti principali che hanno analizzato il processore a raggi X: Chipworks e X-ray che hanno rilevato due strati di RAM Samsung i quali forniscono 256 MB di memoria; questo fa presumere che anche il processore sia fornito direttamente da Samsung.

[3]**Packet-on-Package:** questa tecnica utilizzata, consiste nel aggiungere sopra al microprocessore alcuni strati di memoria. Questi strati di memoria comunicano con la CPU attraverso pin, esattamente come il processore comunica con la motherboard.

Figura 2.1: CPU iPad.

Questa tecnica é più costosa ma si ottengono diversi vantaggi: meno richiesta I/O, riduzione di rumori e di energia utilizzata, maggiori velocità di trasferimento dati e risparmio di spazio. Nel secondo modello, iPad 2, troviamo il processore A5 che utilizza la stessa tecnica di PoP ed é un processore dual core. Vengono inseriti due processori A4 in parallelo e due blocchi da 256Mb di memoria RAM, le prestazioni sono quindi esattamente raddoppiate.

Figura 2.2: Packet-on-Package.



Capitolo 3

iOS e la gestione dei dati

[4]Come per Mac OS X, in iOS il metodo principale per il salvataggio dei dati é il framework Core Data. Tuttavia esiste un altro sistema chiamato NSUserDefaults che, come preannuncia intuitivamente il nome, è una classe utilizzata per poter salvare le impostazioni personali dell'utente e che cambia a seconda delle applicazioni. Per ogni applicazione considerata, si può quindi dire che NSUserDefaults coincide con le impostazioni personali o preferenze che si salvano al suo interno. NSUserDefaults è stata pensata per poter salvare ad esempio le credenziali per accedere a un social network, oppure per salvare le ricerche recenti del browser Safari.

3.1 NSUserDefaults

NSUserDefaults, può quindi essere definita come una serie di dati scelti dall'utente, serve ad ogni applicazione per salvare eventuali impostazioni che vogliamo conservare lungo (es. quando impostiamo una determinata immagine come sfondo, questa viene salvata per essere ripresentata all'apertura successiva dell'applicazione). Nelle NSUserDefaults possiamo salvare i seguenti tipi di dato: NSString, NSDictionary,

NSArray, NSData, NSDate e NSNumber. Per poter salvare altri tipi di dato occorre serializzarli usando NSCoding. Il sistema di salvataggio delle impostazioni NSUserDefaults, utilizza la tecnica delle Property Lists. Le Property Lists sono delle liste in formato XML nelle quali, per salvare un oggetto, si crea una chiave alla quale il nostro oggetto è associato. In altre parole si usa un metodo chiamato Key Value Object, che consiste proprio nel salvare un'oggetto associandolo ad una chiave univoca; la cosa più importante è creare ogni volta una chiave diversa in modo da non avere due chiavi che puntano allo stesso elemento. Una volta effettuato un salvataggio all'interno delle NSUserDefaults, viene generato un file in formato .plist che risulta molto leggibile in quanto questo file crea una tabella con due colonne: chiave e valore. NSUserDefault è dunque una classe che permette di astrarre la complessità che risulterebbe dal dover generare un file XML e scrivere al suo interno. I metodi di gestione delle NSUserDefaults sono molto intuitivi e semplici da utilizzare. Il vantaggio delle NSUserDefaults è che salvare dati semplici risulta molto immediato e permette anche di organizzare dati senza limiti di dimensioni, per questo motivo ho provato a generare un database in esse per provarne l'efficenza.

Figura 3.1: Esempio di NSUserDefault.

```
NSMutableDictionary *dictionaryDoctor = [[NSMutableDictionary alloc] init];
   NSUInteger nDoc=0;
   for (nDoc=0; nDoc<20; nDoc++) {
       [dictionaryDoctor setObject:[NSString stringWithFormat:@"Dottore n:%i",nDoc]
                           forKey:[NSString stringWithFormat:@"%@ %i",DOCTOR_KEY,nDoc]];
       NSUInteger nPat = 0;
       for (nPat=0; nPat<200; nPat++) {
           [dictionaryDoctor setObject:[NSString stringWithFormat:@"Paziente n:%i",(nPat*nDoc)]
                            forKey:[NSString stringWithFormat:@"%@ %i",PATIENT_KEY,(nPat*nDoc)]];
       }
    }

   [[NSUserDefaults standardUserDefaults] setObject:dictionaryDoctor
                                             forKey:DICTIONARY_UNIQUE_KEY];
```

Questo piccolo frammento di codice creato utilizzando i metodi di gestione delle NSUserDefaults, genera un database in modo molto semplice e veloce, infatti, per creare 1804 records o elementi del database, impiega circa 27ms. I dati inseriti con-

sistono in stringhe che rappresentano nomi e cognomi di dottori e pazienti ipotetici. Come detto precedentemente, le NSUserDefaults salvano quindi i nostri dati in un file .plist, questo file é esattamente come un file XML e può essere rappresentato come un dizionario. All'inizio del file .plist troviamo la chiave ¡dict¿ ovvero il nome del nostro file. Successivamente troviamo una lista di paramentri ¡key¿ ovvero la lista di chiavi che abbiamo inserito per accedere ai dati. Per ogni ¡key¿ troviamo il valore assegnato alla chiave ovvero ¡string¿, ¡array¿ oppure numeri. Ogni dato viene visualizzato in riferimento alla sua chiave unica, al tipo di dato e al suo valore. Il file è molto leggibile e anche la sua realizzazione, in questo caso, si è rivelata semplice e abbastanza immediata. Se come creazione e salvataggio le NSUserDefaults sembrano molto vantaggiose, la ricerca e la gestione di tutto il database, risulta un po' più ardua, infatti, per effettuare una ricerca e ottenere l'oggetto cercato (paziente o dottore), si richiede l'estrazione di tutta la base dati e successivamente una ricerca su di essa. Viceversa, per aggiungere un nuovo dato nel database, occorre nuovamente estrarre tutta la base dati, aggiungere l'oggetto in essa e reinserire tutto.

3.2 Core Data

[5]Un secondo e più consigliato metodo di salvataggio dati di grandi dimensioni che si vuole analizzare è Core Data. Questo framework è stato introdotto con il sistema operativo Mac OS X 10.5 Tiger ed è la soluzione fornita da Apple per la persistenza dei dati. L'idea di Apple é fornire agli sviluppatori un framework molto potente ma allo stesso tempo semplice da utilizzare. La progettazione parte infatti dal disegno di un semplice grafico per poi estendersi a fondo e controllare ogni particolare del database. Core Data offre un insieme di classi per la creazione e gestione a lungo termine del proprio database; le tre classi principali che permettono queste operazioni sono: NSPersistentStoreCoordinstor, NSManagedObjectModel e NSManagedObjectContext. Questi tre elementi lavorano insieme per leggere e salvare i

nostri oggetti che sono di tipo NSMangedObject il quale rappresenta le nostre reali entità del database.

NSMangedObject è un oggetto generico che può assumere dinamicamente le sembianze di una qualsiasi entità e attraverso di esso possiamo impostare gli attributi dell'entità e le relazioni fra più entità. Si può immaginare come una tabella SQL, dove il nome della tabella è lo stesso nome dell'entità e le colonne sono i suoi attributi.

