ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

DISI

TESI DI LAUREA

in

Tecnologie Web T

Piattaforme di supporto per applicazioni mobili iOS in servizi e-Commerce basati su PrestaShop

CANDIDATO:

Ilario Giuseppe Genuardi

RELATORE:

Prof. Ing. Paolo Bellavista

CORRELATORI:

Dott. Ing. Luca Foschini

Dott. Ing. Andrea Cirri

Anno Accademico 2012/2013

Sessione II

SOMMARIO

INTRODUZIONE

In un periodo dove l'innovazione e le nuove tecnologie influenzano e semplificano la vita di tutti i giorni, l'avvento di Internet è senza ombra di dubbio quella che più ha stravolto le abitudini di ogni persona. Grazie ad Internet il mondo è più a portata di mano e con il suo crescente sviluppo sono nati nuovi servizi, o meglio nuovi modi di fare le cose, uno di questi è il commercio elettronico.

Il commercio elettronico, o e-Commerce, è quella forma di commercio che si avvale delle reti telematiche, soprattutto Internet, con il quale un soggetto può vendere o acquistare beni e servizi, utilizzando qualunque dispositivo in grado di connettersi a queste reti. Ciò ha comportato la nascita di veri e propri negozi virtuali, nei quali gli acquirenti possono acquistare qualunque genere di cose senza recarsi fisicamente in un negozio. Nel commercio elettronico il software sostituisce il commesso. I venditori diventano più competitivi ed allargano i propri orizzonti, basti pensare che ognuno ha la possibilità di effettuare transazioni, anche con persone dislocate in varie parti del mondo, il tutto semplicemente e comodamente da casa.

In particolare, la crescita dei servizi offerti dall'e-Commerce ha permesso a molti sviluppatori di software di creare delle piattaforme che permettano a chiunque (anche chi non conosce un minimo di programmazione) di "aprire" un negozio virtuale nel giro di pochi minuti ed in maniera semplice ed efficace. Le piattaforme più importanti sono: Magento, VirtueMart (add-on di Joomla!), WP-eCommerce e Prestashop. Ognuna di esse si differenzia dalle altre per i servizi offerti. Prestashop, quella utilizzata per il progetto associato a questa tesi è, ad esempio, una piattaforma gratuita ed open source realizzata completamente in PHP e funzionante su database MySQL, molto apprezzata dagli sviluppatori per la sua semplicità d'uso e l'estrema intuitività. Inoltre, nasce con l'intendo di essere modulare, cioè costituita da moduli di terze parti e per questo motivo estremamente personalizzabile.

L'avvento degli smartphone e dei tablet, sui quali girano Android, iOS e Windows Phone, ha ulteriormente agevolato la diffusione dell'eCommerce e tale pratica ormai è divenuta molto frequente, grazie anche all'affidabilità delle reti, in termini di sicurezza. Le ridotte dimensioni degli schermi dei dispositivi, però, rendono in un certo senso più "scomoda" l'esplorazione dei cataloghi dei prodotti, per questa ragione i proprietari dei negozi online spesso si avvalgono di un'applicazione correlata ed installabile fisicamente sul dispositivo che permette di adattare i contenuti e semplificare l'intero processo di vendita.

Nonostante molti siti modifichino la loro interfaccia per le differenti risoluzioni dei display, le app, a differenza dei comuni browser, sono ottimizzate, quindi più efficienti in termini di prestazioni, e pertanto preferite. Uno dei motivi è anche la migliore integrazione con i sistemi operativi che permette di sfruttare le caratteristiche di ogni dispositivo, basti pensare alla possibilità di aggiungere funzioni di geolocalizzazione. In iOS, il sistema operativo dei dispositivi mobili di casa Apple (iPhone, iPad, iPod Touch), ad esempio, esistono due tipi di applicativi: le app native e le app web-based. Le prime si installano sui dispositivi, le seconde sfruttano il browser Safari.

L’obiettivo di questa tesi è lo sviluppo di un framework che permetta l’interazione tra il sistema operativo iOS, in particolare un’app, e la piattaforma di commercio elettronico Prestashop. Questo framework racchiuderà tutte le tecnologie utili alla comunicazione con un server remoto e può essere utilizzato come base per coloro che vogliano sviluppare un’app che comunichi proprio con questa piattaforma.

Un framework è un insieme di librerie di codice riusabili che svolgono operazioni spesso utilizzate in più progetti, agevolando il lavoro degli sviluppatori che non dovranno preoccuparsi di scrivere ulteriori linee di codice per le loro applicazioni, dato che questo codice è già stato scritto precedentemente da altri.

La creazione di un framework dedicato, per iOS, permette di estendere i servizi offerti dalla piattaforma Prestashop, aggiungere nuove funzionalità, come ad esempio il caching e la gestione dello stato, e nel contempo agevolare lo sviluppo di applicazioni native per il sistema operativo dei dispositivi di casa Cupertino.

La parte iniziale del progetto verrà dedicata all’analisi dei servizi offerti da Prestashop ed allo sviluppo in Objective-C, il linguaggio di programmazione con cui vengono sviluppate le app ed i framework di iOS, di una libreria di metodi che permettono la comunicazione con le API della piattaforma. Tali metodi, denominati REpresentational State Transfer (REST), sfruttano le specifiche del protocollo HTTP, e serviranno per creare, modificare, recuperare ed eliminare i dati presenti sul server su cui risiede il software della piattaforma. A tal fine verrà effettuato il porting di una libreria di metodi per Android già esistente, denominata PrestashopRESTLib, il cui scopo è proprio quello di gestire la connessione tra framework e server remoto.

Nella seconda parte del progetto, dopo opportuni test della libreria precedente, si svilupperanno: una classe che implementerà le funzionalità di alto livello offerte dal framework, in particolare le estensioni proposte; degli oggetti che conterranno i dati richiesti al web server e quindi la logica di conversione di quest’ultimi, utili per permettere al framework di utilizzare le risorse remote; varie classi atte ad aggiungere ulteriori funzioni, come quella per gestire la cache immagini.

La terza ed ultima parte del progetto, invece, sarà dedicata allo sviluppo di un’app nativa che servirà per testare il framework nel suo insieme. Verranno create delle viste utente gestite da classi indipendenti al framework che permetteranno di visualizzare i risultati delle operazioni svolte dai metodi del framework. Lo studio a runtime verrà effettuato con l’ausilio di Instruments, applicativo della suite XCode (l’ambiente di sviluppo di Apple), che consentirà di vedere il comportamento del framework, inteso come ritardi, consumi di memoria e uso del processore.