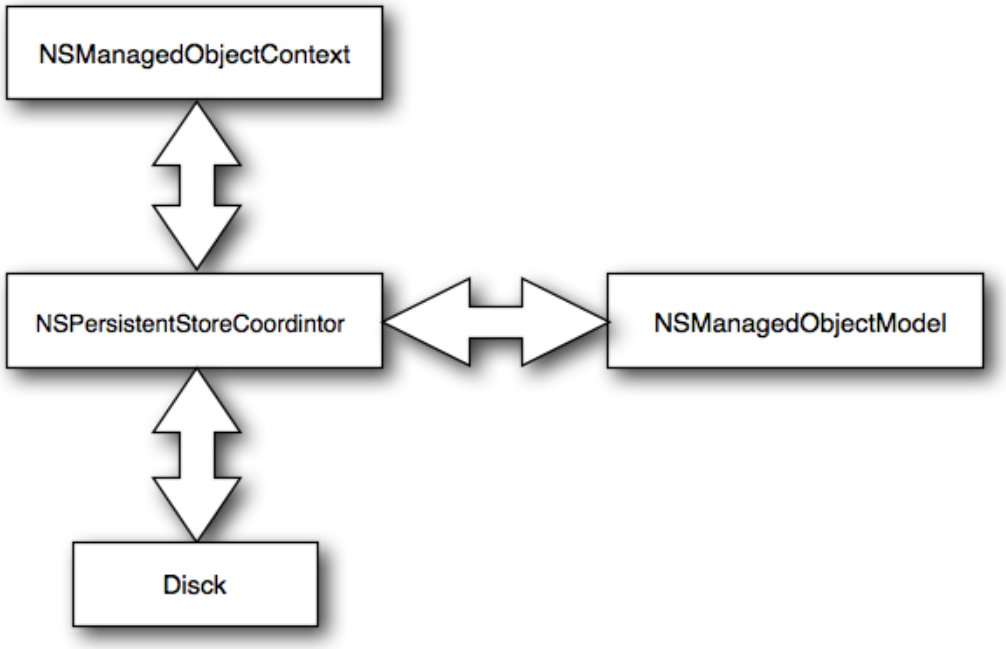
NSManagedObjectContext si può considerare come un contenitore dove vengono caricati o creati oggetti per poter essere manipolati e successivamente salvati. Tutti gli oggetti del database passano per questo livello per poi essere salvati nel database o cancellati; é un livello ulteriore tra il database e le interazioni con l'utente. Questo strato garantisce più sicurezza nel mantenere intatto l'oggetto reale nel momento in cui si effettuano su di esso delle operazioni (modifica, salvataggio o eliminazione).

NSManagedObjectModel può essere considerata come la copia del nostro database salvato. Di fatto il nostro database viene compilato in un file in formato .momd (ManagedObjectModel), questo file viene caricato dall'applicazione e al suo interno sono salvate tutte le eventuali entità e relazioni del database.

NSPersistentStoreCoordinstor è la classe che si occupa del salvataggio del database su disco. In questa fase si decide infatti il nome e il tipo di database che si vuole salvare. La Apple ha voluto creare una classe apposita per il salvataggio e il caricamento del database che permette di gestire più richieste in contemporanea. Questa particolare classe, si occupa di più richieste concorrenti in modo da non avere conflitti, nel momento in cui, queste richieste, riguardano il salvataggio e la cancellazione del medesimo oggetto.

Per capire meglio la struttura delle classi utilizzare in Core Data ho creato un diagramma:

Figura 3.2: Core Data Stack.



Ho voluto mostrare uno stack[1] per capire meglio i livelli d'astrazione che vengono utilizzati in Core Data, per dividere gli oggetti gestiti ad alto livello, dalla memoria fisica dove vengono salvati.

[1]In informatica, il termine stack o pila viene usato in diversi contesti, in questo caso definisce la gerarchia di collegamento tra le principali classi di Core Data.

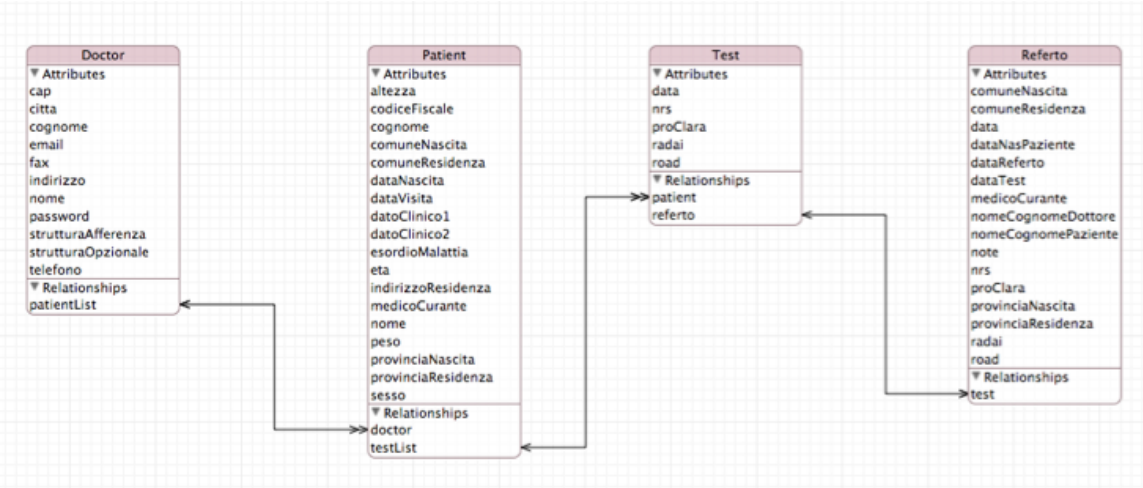
Capitolo 4

Core Data: progettazione e sviluppo.

4.1 Costruzione del grafico: Data Model.

[6]Core Data propone un metodo semplice ed intuitivo per la creazione di un database. Possiamo creare un'entità, tutti i suoi attributi e le possibili relazioni, il tutto in maniera grafica. Per fare questo, dal menù di Xcode si procede: File / New File / Core Data / Data Model.

Figura 4.1: Data Model.



Quest'immagine rappresenta il grafico del Data Model, più precisamente rappresenta graficamente il database utilizzato nell'applicazione ProClara. Ogni tabella rappresenta un'entità, all'interno della tabella si vedono elencati gli attributi e le possibili relazioni. Seppur questo metodo possa sembrare semplice e banale, al suo interno possiamo trovare tutto ciò di cui abbiamo bisogno per effettuare una progettazione di grandi e complessi database. Una schematizzazione così semplice, aiuta il programmatore risparmiando sul tempo evitando di scrivere righe di codice ed offre una visione più chiara dell'intero database; utile in particole modo quando dopo lungo tempo si ha necessita di tornare sul progetto, si può interpretare meglio la gerarchia di classi dell'applicazione.

4.1.1 Diagramma E-R

Questa fase della creazione di un database è chiamata: costruzione del diagramma E-R, ovvero diagramma entità - relazione. Il diagramma E-R serve a suddividere il problema in tanti piccoli sotto problemi in modo da rendere più semplice la sua soluzione. Viene utilizzato quasi sempre prima di costruire un database per trasformare

l'analisi di un problema in uno schema logico ed è formato da un insieme di entità e di relazioni. **L'entità** rappresenta una classe d'oggetti, cose o persone con proprietà in comune e con un'esistenza indipendente dalle altre entità considerate. Nel mio diagramma, ad esempio, dottori, pazienti, test e referto sono entità. **Le relazioni o associazioni**, rappresentano un legame tra una o più entità. Nell'esempio, la relazione è indicata con una freccia che collega un'entità ad un'altra. Le relazioni possono collegare una entità con un'altra o più entità, infatti un dottore può avere in cura uno o più pazienti. Per indicare questo tipo di legame (To-Many) viene utilizzata una doppia freccia verso l'entità in questione. Esistono diversi metodi per risolvere questo tipo di problemi e arrivare a un diagramma E-R, i due metodi principali sono: top-down e bottom-up.

4.1.2 Metodo Top-Down

Come suggerisce il nome, con questo metodo partiamo dalla visione generale del problema (Top) e passo dopo passo scendiamo nel particolare (Down) suddividendo il problema generale in macro problemi. In questo modo, si ha una visione a piramide del problema e, quando i sotto problemi non possono più essere suddivisi, il processo è terminato. Questo metodo è molto efficace per risolvere problemi complessi ma obbliga a conoscere in anticipo tutti i dettagli del problema da analizzare.

4.1.3 Metodo Bottom-Up

Al contrario di Top-Down, con il metodo Bottom-Up partiamo dal basso, ovvero da risolvere prima tutti i micro-problemi. Dopo aver risolto tutti i piccoli problemi del nostro progetto, si procede collegandoli tra loro così da formare la soluzione finale. Questo metodo risulta un po' più complesso quando si deve analizzare un problema molto articolato, poiché bisogna saperlo scomporre in maniera adeguata; tuttavia, in

questo modo risulta più semplice aggiungere o modificare dettagli al nostro problema anche mentre si sta lavorando su di esso.

Nel caso di ProClara, ho utilizzato un approccio di progettazione Bottom-Up, infatti, la costruzione del Data Model è avvenuta aggiungendo un'entità alla volta e successivamente collegando queste entità tra loro attraverso le relazioni. Core Data permette di salvare più versioni, ovvero più modifiche successive del Data Model ma è possibile avere una sola versione o modifica corrente. Nel progetto di ProClara ho sviluppato sei versioni del Data Model: quattro sono dovute al fatto che l'applicazione è stata scritta con il metodo Bottom-Up, le altre due modifiche sono state aggiunte per concludere il progetto finale a seguito di un cambiamento rispetto al progetto originale di partenza. Infatti, capita molto spesso che colui che richiede il progetto non abbia le idee chiare su ciò che vuole esattamente, oppure, davanti alla realizzazione di esso, si rende conto di voler modificare l'idea iniziale. Quest'aspetto, relativo alle modifiche successive del database, è molto importante e prende il nome di "Versioning" di cui si parlerà in un capitolo successivo.

4.2 Creare un'entità: NSManagedObject

NSManagedObject rappresenta una classe generica che implementa tutti i metodi di base richiesti da un oggetto nel modello dati di Core Data. Dopo aver aggiunto le entità al grafico, nel progetto si aggiungono tante sottoclassi di NSManagedObject quante sono le entità sul grafico. Ad ogni sottoclasse di NSManagedObject viene associata una descrizione (un'istanza di NSEntityDescription) che fornisce i meta-dati relativi all'oggetto (il nome del soggetto che l'oggetto rappresenta, i nomi dei suoi attributi e le possibili relazioni) e un'istanza della classe NSManagedObjectContext che tiene traccia delle modifiche per l'oggetto. Per alcuni aspetti, un oggetto NSManagedObject agisce come un dizionario: si tratta di un oggetto contenitore generico che fornisce in modo efficiente l'archiviazione per le proprietà defi-

nite dal suo oggetto NSEntityDescription associato; inoltre necessita anche di un'istanza di NSManagedObjectContext che ne descrive il contesto dove viene gestito l'oggetto.

NSManagedObject fornisce il supporto per una gamma di tipi comuni come valori per i suoi attributi, tra cui stringhe, date e numeri. A volte, tuttavia, si desidera utilizzare i tipi che non sono supportati direttamente, come i colori e le strutture C. Per esempio, in un'applicazione grafica è possibile definire un ente rettangolo che ha gli attributi di colore che sarebbero un'istanza di NSColor e un NSRect struct. Per risolvere questo problema è possibile utilizzare un attributo trasformabile di classe NSTransormableData. Gli attributi standard, per il nostro oggetto di Core Data, possono essere sottoclassi di: NSString, DSDate, NSInteger e NSFloat.

Ecco un esempio di un'entità a livello di codice:

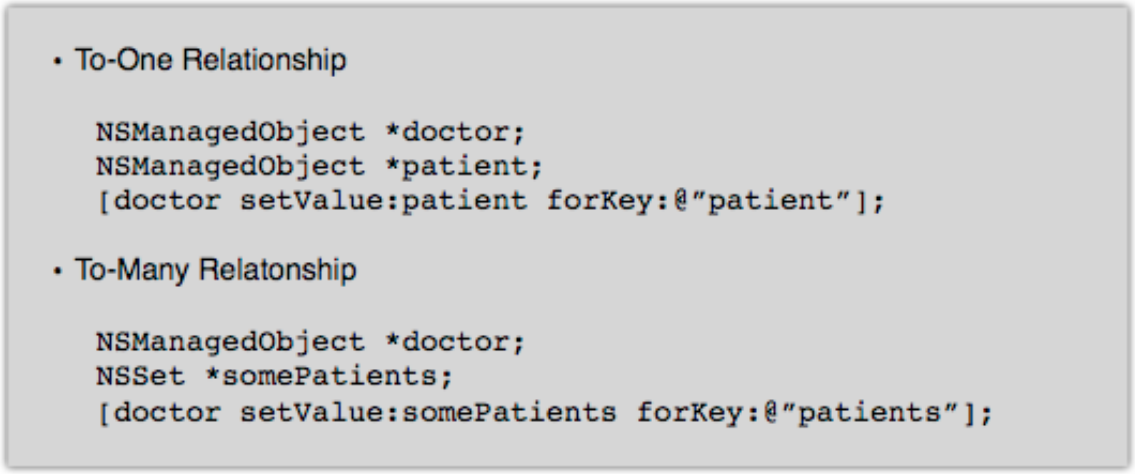
```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Doctor : NSManagedObject {
}
@property (nonatomic, retain) NSString * nome;
@property (nonatomic, retain) NSString * cognome;
@property (nonatomic, retain) NSString * password;
@property (nonatomic, retain) NSSet * patientList;

+(Doctor*)doctor;
@end
@interface Doctor (CoreDataGeneratedAccessors)
- (void)addPatientListObject:(NSManagedObject *)value;
- (void)removePatientListObject:(NSManagedObject *)value;
- (void)addPatient:(NSSet *)value;
- (void)removePatient:(NSSet *)value;
@end
```

Con questo codice, abbiamo collegato la tabella disegnata nel Data Model, al nostro progetto, dove possiamo manipolare come preferiamo gli oggetti dichiarati. Tutti gli attributi aggiunti nel Data Model e le relazioni, vengono dichiarate ognuna con il proprio tipo. Per semplicità di sviluppo, da Data Model, ho impostato dei valori di default per ogni attributo: ogni volta che viene creato un nuovo dottore, per sicurezza, vengono automaticamente impostati dei valori di default per non avere oggetti nulli che creerebbero problemi nel database.

Come detto in precedenza, NSManagedObject è contenitore di NSEntityDescripion, questa classe, incorpora tre classi che descrivono l'entità: NSAttributeDescription, NSRelationDescription e NSFetchedPropertyDescription; ognuna di esse è un'estensione della classe NSPropertyDescription, sono le proprietà che può avere la nostra entità. Gli attributi sono istanze della classe NSAttributeDescription, la classe salva il tipo di dato e l'eventuale valore di default se impostato. NSRelationshipDescription definisce le relazioni tra entità: possiamo indicare a quale entità si riferisce la relazione, se la relazione è anche inversa se è di tipo To-Many e le regole di cancellazione (si può indicare che quando l'entità viene cancellata, vengano cancellate anche le entità collegate ad essa). Le relazioni possono essere dunque, di due tipi: To-One, To-Many; a livello di codice sono molto simili ad un attributo, cambia il tipo di dato in base al tipo di relazione:

Nel caso la relazione non sia To-One, il caso è molto semplice, si salva un attributo di tipo NSManagedObject che punta all'entità con cui si ha una relazione. Nel primo caso successivo viene mostrata una relazione di tipo To-Many: l'attributo patientList è un'istanza della classe NSSet che definisce un set generico di oggetti, questi oggetti saranno istanze di NSManagedObject, formano un gruppo di entità associate. Per gestire questo tipo di relazioni, ho creato i "CoreDataGeneratedAccessors", sono metodi che vengono aggiunti ai metodi normali di NSManagedObject, aiutano a gestire i gruppi di oggetti nelle relazioni. Nel caso di ProClara, viene utilizzata una relazione di tipo To-Many, infatti a un dottore viene associato un gruppo di pazienti; inoltre ho definito come regola di cancellazione, per la relazione, la modalità "cascade", con

Figura 4.2: Relationship.



questa modalità se un dottore viene eliminato dal database vengono eliminati a loro volta tutti i pazienti associati ad esso; questo per evitare che rimangano oggetti senza padre nel database.

4.3 Collegamento dell'entità nel progetto: NSManagedObjectContext

Questa classe viene utilizzata in Core Data, come livello ulteriore per gestire oggetti di tipo NSManagedObject: quando vogliamo salvare un oggetto su disco, leggere un oggetto in memoria oppure crearne uno nuovo. Infatti, per accede alla memoria, si controlla l'esistenza di un database, utilizzando NSPersistentStoreCoordinator, se la ricerca ha successo si restituisce un oggetto associato al database trovato. NSManagedObjectContext è alla prima posizione dello "stack" mostrato in precedenza, poiché è la classe più utilizzata: basta pensare a quante volte può servire salvare, modificare o creare oggetti nel database. Per questo motivo potrebbe verificarsi un accesso contemporaneo, chiamato multi-thread, alla stessa risorsa. Per evitare conflitti di

concorrenza si accede al database mediante un unico NSPersistentStoreCoordinator che, come vedremo in seguito, organizza gli accessi al database.