Questo testo verrà strutturato in due parti per un totale di quattro capitoli. La prima parte, formata dai primi due, analizzerà il sistema operativo iOS e il linguaggio di programmazione Objective-C. La seconda parte, invece, formata dai restanti, sarà incentrata sul framework, ed in particolare sull’architettura e sugli elementi che lo compongono. Infine verranno analizzate le sue performance a tempo di esecuzione.

Nel primo capitolo, quindi, verrà data una panoramica di iOS nel suo insieme attraverso la descrizione generale del sistema, l’analisi dei suoi livelli di astrazione, quattro in totale e disposti uno sull’altro (Cocoa Touch, Media, Core Services e Core OS), e le caratteristiche

e le funzionalità che lo differenziano dagli altri sistemi mobili (threading, multitasking, etc.), nonché ciò che lo rende uno dei sistemi più performanti anche a livello di gestione delle risorse e della reattività. Infine, si introdurrà l'ambiente di sviluppo XCode, utilizzato per lo sviluppo del codice.

Il secondo capitolo, utile per comprendere meglio l'analisi del codice delle componenti del framework dei capitoli successivi, descriverà la sintassi e le caratteristiche del linguaggio di programmazione Objective-C, derivato dal C, confrontandolo anche con altri linguaggi come Java, C# e C++.

Nel terzo capitolo, dopo aver introdotto il significato di eCommerce, aver descritto la piattaforma Prestashop e ciò che mette a disposizione, si inizierà a parlare del framework. Inizialmente ne verrà mostrata l'architettura e si darà una descrizione generale degli elementi che la compongono. Nel proseguo si parlerà delle funzionalità di basso livello e della libreria di metodi REST utilizzati per lo scambio di messaggi tra il framework e la piattaforma. Infine, verranno presi in esame i metodi della classe che gestisce tali funzionalità, utili per comprendere meglio come viene effettuata la comunicazione.

L'ultimo capitolo riprenderà la descrizione dell'architettura con risalto alle funzionalità di alto livello del framework ed alle sue estensioni rispetto alla piattaforma Prestashop. Nel dettaglio si parlerà della cooperazione tra livelli e della classe che implementerà i metodi necessari per interagire con le API di un negozio virtuale e tutte le tecnologie offerte dal framework, come ad esempio l'aggiunta dei prodotti alle viste, la gestione dello stato, il caching, etc. Di questa classe verrà descritto anche il funzionamento dei vari metodi. Nell'ultimo paragrafo, infine, verrà fatta l'analisi a runtime sul comportamento del framework utilizzando un'app di test appositamente creata; verranno quindi proposti degli scenari d'uso e si raccoglieranno i dati sperimentali.

RINGRAZIAMENTI

Desidero innanzitutto ringraziare il Signore per aver vegliato su di me e per non avermi mai fatto sentire la Sua mancanza nei momenti più bui.

Un grazie di vero cuore al Professor Paolo Bellavista per i preziosi insegnamenti durante questi anni di laurea e per le numerose ore dedicate alla mia tesi.

Inoltre, ringrazio sentitamente il Dott.Ing. Luca Foschini ed il Dott.Ing. Andrea Cirri che sono stati sempre disponibili a chiarire i miei dubbi durante la stesura di questo elaborato e lo sviluppo del progetto correlato.

Intendo poi ringraziare il corpo docenti di questa Università per tutti gli insegnamenti ricevuti, con particolare menzione al Prof. Carlo Ravaglia per tutte le ore di ricevimento dedicatemi.

Inoltre, vorrei esprimere la mia sincera gratitudine ai miei compagni di corso, in particolare Simone Blandini, per la disponibilità e i numerosi consigli durante la stesura di questo testo.

Desidererei ringraziare con affetto i miei genitori per il sostegno, per il grande aiuto che mi hanno dato e per aver vissuto in prima persona questo mio percorso di studi.

Ringrazio mia nonna Mela, le mie zie Lia e Mariella, Graziella e mio padrino don Giuseppe per essermi stati vicini in ogni momento con incoraggiamenti e preghiere.

Ringrazio mia sorella e i miei nipoti per avermi "sopportato" e ospitato qui a Bologna.

Un ringraziamento speciale anche alla mia ragazza, ai suoi genitori, a Rita e Davide per l'appoggio morale e la pazienza avuta nei miei confronti.

Infine per ultimi, ma non meno importanti, ringrazio tutti i miei parenti e gli amici più cari, in particolare Salvatore (mio "figlioccio"), Giuseppe, Simone e Massimo.

CAPITOLO 1
IL SISTEMA OPERATIVO iOS

1.1 PANORAMICA SU iOS

iOS è il sistema operativo, sviluppato da Apple che gira su iPhone, iPod touch, ed iPad. È derivato da UNIX (famiglia BSD) ed usa un microkernel XNU Mach basato su Darwin OS. Come tutti i sistemi operativi gestisce l'hardware e fornisce i servizi e le tecnologie che servono all'implementazione delle applicazioni native degli iDevice (nome generico dei dispositivi sui quali è installato iOS).

Esistono due tipologie di applicazioni che possono essere avviate su iOS, le applicazioni native e quelle web-based. Le applicazioni native, sviluppate utilizzando i framework e il linguaggio Objective-C, vengono eseguite direttamente nel sistema e sono installate fisicamente sul dispositivo, quindi sempre disponibili agli utenti anche in assenza di rete. Le applicazioni web, create dalla combinazione di HTML, fogli di stile (CSS) e codice JavaScript, invece, vengono eseguite nel browser Safari, ma a differenza delle applicazioni native richiedono una connessione di rete per accedere al web server.

iOS agisce da intermediario tra le applicazioni e l'hardware sottostante. Le applicazioni, salvo in alcuni casi, non comunicano direttamente con l'hardware, ma comunicano attraverso un insieme di interfacce ben definite che premettono di farle funzionare senza problemi su dispositivi con differenti specifiche hardware.

L'architettura di iOS è distribuita su quattro livelli di astrazione: Cocoa Touch, Media, Core Services, Core OS.

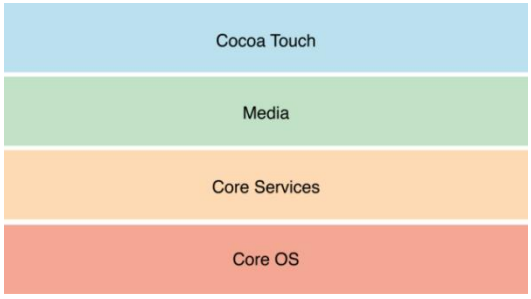

Figura 1 – Livelli di iOS

Nei livelli più bassi sono presenti i servizi fondamentali e le tecnologie su cui si basano tutte le applicazioni, mentre in quelli superiori i servizi e le tecnologie più sofisticate. I quattro livelli verranno approfonditi nel paragrafo successivo.

Le tecnologie di iOS sono raccolte in framework che sono directory contenenti librerie dinamiche condivise (simili ai package Java) e risorse (file header, immagini, etc). La maggior parte delle interfacce fornite da Apple sono contenute in alcuni di essi.

I framework di livello superiore forniscono le astrazioni object-oriented per i costrutti dei livelli inferiori. Queste astrazioni racchiudono caratteristiche complesse e semplificano il processo di sviluppo delle applicazioni perché rendono più semplice la scrittura del codice, in quanto ne riducono la quantità.

Per lo sviluppo di applicazioni si utilizza XCode e l'iOS Software Development Kit (SDK) che contiene tutti gli strumenti e le interfacce necessarie per l'implementazione, l'installazione, l'esecuzione ed il testing delle applicazioni native che verranno aggiunte alla schermata iniziale (springboard) di un dispositivo iOS.

Una delle particolarità di iOS è data dalle limitazioni, complete o parziali, di alcune funzionalità, come l'impossibilità di installare applicazioni non certificate da Apple o il blocco del bluetooth. Quest'ultimo, infatti, può essere utilizzato solo per connettere dispositivi ausiliari (auricolari, tastiere, etc), ma non per scambiare dati con computer o smartphone.

Prima di essere approvata e resa disponibile nell'App Store, un'applicazione viene controllata e testata al fine di controllarne: la sicurezza, eventuali problematiche legate al suo utilizzo e la qualità. Questo rende più difficile l'installazione di applicazioni malevoli ma nel contempo limita la libertà degli utenti. Tuttavia, è possibile aggirare questa limitazione attraverso una procedura ormai frequente e dichiarata legale dal Tribunale Federale USA, chiamata "jailbreak" (in italiano "sblocco"), che permette l'uso anche di applicazioni non approvate, presenti non sullo Store di Apple, ma su un market alternativo che prende il nome di Cydia.

1.2 ARCHITETTURA A LIVELLI

La struttura del sistema è di tipo stratificato e può essere immaginata, come accennato precedentemente, in livelli disposti uno sull'altro, delegati a gestire ed implementare determinate funzionalità. I livelli superiori beneficeranno dei servizi offerti da quelli inferiori senza conoscerne la logica implementativa.

Quelli che compongono l'architettura di iOS sono quattro e dal più alto al più basso troviamo: il **Cocoa Touch** che gestisce le "feature" più ad alto livello, come il riconoscimento del touch, e che contiene i framework su cui si basano direttamente le applicazioni di iOS (i più importanti UIKit e Foundation), il livello **Media** che fornisce il supporto a tutti i dati multimediali, il **Core Services** che si occupa dei servizi di sistema e per finire il livello **Core OS** (Darwin) che contiene il kernel, il file system, etc., e che gestisce le funzionalità a livello di sistema.

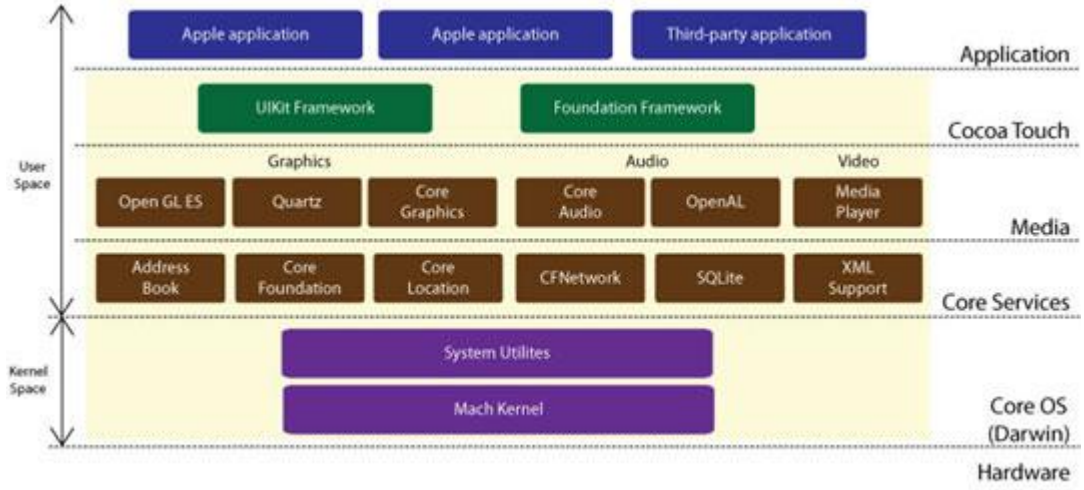

Figura 2 – Architettura livelli di iOS

1.2.1 Il livello Cocoa Touch

È il livello che sta più in alto, contiene le API che permettono la creazione di applicazioni ed il supporto al multitasking, alle notifiche push, all'input basato sul touch ed a parecchi servizi di sistema di alto livello. Gestisce funzionalità come l'accelerometro ed il giroscopio riuscendo dunque a capire in che modo è orientato il dispositivo rispetto ad un asse orizzontale perpendicolare all'asse gravitazionale.

Nello sviluppo delle applicazioni ricopre un ruolo importante perché su di esso si basano le tecnologie implementative da seguire. Tra le più importanti caratteristiche del livello abbiamo:

Auto Layout (introdotta con iOS 6): definisce una serie di regole che permettono di disporre automaticamente gli oggetti presenti nell'interfaccia utente, come ad esempio la posizione di un pulsante. Contrariamente al modello precedente "springs and struts", questa caratteristica offre un approccio più intuitivo e permette di avere una serie di vantaggi quali ad esempio il mirroring per la scrittura da destra verso sinistra (per le lingue come l'arabo o l'ebraico) o un migliore livello di responsabilità tra gli oggetti delle view-controller.

Mantenimento dello stato (introdotta in iOS 6): permette alle applicazioni di salvare e ripristinare lo stato della loro interfaccia utente. Alla riapertura dell'applicazione, grazie a questa caratteristica, l'utente avrà l'impressione che l'applicazione non fosse mai stata chiusa. Il supporto per il mantenimento dello stato è integrato nel framework *UIKit*.