Creando un'istanza di NSManagedObjectContext, si va a lavorare si di un unico spazio oggetto ovvero sulla stessa porzione di memoria. Come detto in precedenza, la sua responsabilità principale è quello di lavorare su di un insieme di NSManagedObject; questi formano un gruppo di oggetti correlati tra loro, che posso essere salvati su di un NSPersistentStoreCoordinator. Ogni istanza di NSManagedObject viene gestita da un solo NSManagedObjectContext, eventuali copie dello stesso oggetto possono essere gestite da context differenti.

```
- (NSManagedObjectContext *) managedObjectContext {

    if (managedObjectContext != nil) {
        return managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self
        persistentStoreCoordinator];
    if (coordinator != nil) {
        managedObjectContext = [[NSManagedObjectContext alloc] init];
        [managedObjectContext setPersistentStoreCoordinator:
            coordinator];
    }
    return managedObjectContext;
}
```

Analizzando il codice, se esiste già' un'istanza non nulla di NSManagedObjectContext il metodo ritorna immediatamente quell'istanza ed esce direttamente, altrimenti se esiste un database possiamo creare l'istanza che verra' successivamente ritornata ad ogni chiamate in futuro.

Per capire meglio lo strato ulteriore che NSManagedObjectContext crea tra NSManagedObject e il database reale andiamo ad analizzare il metodo d'inizializzazione di un NSManagedObject:"initWithEntity:insertIntoManagedObjectContext:". Per poter creare un nuovo oggetto di questo tipo, occorre avere un'istanza di NSEntityDescription e una di NSManagedObjectContext. In questo modo il nostro oggetto viene identificato come un pacchetto che ingloba un'entità in un contesto.

```
+(Doctor*)doctor{
  NSManagedObjectContext *managedObjectContext = nil;
  managedObjectContext = [[DataSource sharedServices]
     managedObjectContext];
  NSEntityDescription *entityDescription = nil;
  entityDescription = [NSEntityDescription entityForName:@"Doctor"
     inManagedObjectContext:managedObjectContext];
  Doctor *doctor = nil;
  doctor = [[Doctor alloc] initWithEntity:entityDescription
     insertIntoManagedObjectContext:managedObjectContext];
  return [doctor autorelease];
}
```

Questo codice: carica l'istanza del context (viene utilizzata una classe chiamata Data Source), crea un'istanza di entity description con nome Doctor; dopo di ché inizializza un oggetto di tipo "Dottore".

4.4 Salvataggio e gestione della memoria: NSManagedObjectModel

NSManagedObjectModel può essere considerata come la versione compilata del database creato con Xcode in versione binaria. Infatti nel processo di compilazione

del progetto: il file .xcdatamodel viene compilato in un file .momd che verrà salvato nella cartella "Resources" nel "bundle" dell'applicazione.

```
- (NSManagedObjectModel *)managedObjectModel {

    if (managedObjectModel != nil) {
        return managedObjectModel;
    }
    managedObjectModel = [[NSManagedObjectModel mergedModelFromBundles:
        nil] retain];
    return managedObjectModel;
}
```

Questo codice crea l'istanza di NSManagedObjectModel utilizzata in ProCLara, non servirà modificarlo in futuro, rimarrà invariata anche con le future versioni del database. Si parte determinando la posizione del file .momd: utilizzando NSBundle ed il suo metodo pathForResource: per trovare il percorso del nostro database inseguito usato con NSURL. Con questo NSURL, possiamo dunque creare l'istanza di NSManagedObjectModel, utilizzando il suo metodo initWithContentsOfURL.

Comunemente non è necessario accedere all'istanza di NSManagedObjectModel dopo che tutto lo "stack" di Core Data è stato inizializzato, può dipendere dalla complessità del progetto. Ad esempio, se volessimo avere una descrizione di tutte le entità presenti nel database, grazie ai metodi presenti in NSManagedObjectModel, potremmo conoscere dinamicamente: relazioni, attributi ed entità esistenti. Tuttavia non è la scelta proposta da Core Data, è più utilizzata, dagli sviluppatori del progetto stesso, come via più breve per accede al database. Un altro motivo per cui potrebbe essere necessario accadere direttamente a NSManagedObjectModel è per la gestione del versioning. Quando è necessario scrivere o sovrascrivere una versione del database, può essere comodo interrogare NSManagedObjectModel, per risolvere eventuali conflitti logici tra versioni diverse.

4.5 Persistenza e integrità del database: NSPersistentStoreCoordinator

Questa classe sta alla base dello "stack" di Core Data, è l'oggetto responsabile della persistenza della base dati della nostra applicazione. Core Data offre la possibilità di salvare in tre modalità: NSSQLiteStoreType, NSBinaryStoreType e NSInMemoryStoreType; in ProClara è stato utilizzata la modalità SQLite, che è la modalità più diffusa. La possibilitá di poter salvare in più modalità é un concetto molto importante: anche se modifichiamo la modalità la struttura di Core Data rimane invariata. Se ad esempio decidessimo di sviluppare una applicazione in SQLite e una seconda applicazione su .plist (NSBinaryStoreType) non abbiamo bisogno di studiare concetti nuovi. É la base di Core Data che pensa a come salvare i nostri dati, utilizzando questi sistema embedded inoltre, se ad esempio Apple decidesse di effettuare modifiche: come il cambiamento del filesystem o l'aggiornamento del firmware; la nostra applicazione continuerebbe a funzionare.

Ecco come viene creata l'istanza di NSPersistentStoreCoordinator in ProClara:

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
    if (persistentStoreCoordinator != nil) {
        return persistentStoreCoordinator;
    }
    NSURL *storeUrl = [NSURL fileURLWithPath: [[self
        applicationDocumentsDirectory] stringByAppendingPathComponent: @
        "ProClara.sqlite"]];
  NSDictionary *options = [NSDictionary dictionaryWithObjectsAndKeys:
                    [NSNumber numberWithBool:YES],
                        NSMigratePersistentStoresAutomaticallyOption,
                    [NSNumber numberWithBool:YES],
                        NSInferMappingModelAutomaticallyOption, nil];
  NSError *error = nil;
    persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
        initWithManagedObjectModel:[self managedObjectModel]];
    if (![persistentStoreCoordinator addPersistentStoreWithType:
        NSSQLiteStoreType configuration:nil URL:storeUrl options:options
         error:&error]) {
    NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
    abort();
    }
    return persistentStoreCoordinator;
}
```

Dopo aver controllato la non esistenza dell'istanza, si controlla se la posizione del file del database esiste. Dopo di ché come detto in precedenza, si prende l'istanza di NSManagedObjectModel per poter creare l'istanza del nostro database; da ora accederemo sempre a questa istanza e ad ogni riapertura dell'applicazione potremo caricare il database esistente.

Da notare che, dopo aver allocato e istanziato l'oggetto, viene aggiunto il vero e proprio persistent store con il metodo addPersistentStoreWithType. Quest'ultimo

é definito da: storeType (visto in precedenza), configuration (stringa contenente il nome della configurazione, parametro opzionale), storeURL (percorso del file creato in precedenza), options (dizionario contenente parametri di cui parlerò nel prossimo capitolo) e error (istanza di NSError che viene popolata in caso d'errore, altrimenti viene ritornato nil).

4.6 Aggiornamento delle versioni del database: MappingModel

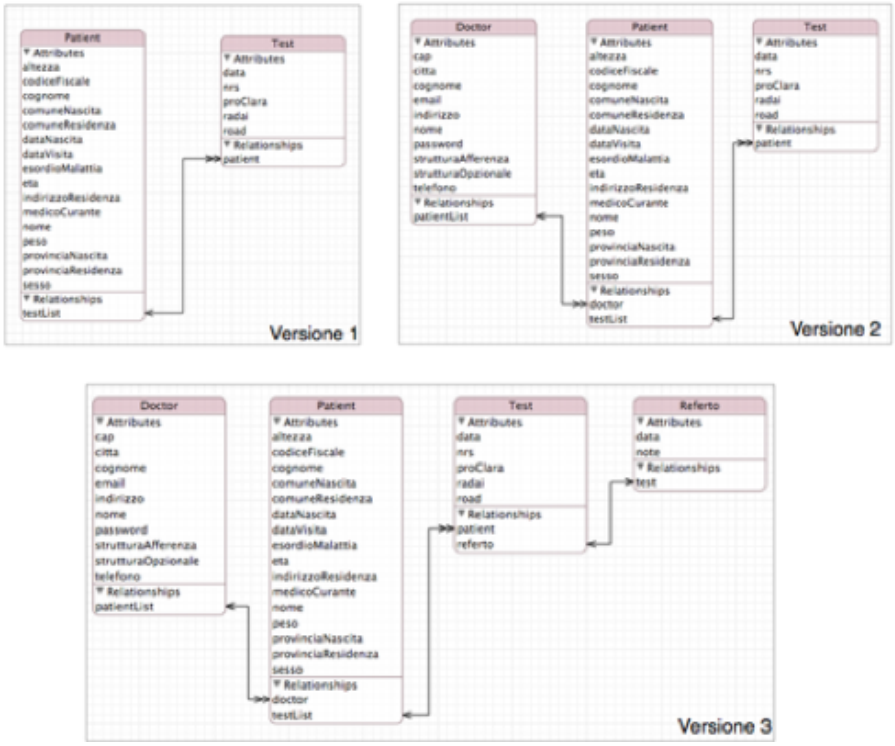
In ogni progetto in cui ho lavorato, mi è capitato di dover correggere le idee iniziali nel corso dello sviluppo. Riuscire a pensare e tutte le possibili funzionalità di un applicativo, o immaginarsi ogni percorso che l'utente può percorrere è quasi impossibile. Per questo motivo è una buona regola da programmatore predisporre l'applicativo, ed in particolare il database, per poter essere modificato, come ad esempio: aggiungere un attributo ad un'entità, modificate il nome di un'entità e possibili modifiche alle relazioni. Questi cambiamenti posso trasformare notevolmente la Base Dati fino al punto da non riconoscerlo più come prima. A questo proposito, la *migrazione* aiuta l'applicativo ad affrontare i cambiamenti della Base Dati, fornisce regole per poter gestire tutti i cambiamenti del database tra le varie versioni.

4.6.1 Il concetto di "versioning"

Core Data non permette la modifica di un database all'interno di un'applicazione già pubblicata su AppStore; Apple ha pensato un modo per tener traccia di tutte le possibili versioni della base dati che possono essere installate su ogni device. Per spiegare meglio questo concetto, si potrebbe pensare ad un cliente che scarica l'applicativo alla versione 1.0, successivamente non aggiorna alla versione 2.0 ma direttamente alla versione 3.0. Dalla versione 1.0 alla versione 3.0 potrebbero essere state create

o cancellate intere tabelle, l'applicativo deve saper come gestire eventuali relazioni con queste tabelle. Per questo motivo, ogniqualvolta apportiamo modifiche ad un database preesistente in Core Data, occorre congelare la versione corrente e, successivamente, apportare le modifiche richieste. In questo modo si tengono salvate tutte le versioni del database pubblicate; questa tecnica viene chiamata *versioning* ovvero aggiornamento a versioni. Dopo aver effettuato il *versioning* del nostro Data Model, occorre fornire all'applicativo le regole sul comportamento rispetto alle modifiche effettuate: la migrazione. A questo proposito, Core Data possiede già un metodo di default da poter utilizzare nel caso in cui i cambiamenti al database non siano troppo complessi. In ProClara viene utilizzata la migrazione standard, nel mottetto in cui viene caricato il Data Model si aggiunge una serie di parametri opzionali sulle modalità di caricamento: una di queste é NSMigratePersistentStoresAutomaticallyOption (Vedi PersistentStoreCoordinator).

Figura 4.3: Versioning.



In questa immagine ho voluto mostrare vari passi di espansione del database; pezzo dopo pezzo il database viene ultimato e anche il reste dell'applicazione viene scritto gradualmente. Una volta completata la prima versione dell'applicazione, le tecniche di versioni e migrazione, verranno utilizzate per eventuali bugfix o espansioni dell'applicazione.

4.7 Query al DataBase

Per poter interrogare il database, Core Data offre una classe dedicata a questo compito: NSFetchRequest. Questa classe permette di operare su oggetti di tipo NSManagedObject: una volta impostato il contesto (NSManagedObjectContext) e il nome

dell'entità su cui operare, possiamo caricare tutti gli oggetti trovati o conoscerne la quantità.

```
NSFetchRequest * request = [[NSFetchRequest alloc] init];
[request setEntity:[NSEntityDescription entityForName:@"Doctor"
    inManagedObjectContext:self.managedObjectContext]];
[self.managedObjectContext executeFetchRequest:request error:&error];
```

Uno dei parametri che e possibile impostare in NSFetchRequest è il predicate, questo parametro richiede un'istanza della classe NSPredicate. Attraverso questo predicato è possibile filtrare in memoria i risultati della chiamata al database. Vi sono molti modi per impostare un predicato, il metodo più utilizzato è la chiamata di tipo SQL. Si può infatti creare un NSPredicate da una NSString con all'interno un pseudo-codice SQL:

```
NSPredicate *tagPredicate = [NSPredicate predicateWithFormat:[NSString
    stringWithFormat:@"(nome like '%@') AND (cognome like '%@')",nome,
    cognome]];
```

Come si vede dall'esempio, il codice inserito nel NSPredicate non è un vero codice SQL, la sintassi è diversa ma la logica è la stessa. Molti esempi di query si possono trovare nella guida all'interno di xCode. Non é possibile effettuare tutte le operazioni presenti in SQL, in Core Data, a volte potrebbe occorrere effettuare due richieste per operazioni molto complesse.

L'unione di NSFetchRequest e NSPredicate formano uno strumento di ricerca molto potente, sono entrambe classi molto generiche e possono essere utilizzate sia per ottenere record del database ma anche per ottenere informazioni su di esso. Nell'esempio sopra viene chiamato il metodo executeFetchRequest: che ritorna un oggetto di tipo NSArray; senza avere informazioni sul database o senza aver importato l'NSPredicate con un numero massimo di risultati, potrebbe accadere che la chiamata

ritorni un array troppo grande da far andare la memoria un overflow. È consigliato mettere un limite ai risultati di NSFetchRequest, oppure eseguire prima la chiamata countForFetchRequest: che ritorna un integer; quest'ultima è anche molto utile per sapere quante entità o attributi di entità sono presenti nel database[7].

Capitolo 5

PROClara

PROClara é la prima applicazione per iPad sviluppata dall'Apple Development Center Unibo, laboratorio Apple nato dalla collaborazione fra Apple e Alma Mater Studiorum Bologna.

Il progetto é stato reso possibile grazie al coordinamento generale del docente Danilo Montesi[1] ed al consenso all'uso del test Pro-Clara del docente Fausto Salaffi[2]. L'applicazione é stata realizzata da me e dai miei colleghi Michele Foscardi e Francesca Guadagnini.

5.1 Descrizione.

L'applicazione PROClara é stata pensata per poter essere utilizzata sia dal medico che dai pazienti. Questo programma offre la possibilità di eseguire un test di autovalutazione del dolore, composto da 29 domande, da sottoporre ai pazienti in cura.

[1]Professore ordinario presso Alma Mater Studiorum, Università di Bologna. Presidente del corso di studi in informatica, docente in basi di dati

[2]Professore associato presso l'Università Politecnica delle Marche, coordinatore dello studio INDICES

Il test, chiamato PRO-Clara e da esso deriva il nome dell'applicazione, serve a misurare l'attività di malattia in corso di artrite reumatoide. Riuscire a valutare il dolore di un paziente malato di artrite non é facile, il dolore si può manifestare in diverse intensità durante l'arco della giornata e può dipendere da quale parte del corpo e' messa sotto sforzo e a quale movimento si esegue. Questo test é stato creato apposta per interrogare il paziente allo scopo di valutare e localizzare il dolore.

L'applicazione offre al medico la possibilità di aggiungere alla propria rubrica i pazienti compilando tutti i dati personali utili per la cura. Il medico può sottoporre il test al paziente il quale ha la possibilità di eseguirlo con l'assistenza del medico. I risultati del test vengono salvati e il medico ha la possibilità di visualizzarli in qualsiasi momento, anche con l'aiuto di un grafico che sia in grado di mostrare l'andamento medio del risultato di tutti i test eseguiti dal paziente. Il medico ha anche la possibilità di creare un referto che mostra i risultati ottenuti, in questo modo é possibile consegnare al paziente una copia del risultato con eventuali note.

Con questa applicazione, il processo di salvataggio e la ricerca dei dati avvengono molto più velocemente poiché digitali; inoltre il medico ha la possibilità di aver a portata di tablet tutto l'archivio dei suoi pazienti e dei relativi test.

5.2 Realizzazione.

Attraverso Core Data, ho realizzato un database in grado di poter salvare i parametri di medici, pazienti, test e referti. Nel prossimo capitolo approfondirò meglio le fasi intermedie di realizzazione del database. Tutta la parte grafica, l'aspetto User Interface, é stato realizzato tenendo conto che l'applicazione sarebbe stata utilizzata da persone anziane (i pazienti) o medici. L'applicazione deve quindi risultare molto semplice nel suo utilizzo e utilizzare un linguaggio comune, non tecnico. Per questi motivi, é stato scelto di utilizzare uno sfondo scuro con testo chiaro in tutta l'applicazione, per ottenere maggior contrasto e leggibilità.

Capitolo 6

Analisi delle prestazioni attraverso lo standard TPC-C

Per poter stabilire in modo neutro le prestazioni di Core Data, ho adattato un benchmarck riconosciuto come modello standard a livello mondiale su iPad e l'ho eseguito per verificarne le prestazioni.

6.1 TPC

[8]La TPC, Transaction Processing Performance Council, è un'organizzazione non-profit fondata per definire i processi di transazione per database e diffondere dati oggettivi e verificabili per l'industria.

Il termine transazione viene spesso applicato a una vasta gamma di attività e funzioni del computer. Vista come una funzione del computer, una transazione, potrebbe riferirsi a un insieme di operazioni , come ad esempio, le operazioni fatte dal disco in lettura / scrittura, quelle del sistema operativo, o anche per definire trasferimenti di dati da un sistema all'altro.

Il TPC benchmark, oltre ad essere considerato per la misurazione e la valutazione delle prestazioni del computer, viene inteso nel mondo del lavoro come uno scambio commerciale di beni, servizi o denaro. Una transazione tipica, come definito dal TPC, dovrebbe prevedere l'aggiornamento di un database per dati relativi al controllo delle scorte (beni), alle prenotazioni aeree (servizi) o conti correnti bancari (il denaro).

6.2 Standard TPC-C

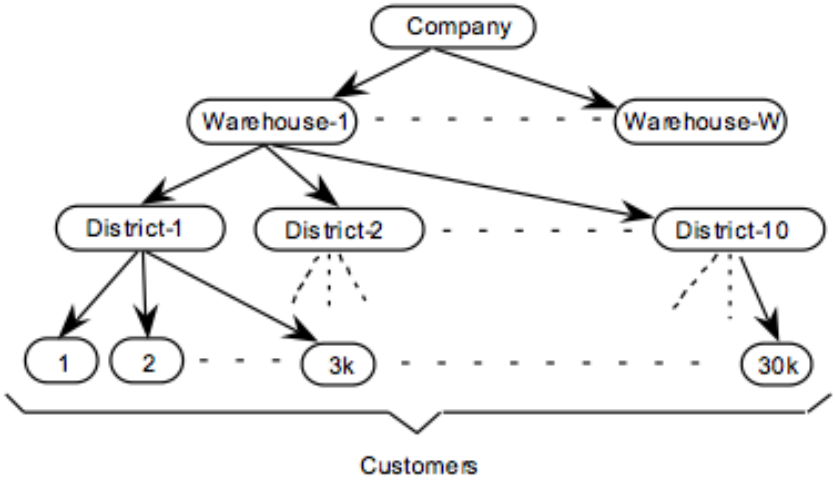
[9]L'attuale versione del benchmark TPC-C è la versione 5.11. Il TPC-C ha continuato ad evolvere il suo benchmark per rimanere il simbolo della pratica per le transazioni. Il benchmark TPC-C continua ad essere un parametro importante per confrontare le prestazioni OLTP (Online Transaction Processing) su diverse configurazioni hardware e software.

TPC-C è un estensione del benchmark TPC-A, versione standard del TPC che simula la gestione di DebitCredit. Pertanto, TPC-C include tutti i componenti di base del OLTP. Per poter essere il punto di riferimento per sistemi di varie potenze di calcolo, TPC-C deve poter scalare sia il numero di terminali, che la dimensione del database, in modo proporzionale alla potenza di calcolo del sistema misurato. Vedremo in seguito che modificando il numero di aziende presenti nel database, la complessità aumenta; questo parametro verrá modificato in base al calcolatore che si vuole valutare.

6.2.1 Definizione e costruzione del database

[9]Come punto di riferimento del sistema OLTP, TPC-C simula un ambiente completo in cui una serie di operatori esegue transazioni su un database. Il benchmark è incentrato sulle principali attività (transazioni) di ordine e di inserimento sul databa-

Figura 6.1: Schema database TPC-C.



se. Queste transazioni sono l'inserimento e la consegna degli ordini, la registrazione dei pagamenti, il controllo dello stato degli ordini, e controllo del livello delle scorte presso i magazzini. Tuttavia, va sottolineato che non è nelle intenzioni di TPC-C studiare com'é meglio implementare un sistema di inserimento ordini. Infatti, mentre l'obiettivo di un normale benchmark é di analizzare le prestazioni di fornitore all'ingrosso di un genere specifico, TPC-C può analizzare qualsiasi industria che deve gestire, vendere o distribuire un prodotto o servizio.