Multitasking: questa caratteristica, introdotta a partire da iOS SDK 4.0, permette ad un utente di poter tenere aperte più applicazioni contemporaneamente. Rispetto alle precedenti versioni di iOS, le applicazioni alla pressione del tasto home non vengono più terminate ma vengono eseguite in background. Essendo una caratteristica che incide parecchio sulla durata della batteria, la maggior parte delle applicazioni viene sospesa dal sistema poco dopo l'ingresso in background. Un'applicazione sospesa rimane in memoria, ma non esegue alcun codice. Questo comportamento consente di riprenderne velocemente l'esecuzione al momento della riapertura, senza incidere sulla carica della batteria. Tuttavia, le applicazioni possono essere autorizzate a continuare l'esecuzione in background per i seguenti motivi: per completare un compito importante, per sostenere servizi specifici o per generare avvisi per gli utenti. Il supporto al multitasking, indipendentemente se l'applicazione verrà sospesa o continuerà a funzionare in background, non richiede lavoro aggiuntivo da parte degli sviluppatori. Il supporto al multitasking è definito nel framework *UIKit*.

Riconoscimento di movimento (introdotta con iOS 3.2): permette di definire le azioni che il sistema operativo dovrà svolgere al verificarsi di un gesto compiuto dall'utente sul display del dispositivo. Per gesto s'intende, ad esempio, il tocco con la punta del dito su un bottone (*tapping*), il trascinamento di un oggetto (*panning e dragging)*, il "pizzicare" con le dita per lo zoom in/out delle foto (*pinching in e out*). Grazie proprio all'introduzione di questa caratteristica lo sviluppatore non dovrà più preoccuparsi della gestione dell'input dell'utente, in quanto già definita nel framework *UIKit*. Questo framework include una classe *UIGestureRecognizer* che definisce il comportamento di base per tutti i riconoscitori di movimento. Tuttavia è possibile definire anche riconoscitori personalizzati ma l'operazione è alquanto complicata.

Servizio di notifica push di Apple (introdotta in iOS 3.0): permette di avvisare l'utente quando nuove informazioni (notifiche) sono disponibili. È un servizio che funziona anche quando l'applicazione in esecuzione non è attiva. Utilizzando questo servizio, è possibile ricevere sui propri dispositivi notifiche di testo, aggiungere un badge all'icona dell'applicazione (un pallino rosso con un numero che indica il numero di notifiche), o attivare gli avvisi acustici alla ricezione di una notifica. Il funzionamento delle notifiche push è caratterizzato da due parti:

1) L'applicazione richiede il servizio ed elabora i dati di notifica una volta ricevuti;

2) Un processo lato server genererà le notifiche che dovranno essere elaborate al punto 1.

Notifiche locali (introdotta con iOS 4.0): estende l'attuale meccanismo di notifica push, dando alle applicazioni la possibilità di generare delle notifiche in locale, invece che basarsi su un server remoto. Le applicazioni in esecuzione in background possono utilizzare le notifiche locali per attirare l'attenzione di un utente quando si verificano degli eventi importanti. Ad esempio, un'applicazione di navigazione stradale può avvisare l'utente quando è nelle vicinanze di un luogo d'interesse. Le applicazioni possono consegnare le notifiche locali ad una certa data e ora, anche se non sono in esecuzione. Il vantaggio, quindi, è che le notifiche locali sono indipendenti dalle applicazioni stesse in quanto è Il sistema operativo che si occupa della loro gestione.

Servizio per il P2P (introdotto con iOS 3.0): è una caratteristica presente nel framework Game Kit (*GameKit.framework*). Offre il supporto alla comunicazione peer-to-peer tra due dispositivi mediante l'ausilio della tecnologia Bluetooth. Sebbene sia usata principalmente nei giochi, soprattutto quelli multiplayer, è anche possibile utilizzare questa funzione per altri tipi di applicazioni.

Storyboard: (introdotta con iOS 5), consente di risparmiare molto tempo nella costruzione dell'interfaccia utente delle applicazioni. Le storyboard permettono, a differenza dei nib file, di contenere tutte le viste dell'applicazione in un unico file. Proprio quest'ultima caratteristica rende più semplice il compito dello sviluppatore che si ritroverà a gestire un unico file piuttosto che tanti separati. Dal punto di vista estetico, le storyboard introducono il concetto delle transizioni ("segue") tra le varie viste.

Supporto ai documenti: (Introdotta con iOS 5), permette di gestire i dati associati ai documenti mediante la classe *UIDocument* del framework *UIKit*. Questa classe rende semplice l'implementazione delle applicazioni document-based, specialmente quelle che scambiano dati con iCloud. Oltre a fornire un container per i dati contenuti nei documenti, la classe *UIDocument* fornisce il supporto alla lettura e scrittura asincrona, il salvataggio automatico, il salvataggio sicuro dei dati, il supporto per la rilevazione di conflitti con iCloud, etc.

Supporto per display esterni (introdotta con iOS 3.2): permette ad alcuni dispositivi di essere collegati ad un monitor esterno attraverso una serie di cavi supportati. Quando uno schermo associato è collegato, può essere utilizzato dall'applicazione per visualizzarne i contenuti. Le informazioni sullo schermo, comprese le risoluzioni supportate, sono accessibili attraverso le interfacce del framework UIKit.

Supporto di stampa (introdotta con iOS 4.2): consente alle applicazioni di inviare, a stampanti vicine, dei dati in modalità wireless. Gestisce le interfacce di stampa, esegue il rendering del contenuto stampabile e gestisce la coda e l'esecuzione dei lavori sulla stampante. Quando un'applicazione invia una richiesta, delega il sistema di stampa che si occuperà di gestire il processo vero e proprio.

Standard System View Controller**:** molti dei framework del livello Cocoa Touch contengono le view-controller delle interfacce standard di sistema. Ogni volta che si deve eseguire un'operazione, come ad esempio scattare o selezionare un'immagine dalla libreria delle foto dell'utente, è necessario utilizzare uno dei controller definiti nel framework relativo.

Alcuni esempi sono:

- Address Book UI framework (*AddressBookUI.framework*): per visualizzare o modificare le informazioni di contatto;
- Event Kit UI framework (*EventKitUI.framework*): per creare o modificare gli eventi del calendario;
- Message UI framework (*MessageUI.framework*): per comporre una e-mail o un SMS;
- La classe UIDocumentInteractionController del framework UIKit (UIKit.framework): per aprire o visualizzare in anteprima il contenuto di un file;
- La classe *UIImagePickerController* del framework *UIKit*: per scattare o selezionare una foto dalla libreria delle foto dell'utente.

1.2.2 Il livello Media

Il livello media contiene le tecnologie per la visualizzazione e la gestione degli aspetti multimediali dei dispositivi di casa Apple. Queste tecnologie sono state progettate per rendere semplice la creazione delle applicazioni mantenendo un certo grado di qualità. Esse sono raggruppate in tre famiglie distinte: tecnologie grafiche, tecnologie audio e tecnologie video.

L'aspetto grafico in iOS è una componente importante, ragion per cui tutte le applicazioni devono essere progettate tenendone conto. Il modo più semplice per creare un'applicazione di ottima qualità è quella di usare delle immagini prerenderizzate insieme alle viste standard ed ai controlli del framework *UIKit*, e lasciare che sia il sistema ad occuparsi del resto. Tuttavia, non è esclusa la possibilità in cui si possa andare al di là dei semplici elementi, per questo il livello fornisce le seguenti tecnologie per gestire i contenuti grafici:

- *Core Graphics* (noto anche come *Quartz*): gestisce in maniera nativa immagini vettoriali 2D e rendering;
- *Core Animation* (parte di *Quartz*): fornisce il supporto avanzato per l'animazione delle viste ed di altri contenuti;
- *Core Image*: offre un supporto avanzato per la manipolazione di immagini e video;
- *OpenGL ES* e *GLKit*: forniscono il supporto per il rendering 2D e 3D con accelerazione hardware;
- *Core Text*: fornisce un sofisticato layout per il testo e motore di rendering;
- *Image I/O*: fornisce le interfacce per la lettura e la scrittura della maggior parte dei formati di immagine;
- Il framework *Assets Library*: consente di accedere alle foto ed ai video della libreria fotografica dell'utente.

Le tecnologie audio disponibili in iOS sono state progettate per fornire una ricca esperienza audio agli utenti. Questa esperienza include la possibilità di riprodurre e registrare audio di alta qualità ed attivare la funzione di vibrazione su alcuni dispositivi.

Il sistema fornisce diversi modi per riprodurre e registrare contenuti audio. I framework del seguente elenco sono ordinati dal livello più alto al più basso. I livelli più in alto sono più facili da usare, e per questo preferiti, mentre i livelli sottostanti a differenza dei primi, offrono una maggiore flessibilità e controllo, ma richiedono più lavoro implementavo.

- *Media Player framework*: offre un facile accesso alla libreria iTunes dell'utente e il supporto per la riproduzione di tracce e di playlist;
- *AV Foundation framework*: fornisce un insieme di interfacce semplici, scritte in Objective-C, per la gestione della riproduzione e registrazione dell'audio;
- *OpenAL*: fornisce una serie di interfacce cross-platform per la trasmissione di audio posizionale;
- *Core Audio framework*: offrono semplici e sofisticate interfacce per la riproduzione e la registrazione di contenuti audio. È possibile utilizzare queste interfacce per riprodurre suoni di avviso di sistema, attivare la vibrazione di un dispositivo, gestire il buffer e per la riproduzione multicanale locale o di streaming audio.

I formati audio supportati in iOS sono: AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM (IMA4), Linear PCM, µ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10, AES3-2003

Per la riproduzione di filmati o lo streaming in rete, iOS fornisce diverse tecnologie video. È anche possibile utilizzare queste tecnologie per catturare video e incorporarli nelle applicazioni, ma il dispositivo deve essere dotato di hardware video appropriato, come ad esempio una fotocamera integrata.

Il sistema fornisce diversi modi per riprodurre e registrare contenuti video e possono essere scelti in base alle esigenze. Quando si sceglie una tecnologia video, bisogna ricordarsi che i framework di livello superiore semplificano il lavoro e sono pertanto consigliati. I seguenti framework sono ordinati dal livello più alto a quello più basso.

- La classe *UIImagePickerController* (framework *UIKit*): fornisce un'interfaccia standard per la registrazione video sui dispositivi con fotocamera;
- *Media Player framework:* fornisce una serie di semplici interfacce per la visualizzazione di filmati a schermo intero o parziale;
- *AV Foundation framework*: fornisce un insieme di interfacce Objective-C per la gestione della cattura e riproduzione di filmati;
- *Core Media*: è il più basso livello e descrive i tipi di dati utilizzati dalle strutture di livello superiore, fornisce interfacce di basso livello per la manipolazione dei media.

Le tecnologie video supportate in iOS per la riproduzione di file video sono: mov, mp4, m4v e 3gp.

1.2.3 Il livello Core Services

Il livello Core Services contiene i servizi fondamentali di sistema su cui si basano tutte le applicazioni. Anche se questi non vengono utilizzati direttamente, su di essi sono costruite alcune parti del sistema.

Questo strato implementa le utility di sistema come la gestione del networking, la lista dei contatti e le preferenze dell'utente. Inoltre, se nelle applicazioni viene utilizzato un database SQLite, quest'ultimo lavorerà proprio a questo livello e dunque l'accesso e le interrogazioni verranno gestite proprio dal Core Services.

Le caratteristiche del livello sono:

Automatic Reference Counting:

L'Automatic Reference Counting (ARC) è una funzionalità che semplifica il processo di gestione del tempo di vita degli oggetti Objective-C. Prima di ARC, i metodi retain, release e autorelease si dovevamo chiamare manualmente e nei posti giusti per assicurare che gli oggetti rimanessero "in vita". Il dimenticare o inserire nel posto errato anche solo una chiamata a questi metodi dava origine a sprechi di memoria, che tipicamente crescono man mano che si usa l'applicazione, o addirittura a crash, cosa inaccettabile per applicazioni professionali.

Block Objects:

Introdotti in iOS 4.0, i Block Object o semplicemente Blocchi, sono dei costrutti del linguaggio C che è possibile incorporare nel codice Objective-C. Un blocco è essenzialmente una porzione di codice racchiusa tra parentesi graffe e preceduta dal simbolo ^. In iOS, i blocchi vengono usati comunemente per sostituire i metodi delegate, per sostituire le funzioni di callback, per implementare i gestori di completamento di operazioni one-time, per semplificare l'esecuzione di un compito su tutti gli elementi di una collezione e per effettuare attività asincrone (insieme alle code di dispatch).

Data Protection:

La Data Protection (protezione dei dati) consente alle applicazioni che utilizzano dati sensibili di sfruttare la crittografia. Quando l'applicazione marca un file specifico come protetto, il sistema memorizza il file su disco in un formato codificato. Mentre un dispositivo è bloccato, il contenuto dei file non è accessibile sia dalle applicazioni che da potenziali intrusi; allo sblocco viene creata una chiave di decifratura per consentire all'applicazione di accedere al file.

File-Sharing Support:

Introdotto in iOS 3.2, il supporto ai file-sharing consente alle applicazioni di creare dei file dati da condividere con iTunes. I dati condivisi saranno presenti nella directory "/Documents", situata all'interno del percorso di installazione dell'applicazione. L'utente può spostare i file, come ad esempio un file pdf per un'applicazione reader, in maniera bidirezionale da o in questa directory. Questa funzione però non permette di condividere file con altre applicazioni.