In TPC-C il modello di business, un fornitore all'ingrosso (chiamato anche azienda) opera in un certo numero di magazzini e loro punti vendita. Il benchmark TPC è progettato per scalare proprio come l'azienda: si può espandere e creare nuovi magazzini e nuovi punti vendita. Ogni magazzino nel benchmark TPC-C deve fornire dieci punti vendita, e ogni punto vendita serve 3.000 clienti. Un operatore del punto vendita può selezionare, in qualsiasi momento, una delle cinque operazioni (transazioni) offerte dal sistema di inserimento ordini dell'azienda. Queste operazioni vengono eseguite con una frequenza tale da simulare uno scenario realistico. L'operazione più frequente è costituita dall'entrata di un nuovo ordine che, in media, è composto da

dieci oggetti diversi. Ogni magazzino cerca di mantenere scorte (stock) per i 100.000 articoli a catalogo dell'azienda e riempire gli ordini da quel magazzino. Tuttavia, in realtà, un magazzino probabilmente non ha tutti i pezzi necessari a riempire ogni ordine. Pertanto, TPC-C richiede che quasi il dieci per cento di tutti gli ordini siano forniti da un altro magazzino dell'azienda. Un'altra transazione frequente consiste nella registrazione di un pagamento ricevuto da un cliente. Meno frequentemente può essere richiesto dagli operatori di ottenere lo status di un ordine precedentemente inserito, di elaborare un insieme di dieci ordini di consegna, di richiedere scorte per eventuali prodotti in carenza e di esaminare il livello delle scorte presso il magazzino locale. Un totale di cinque tipi di transazioni sono utilizzate per modellare questa attività di business. L'andamento metrico riportato da TPC-C misura il numero di ordini che possono essere completamente elaborati al minuto ed è espresso in tpm-C (unità di misura del test).

Figura 6.2: Dettagli database TPC-C.

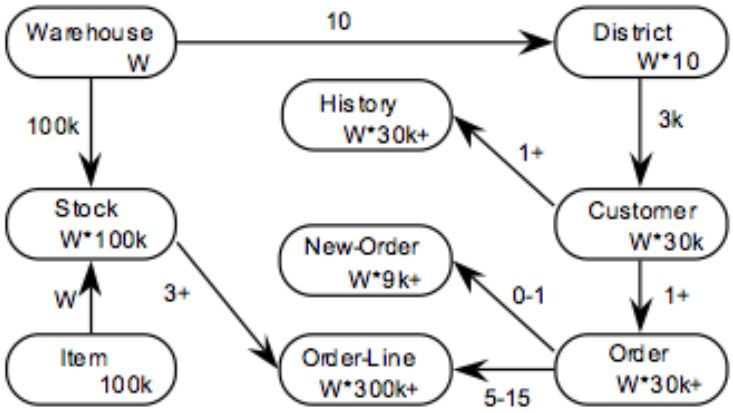
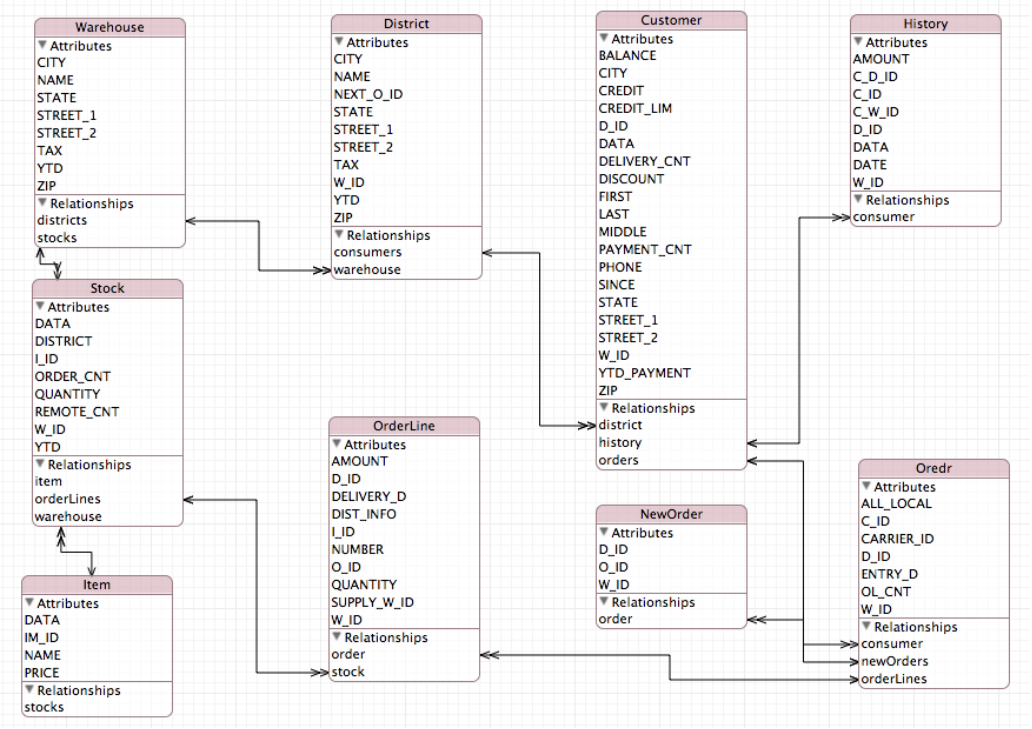


Figura 6.3: TPC-C database su xCode.

datamodel.png



6.2.2 Definizione ed esecuzione delle transazioni

Le transazioni presenti nel benchmark sono cinque:

1. Inserimento di un nuovo ordine:
2. Pagamento dell'ordine inserito
3. Presa in carico dell'ordine ed esecuzione
4. Trasferimento del materiale richiesto
5. Controllo e rifornimento delle scorte nel magazzino

Transazione 1 - inserimento di un nuovo ordine.

Vengono scelti un'azienda e un cliente Random. Il cliente ha la possibilità di effettuare un ordine composto da dieci prodotti. Si effettua prima una ricerca sul database di tutte le aziende salvate prendendone una Random, poi si esegue lo stesso processo attraverso la ricerca Random di un cliente dell'azienda selezionata.

Transazione 2 - pagamento dell'ordine inserito.

Viene creato un oggetto chiamato OrderLine che rappresenta una tabella di relazioni che collegano il cliente con il magazzino e gli oggetti richiesti. Una volta creato l'oggetto, si controlla se l'ammontare richiesto per l'ordine (denaro) può venire pagato dal cliente. In caso positivo, si detrae il denaro dal cliente e si bloccano gli oggetti richiesti dal cliente; in caso negativo si cancella l'oggetto OrderLine e si lasciano gli oggetti nel magazzino sbloccati. La quantità di denaro del cliente é decisa tramite un numero Random dove peró la possibilità di non avere denaro sufficiente é del 20%.

Transazione 3 - presa in carico dell'ordine ed esecuzione.

Per poter soddisfare le richieste dell'ordine preso in carico, si controllano gli elementi richiesti in qualsiasi magazzino: se ad esempio in un magazzino non c'é la quantità

necessaria per soddisfare la richiesta, si controlla nel magazzino successivo. Una volta recuperati dal magazzino gli elementi richiesti, questi si aggiungono alla relazione con l'OrderLine per poterne tener traccia.

Transazione 4 - trasferimento del materiale richiesto.

A questo punto l'oggetto OrderLine é completo: contiene il cliente che ha richiesto l'ordine e il pagamento per gli elementi richiesti, e a questo punto può quindi avvenire la consegna dell'ordine. Si tiene salvato l'oggetto OrderLine creando una relazione con l'oggetto History, dopo di che si eliminano gli elementi dal magazzino, l'ordine e il cliente.

Transazione 5 - controllo e rifornimento delle scorte nel magazzino.

Prima di poter eseguire una nuova transazione, si effettua un controllo sulle scorte dei prodotti presenti nei magazzini. Si controlla in ogni magazzino gli elementi mancanti e, se presenti in altri, si riequilibrano le scorte. Se un prodotto non é presente in nessun magazzino si effettua un rifornimento generale in ogni magazzino di quel prodotto. Al termine si rilancia una nuova richiesta.

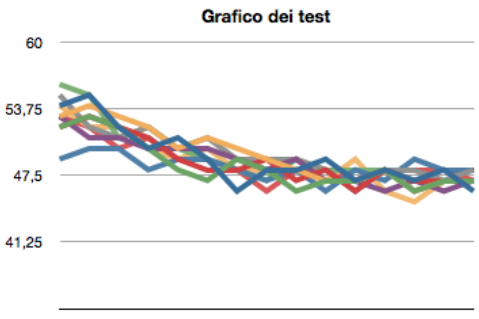
6.2.3 Analisi dei risultati

Al termine della costruzione del database, ho eseguito il benchmark su un iPad 2. Per la costruzione del database, che occupava 1,5GB di memoria, l'iPad 2 ha impiegato circa 45min. Ho eseguito una decina di test della durata di 15 min ciascuno. É molto importante eseguire più prove per avere una valutazione precisa, molti dati sono generati Random per cui, per avere la visione di una situazione realistica, ho dovuto eseguire più di un test e calcolare una media tra i risultati.

Il grafico mostra una curva che con i minuti tende a stabilizzarsi, quasi tutti i test si fermano a 46/48 transazioni al minuto. Il calo di prestazioni é dovuto all'ultima transazione: il controllo delle scorte nei magazzini. Le prime transazioni sono abba-

Figura 6.4: Grafico dei risultati.

Transazioni	1m	2m	3m	4m	5m	6m	7m	8m	9m	10m	11m	12m	13m	14m	15m
1 test	54	55	52	50	51	49	46	48	48	49	47	48	47	48	46
2 test	52	53	52	50	48	47	49	48	46	47	47	48	46	47	47
3 test	53	54	53	52	50	51	50	49	48	47	47	48	46	47	47
4 test	52	53	52	51	49	48	48	49	47	48	46	48	47	47	47
5 test	53	51	51	50	50	50	49	48	49	47	47	46	47	46	47
6 test	55	52	51	52	50	51	49	49	49	48	46	48	48	47	48
7 test	49	50	50	48	49	49	48	47	48	46	48	47	49	48	48
8 test	56	55	51	52	50	49	48	47	49	48	48	48	46	47	47
9 test	54	52	52	50	49	50	48	48	49	47	49	46	45	47	47
10 test	53	52	50	51	49	49	48	46	48	48	47	48	48	48	47



stanza veloci poiché le scorte nei magazzini sono presenti, dopo che alcuni ordini sono stati eseguiti, il controllo e il rifornimento dei magazzini richiede molto tempo.

L'organizzazione TPC effettua e pubblica numerosi test per mostrare le prestazioni di diversi computer. Sul sito ufficiale nel quale sono pubblicati i diversi risultati, ci sono macchine che riescono a compiere oltre 30 milioni di transazioni al minuto (SPARC SuperCluster with T3-4 Servers). Si tratta di computer costruiti appositamente per compiere questo tipo di operazioni, che costano oltre 30 mila dollari dotati di ben 162 processori e che occupano lo spazio di un'unità server. Ci sono poi altri generi di computer un po più gestibili e con il rapporto spesa/tpm-C migliore. L'HP ProLiant DL580 G7 (con un processore Xeon a quatto core), per esempio, riesce ad effettuare 290,040 transazioni al minuto ma, costando 113 dollari, il costo per transazione é di 0.39 dollari per ciascuna di esse.