La condivisione di file per l'applicazione si attiva aggiungendo la chiave `UIFileSharingEnabled` al file Info.plist dell'applicazione ed impostandone il valore su `YES`.

Le applicazioni che supportano la condivisione dei file dovrebbero essere in grado di riconoscere i file quando questi vengono aggiunti alla directory "/Documents", in modo da poterli gestire in modo appropriato.

Grand Central Dispatch:

Introdotto in iOS 4.0, il Grand Central Dispatch (GCD) è una tecnologia BSD-level che si utilizza per gestire l'esecuzione dei task delle applicazioni. GCD combina un modello di programmazione asincrona con un core altamente ottimizzato, al fine di fornire una comoda e più efficiente alternativa al threading. GCD fornisce anche alternative convenienti per molti tipi di compiti di basso livello, come la lettura e la scrittura di file descriptor, implementazione di timer e monitoraggio di segnali ed eventi di processo.

In-App Purchase:

Introdotto in iOS 3.0, In-App Purchase dà la possibilità di vendere contenuti e servizi all'interno delle applicazioni. Questa funzionalità è implementata nel framework *Store Kit* e fornisce l'infrastruttura necessaria per le transazioni finanziarie tramite gli account iTunes degli utenti. In iOS 6, è stato aggiunto anche il supporto per l'hosting e l'acquisto di contenuti da iTunes.

SQLite:

La libreria SQLite consente di incorporare un database leggero SQL alle applicazioni senza eseguire un separato processo lato server remoto. Dalle applicazioni è possibile creare database locali e gestirne le tabelle e i record. La suddetta libreria è stata progettata per usi general purpose, ma è anche ottimizzata per fornire accessi rapidi ai record dei database.

Supporto XML:

Il supporto ai documenti XML è fornito dal Foundation framework che fornisce la classe NSXMLParser per il recupero degli elementi e la libreria open source libxml2 per parsificare o scrivere dati XML in modo rapido e per trasformare i contenuti da XML a HTML.

Le sezioni seguenti descrivono i framework del livello Core Services ed i servizi che offrono:

Account Framework:

Introdotto in iOS 5, l'Account Framework (*Accounts.framework*) fornisce un modello d'accesso per gli account utente. Il framework è caratterizzato dalla funzione Single Sign-On che migliora l'esperienza degli utenti facilitandone l'accesso ai propri account, richiedendo una sola volta le credenziali che verranno immagazzinate per usi futuri.

Address Book Framework:

L'Address Book Framework (*AddressBook.framework*) fornisce l'accesso alla rubrica dei contatti memorizzata sul dispositivo. Se l'applicazione ha bisogno di utilizzare le

informazioni di un contatto, è possibile utilizzare questo framework per accedere e modificare i record nel database della rubrica. Un programma di chat, ad esempio, potrebbe utilizzare questo framework per recuperare l'elenco dei contatti disponibili per avviare una sessione di chat o per visualizzarli in maniera personalizzata. Da iOS 6, l'accesso ai contatti richiede l'autorizzazione esplicita da parte dell'utente.

Ad Support Framework:

Introdotto in iOS 6, il framework Ad Support (AdSupport.framework) fornisce l'accesso a un identificatore che le applicazioni possono utilizzare per scopi pubblicitari. Questo framework fornisce anche un flag che indica se l'utente ha scelto di monitorare determinati tipi di annunci.

CFNetwork Framework:

Il framework CFNetwork (CFNetwork.framework) è un insieme di interfacce C ad alte prestazioni che utilizzano astrazioni orientate agli oggetti per lavorare con i protocolli di rete. Queste astrazioni danno un controllo dettagliato sullo stack dei protocolli e rendono facile da usare i costrutti di livello inferiore, come le socket BSD. È possibile utilizzare questo framework per semplificare le attività, come la comunicazione con i server FTP e HTTP o per la risoluzione host DNS. Con il framework CFNetwork, è possibile:

- Utilizzare le socket BSD;
- Creare connessioni crittografate tramite SSL o TLS;
- Risolvere host DNS;
- Lavorare con server HTTP, di autenticazione HTTP, HTTPS e FTP;
- Pubblicare, risolvere e visualizzare i servizi Bonjour.

Core Data Framework:

Introdotto in iOS 3.0, il framework Core Data (*CoreData.framework*) è una tecnologia per la gestione del modello di dati di un'applicazione Model-View-Controller.

Il Core Data è concepito per essere utilizzato in applicazioni in cui il modello di dati è già altamente strutturato. Invece di definire complesse strutture di dati, è possibile utilizzare gli strumenti grafici di XCode per costruire uno schema rappresentante il modello. In fase di esecuzione, le istanze delle entità data-model vengono create, gestite e rese disponibili attraverso questo framework.

Il Core Data riduce in modo significativo la quantità di codice da scrivere in quanto gestisce in maniera automatica il modello dati delle applicazioni.

Core Data fornisce inoltre le seguenti caratteristiche:

- Immagazzinamento dei dati oggetto in un database SQLite per prestazioni ottimali;
- Una classe `NSFetchedResultsController` per gestire i risultati delle viste di una tabella;
- Gestione di undo / redo, al di là delle modifiche del testo;

- Il supporto per la validazione dei valori delle proprietà;
- Il supporto per la propagazione delle modifiche e la garanzia che le relazioni tra gli oggetti rimangano coerenti;
- Supporto per il raggruppamento dei dati, del filtraggio e dell'organizzazione in memoria.

Core Foundation Framework:

Il framework Core Foundation (*CoreFoundation.framework*) è un insieme di interfacce C che consentono la gestione dei dati di base e delle caratteristiche dei servizi per le applicazioni iOS.

Ciò include il supporto per: collezioni tipi di dati (array, set, e così via), bundle, gestione delle stringhe, gestione della data e del tempo, gestione di blocchi di dati raw, gestione delle preferenze, URL e manipolazione degli stream, thread e loop di esecuzione, porte e socket di comunicazione.

Il framework Foundation è strettamente correlato al framework Core Foundation, infatti il primo fornisce le interfacce Objective-C per le stesse caratteristiche di base del secondo.

Quando si ha la necessità di usare sia oggetti di Foundation e i tipi di Core Foundation, è possibile usufruire di un "toll-free bridging" esistente tra i due framework che permette di utilizzare i metodi e le funzioni di ciascun framework in maniera intercambiabile. Questo supporto è disponibile per molti dei tipi di dati, comprese le raccolte e le stringhe.

Core Location Framework:

Il framework Core Location (*CoreLocation.framework*) fornisce le informazioni sulla posizione e sulla direzione del dispositivo. Utilizza il GPS integrato, la rete cellulare, o il Wi-Fi per trovare la longitudine e latitudine corrente del dispositivo. È possibile incorporare questa tecnologia in applicazioni che ricercano luoghi d'interesse (come ristoranti e negozi) nelle vicinanze della posizione corrente del dispositivo.

In iOS 3.0, è stato aggiunto il supporto per l'accesso alle informazioni del magnetometro.

In iOS 4.0, Per ridurre i consumi dei servizi di localizzazione e monitoraggio, è stata introdotta una funzione che utilizza i ripetitori cellulari per rintracciare gli spostamenti dell'utente.

Core Media Framework:

Introdotto in iOS 4.0, il Core Media Framework (*CoreMedia.framework*) fornisce i tipi media (di basso-livello) utilizzati dal framework AV Foundation del livello superiore. L'utilizzo di questo framework non è indispensabile, tuttavia può essere utilizzato quando si ha bisogno di avere un maggior controllo sui contenuti audio e video.

Core Motion Framework:

Il framework Core Motion (*CoreMotion.framework*) fornisce un insieme unico di interfacce per accedere a tutti i dati di movimento di un dispositivo. Il framework supporta l'accesso ai

dati dell'accelerometro e del giroscopio (per i dispositivi provvisti). Questi dati possono essere sia raw che elaborati e permettono ad esempio di conoscere la velocità di rotazione del dispositivo o la direzione verso cui è rivolto. Sia l'accelerometro che il giroscopio, grazie a questo framework, possono essere utilizzati come input per alcune applicazioni come ad esempio mappe stellari o giochi.

Core Telephony Framework:

Introdotto in iOS 4.0, il framework Core Telephony (*CoreTelephony.framework*) fornisce le interfacce per interagire con le informazioni telefono-based sui dispositivi che hanno la possibilità di connettersi alla rete cellulare. Le applicazioni possono utilizzare questo framework per ottenere informazioni sui provider di servizi cellulari o sullo stato della rete (ad esempio la potenza del segnale cellulare).

Evento Kit Framework:

Introdotto in iOS 4.0, il framework EventKit (*EventKit.framework*) fornisce un'interfaccia per l'accesso agli eventi del calendario. È possibile utilizzare questo framework per ottenere gli eventi già esistenti o aggiungerne di nuovi. Questi eventi possono includere gli allarmi.

In iOS 6, sono stati aggiunti il supporto per la creazione e l'accesso ai promemoria dell'utente con relativa autorizzazione esplicita da parte dell'utente.

Foundation Framework:

Il framework Foundation (*Foundation.framework*) fornisce il supporto per le seguenti funzionalità: collezioni di tipi di dato (array, set, e così via), bundle, gestione delle stringhe, gestione della data e del tempo, gestione dei blocchi di dati Raw, gestione delle preferenze, URL e manipolazione di stream, thread e loop di esecuzione, bonjour, gestione delle porte di comunicazione, internazionalizzazione, matching di espressioni regolari, supporto alla cache.

Mobile Core Services Framework:

Introdotto in iOS 3.0, il framework Mobile Core Services (*MobileCoreServices.framework*) definisce i tipi di basso livello utilizzati in Uniform Types Identifiers (UTIs).

Newsstand Kit Framework:

L'applicazione Edicola, introdotta in iOS 5, permette di leggere le riviste ed i giornali pubblicate dagli editori. Gli editori, possono creare le proprie applicazioni utilizzando l'iOS Newsstand Kit Framework (*NewsstandKit.framework*), che consente di avviare in background il download di nuova rivista o di un quotidiano. Dopo aver avviato il download, il sistema gestisce l'operazione di scaricamento e notifica all'applicazione quando il nuovo contenuto è disponibile.

Pass Kit Framework:

Introdotto in iOS 6, Pass Kit (*PassKit.framework*) implementa il supporto a buoni, carte di imbarco e biglietti elettronici. Invece di trasportare fisicamente questi elementi, gli utenti possono memorizzarli sul proprio dispositivo iOS e utilizzarli all'occorrenza.

I “pass” vengono creati da un servizio web relativo alla società che li emette e successivamente consegnati sul dispositivo dell'utente via e-mail, Safari, o attraverso un’applicazione specifica. Lo stesso “pass” usa un formato speciale di file al quale viene applicata una firma crittografica prima di essere consegnato. Il formato del file identifica le informazioni pertinenti al servizio offerto in modo che l'utente possa riconoscerlo.

Quick Look Framework:

Introdotto in iOS 4.0, il framework Quick Look (*QuickLook.framework*) fornisce un'interfaccia per l'anteprima del contenuto dei file non supportati dall'applicazione. Questo Framework è orientato principalmente per applicazioni che scaricano file dalla rete o che comunque lavorano con file provenienti da fonti sconosciute. Alla ricezione del file, è possibile utilizzare la View-Controller fornita da questo framework per visualizzarne il contenuto nell'interfaccia utente.

Social Framework:

Introdotto in iOS 6, il Social Framework (*Social.framework*) fornisce una semplice interfaccia per l'accesso agli account dei social network. Questo framework sostituisce il framework Twitter precedentemente introdotto in iOS 5 e aggiunge il supporto ad altri account, tra cui Facebook e Sina Weibo. Le applicazioni possono utilizzare questo framework per inviare aggiornamenti di stato e le immagini di un account utente.

Store Kit Framework:

Introdotto in iOS 3, il framework Store Kit (*StoreKit.framework*) fornisce il supporto per l'acquisto di contenuti e servizi all'interno delle applicazioni iOS. Ad esempio, è possibile utilizzare questa funzione per consentire all'utente di sbloccare le funzionalità aggiuntive dell’applicazione. Il framework gestisce gli aspetti finanziari della transazione, l'elaborazione delle richieste di pagamento attraverso gli account utente dell’iTunes Store e fornisce all'applicazione informazioni sull'acquisto. Assicura, inoltre, che le transazioni avvengano in modo sicuro e corretto.

System Configuration Framework:

Il framework System Configuration (*SystemConfiguration.framework*) fornisce delle interfacce di connettività che è possibile utilizzare per determinare la configurazione di rete di un dispositivo. Questo framework può essere utilizzato per stabilire se una connessione Wi-Fi o cellulare è in uso o se un particolare server è accessibile.

1.2.4 Il livello Core OS

È lo strato che lavora più a basso livello e possiamo dire che è il cuore del sistema operativo. In questo strato sono gestiti i file system, vengono implementate le funzioni per la sicurezza del dispositivo, vengono gestiti i certificati e molto altro.