Seguendo quest'analisi, l'iPad 2 costa attualmente 499 euro ed effettuando 47 transazioni al minuto, il costo a transazione sarebbe di 10,61 euro. Il confronto con i risultati ufficiali della TPC non ha molto senso, poiché si sta confrontando un tablet con un computer fisso. L'analisi risulta curiosa perché si potrebbe ipotizzare che, se Amazon dovesse affidare ad un iPad 2 la gestione delle sue vendite, potrebbe gestire 47 vendite al minuto, risultato che di questi tempi, risulterebbe assurdo.

	SPARC SuperCluster	HP ProLiant DL580 G7	iPad2
CPU	Intel Xeon X5670 2.93GHz	Intel Xeon 2.13GHz	Apple A5 1 GHz
Database	Oracle Database 11g Release 2 Ent.	Oracle Database 11g Release 2 Standard	Core Data SQLite
Prestazioni (tpmC)	30,249,688	290,040	47
Prezzo (USD)	30,528,863	113,012	499
Prezzo/Prestazioni	1.01	0.39	10.61

Purtroppo non vi sono ulteriori test effettuati su altri tablet, come ad esempio Android; quando saranno creati, il confronto potrebbe avere molto più senso. Al momento posso dire che le prestazioni basse dell'iPad 2 sono dovute sicuramente ad una potenza di processore bassa rispetto ad un computer fisso.

Capitolo 7

Conclusione

Dopo aver studiato le modalità di salvataggio possibili su iOS (NSUserDefaults e Core Data), si può affermare che esse siano state create per esigenze completamente diverse. Le NSUserDefaults sono state create per poter contenere piccole quantità di dati e, infatti, normalmente vengono utilizzate per salvare le impostazioni che l'utente desidera mantenere in un'applicazione. Questa classe é semplice da utilizzare poiché limitata nelle funzioni che può offrire ed é per questo motivo che consiglio il suo utilizzo nel momento in cui occorre salvare pochi dati: non perché non potrebbe contenerne grandi quantità ma perché, per poter gestire una grande collezione di essi, il suo utilizzo sarebbe troppo complesso . Per una grande quantità di dati, come ad esempio un database, risulta più semplice l'utilizzo di Core Data. Questo framework é più complesso da inserire in un'applicazione e richiede più tempo d'implementazione, ma una volta creata la base dati, é molto più gestibile rispetto a NSUserDefaults.

Per lo sviluppo di ProClara, ho voluto studiare la struttura di Core Data che risulta caratterizzata da un'implementazione più articolata ma anche più sicura dei dati. Infatti, ogni volta che si lavora su un dato presente nel database, si effettua una copia di esso per poi poterlo elaborare. Compiuto il processo di elaborazione (creazione,

modifica o cancellazione), si va a reinserire l'oggetto sostituendolo al precedente. Questo ulteriore passaggio aggiunge un livello tra l'utente e il database concreto. La gestione del database in questo modo, risulta più lenta ma molto più sicura: non si interagisce mai con il database senza passare da un livello di sicurezza chiamato NSManagedObjectContext. Attraverso questo procedimento, si lavora sempre in un contesto creato per l'occasione ma mai con il dato vero. Tale tipo di gestione comporta una notevole perdita di tempo nel momento in cui si vuole salvare il contesto modificato, questo perché si effettua un controllo su tutto il database e, in seguito, si registrano solo i cambiamenti effettuati mantenendo inalterato il resto.

Le prestazioni di Core Data sono state anche valutate grazie al benchmark TPC e il risultato ha rilevato un'effettiva lentezza nel processo di salvataggio. Nonostante il benchmark non sia stato creato per piattaforma tablet, si può supporre che la gestione di dati tramite contesti porti ad un leggero peggioramento nella velocità di salvataggio delle informazioni. La motivazione di questo tipo di gestione é data sicuramente da un diverso utilizzo della base dati: mentre nel caso di computer fissi si ha meno preoccupazione relativamente all'integrità fisica della macchina, su un tablet, bisogna prendere in considerazione che potrebbe essere soggetto a cadute o esaurimento di batteria. Inoltre, le applicazioni presenti su tablet, non sono predisposte per un utilizzo sempre costante nel tempo (come invece sono i server) bensì per un utilizzo frenetico (apertura e chiusura dell'applicazione molto frequente).

Implementare Core Data si é rivelata la giusta scelta per creare ProClara poiché garantisce un'ottima protezione dei dati, la possibilità di evolvere in nuove versioni nonché una buona velocità nella ricerca dei dati e nel loro salvataggio. Essendo infine la soluzione consigliata da Apple, si ha la ulteriore certezza di poter continuare ad usufruire del supporto delle API per anni garantendo così la funzionalità dell'applicativo a lungo termine.

Bibliografia

[1] Dina Bass. Microsoft net falls below apple's as ipad eats into sales. http://www.businessweek.com/news/2011-04-29/microsoft-net-falls-below-apple-s-as-ipad-eats-into-sales.html, 2011.

[2] iFixit.com. Teardown ipad. http://www.ifixit.com/Teardown/iPad-Wi-Fi-Teardown/2183/2, 2010.

[3] Wikipedia. Package on package. http://en.wikipedia.org/wiki/PackageOnPackage.

[4] Stephen G. Kochan. *Programming in Objective-C 2.0*. Addison-Wesley, 2009.

[5] Apple. Core data. http://developer.apple.com/technologies/ios/data-management.html, 2010.

[6] Sadun Erica. *The iPhone developer's cookbook*. Addison-Wesley, 2010.

[7] Marcus S. Zarra. *Core Data Apple's API for Persisting Data on Mac OS X*. The Pragmatic Programmers, 2009.

[8] Transaction Processing Performance Council organization. Transaction processing performance council organization. http://www.tpc.org/, 2011.

[9] TPC-C Standard Benchmark. Tpc-c standard benchmark. http://www.tpc.org/tpcc/detail.asp, 2011.

Ringraziamenti

Vorrei cogliere l'occasione per ringraziare alcune persone che, oltre ad aver contribuito a portare a termine questo lavoro, mi hanno aiutato durante il mio percorso universitario.

Ringrazio il professor Danilo Montesi per la sua disponibilità e per essere stato mio relatore e guida nella creazione di questo progetto.

Ringrazio Carlotta, per avermi seguito durante tutti i cinque anni, per avermi aiutato in ogni occasione e per esser riuscita a sopportarmi con amore. Spero di riuscire ad aiutarti come sei riuscita a farlo tu con me. Ti amo tanto.

Ringrazio i miei genitori, che ritengo i migliori genitori al mondo; nonostante le mie continue difficoltà nello studio hanno sempre creduto in me e sopportato sacrifici enormi. Questo lavoro é dedicato interamente a loro come segno di riconoscimento per il loro impegno. Oltre a loro ringrazio tutta la mia famiglia e i miei fratelli Davide, Paolo e Clelia.

Ringrazio tutti gli amici conosciuti in università, soprattutto Francesca, Michele e Paolo; per aver affrontato sempre con me questo percorso e, in particolare gli ultimi progetti universitari, e con loro Giacomo, Lorenzo, Nikolas, Gianluca, Tommaso, Marco V., Marco M., Marco C., Davide, Simone, Iolao. Con tutti loro ho superato esami che sembravano impossibili e, insieme abbiamo condiviso diverse passioni in comune.

Ringrazio Andrea Gelati, per essere stato il mio maestro, collega e amico; senza di lui ProClara non sarebbe potuta diventare com'é attualmente. Spero di continuare ad apprendere da lui il più possibile e proseguire nel mio percorso lavorativo.

Ringrazio Command Guru, per la grande opportunità e la fiducia che mi ha dato e un grazie anche alle singole persone che, ogni giorno, lavorano accanto a me e hanno la pazienza di insegnarmi e di sopportarmi, in particolare Alessio, Mary e Ilaria.