Uno dei compiti principali di questo strato è quello di gestire in maniera efficiente, e senza sprechi, l'energia messa a disposizione dalla batteria del dispositivo. Quando ad esempio un utente disabilita la rete wireless, verrà richiamata automaticamente dal sistema una routine che, di fatto, si occuperà di spegnere la scheda di rete del dispositivo e tutti i servizi correlati.

Nei dispositivi come iPhone e iPod Touch dove la quantità di energia erogata è molto limitata risulta di vitale importanza una gestione ottimale della stessa.

Il livello Core OS è quello su cui si basano le tecnologie dei livelli superiori. Anche se queste tecnologie non si utilizzano direttamente nelle applicazioni, esse possono essere usate da altri framework. In certi casi, si utilizzano le strutture in questo strato quando è necessario affrontare problematiche relative alla sicurezza o alla comunicazione con un accessorio hardware esterno.

I framework utilizzati in questo livello sono:

Accelerate Framework:

Introdotto in iOS 4.0, il framework Accelerate (*Accelerate.framework*) contiene le interfacce per l'esecuzione del Digital Signal Processor (DSP), dell'algebra lineare e dell'elaborazione delle immagini. Il vantaggio di utilizzare questo framework è che queste interfacce sono ottimizzate per tutte le configurazioni hardware di tutti i dispositivi basati su iOS, pertanto, se il codice scritto funziona correttamente su un dispositivo si ha la certezza che lo stesso funzioni in modo efficiente anche sugli altri.

Core Bluetooth Framework:

Il Core Bluetooth framework (*CoreBluetooth.framework*) consente agli sviluppatori di interagire specificamente con gli accessori Bluetooth Low-Energy. Le interfacce Objective-C di questo framework consentono principalmente di eseguire la scansione degli accessori LE, di collegare e scollegare quelli trovati, di gestire gli attributi del servizio, etc.

External Accessory Framework:

Introdotto in iOS 3.0, il framework External Accessory (*ExternalAccessory.framework*) fornisce il supporto alla comunicazione di accessori hardware esterni. Questi accessori possono essere collegati tramite un connettore a 30 pin o in modalità wireless tramite Bluetooth. Questo framework, inoltre, fornisce i metodi per ottenere le informazioni relative ad ogni accessorio disponibile e di avviare sessioni di comunicazione. In seguito l'accessorio può essere manipolato utilizzando tutti i comandi supportati.

Generic Security Services Framework:

Il Generic Security Services (*GSS.framework*) è un framework introdotto in iOS 5. Si occupa dello standard di sicurezza dei servizi connessi alle applicazioni iOS. Le interfacce di base sono specificate in IETF RFC 2743 e RFC 4401. Oltre ad offrire le interfacce standard, iOS include anche alcune aggiunte per la gestione delle credenziali, non specificate dagli standard, ma che sono comunque richieste da molte applicazioni.

Security Framework:

Oltre alle caratteristiche di base, iOS fornisce anche un framework esplicito per la sicurezza (*Security.framework*) che viene utilizzato per garantire l'integrità dei dati nelle applicazioni. Questo framework fornisce le interfacce per la gestione di certificati, chiavi pubbliche e private, e le "*trust policy*". Supporta la generazione di numeri pseudocasuali crittograficamente sicuri, la memorizzazione di certificati ed un archivio di chiavi crittografate per i dati sensibili degli utenti che prende il nome di "portachiavi" (*keychain*).

La libreria *Common Crypto* fornisce un ulteriore supporto per la crittografia simmetrica, HMAC e digest. Le funzioni di quest'ultime sono sostanzialmente compatibili con quelle della libreria OpenSSL, non disponibile in iOS.

Da iOS 3.0, è possibile condividere gli elementi del portachiavi tra più applicazioni, ciò ne rende più facile l'interazione. Ad esempio, è possibile utilizzare questa funzione per condividere le password degli utenti o altri elementi che potrebbero, altrimenti, essere richiesti da ogni applicazione li utilizzi.

Sistema:

Il livello di sistema comprende l'ambiente kernel, i driver e le interfacce UNIX di basso livello del sistema operativo. Il kernel è basato su Mach ed è responsabile di ogni aspetto del sistema operativo. Esso gestisce la memoria virtuale, i thread, il file system, la rete e la comunicazione tra processi. I driver di questo strato forniscono anche l'interfaccia tra l'hardware e i framework di sistema. Per motivi di sicurezza, l'accesso al kernel ed ai driver è riservato soltanto ad alcuni framework di sistema ed applicazioni.

iOS fornisce un insieme di interfacce, scritte in linguaggio C, per accedere a molte delle caratteristiche di basso livello del sistema operativo come threading (POSIX thread), networking (BSD socket), accesso al file-system, standard I/O, bonjour e servizi DNS, informazioni locali, allocazione della memoria e calcoli matematici. L'applicazione accede a queste funzioni tramite la libreria *LibSystem*.

1.3 TOOL DI SVILUPPO

Per sviluppare applicazioni per iOS, è necessario un computer Mac basato su architettura Intel Macintosh e l'ambiente di sviluppo integrato XCode (IDE) che può essere scaricato gratuitamente dall'Apple Store, previa l'iscrizione al programma Developer, o fornito in bundle con MacOS X (dalla versione 10.3 Panther).

XCode è, dalla versione 4.4.1, un'applicazione standalone (prima era una suite di programmi) per la realizzazione di software per MacOS X e iOS. Inizialmente nata per realizzare solo applicazioni per il sistema operativo dei Mac, dalla versione 3.1 aggiunge anche il supporto per iPhone e iPod touch mentre quello per iPad verrà aggiunto dalla versione successiva (XCode 3.2). Nasce da un'evoluzione del precedente tool di sviluppo della Apple, Project Builder (ereditato dalla NeXT), e ne estende le funzionalità.

Ha un'interfaccia molto chiara ed intuitiva e fornisce gli strumenti necessari per la creazione e la gestione dei progetti e dei file sorgenti. Inoltre, ha molte funzionalità che permettono di ridurre i tempi di sviluppo, come ad esempio il completamento automatico del codice e il compilatore GCC integrato che permette di compilare sorgenti scritti in C, C++, Objective C/C++ e Java.

Le caratteristiche tecnologicamente più avanzate sono il supporto alla distribuzione del lavoro di compilazione tra più macchine (grazie a Bonjour e Xgrid) ed il supporto alla compilazione incrementale che è in grado di compilare il codice mentre viene scritto. Queste caratteristiche riducono sostanzialmente il tempo di compilazione.

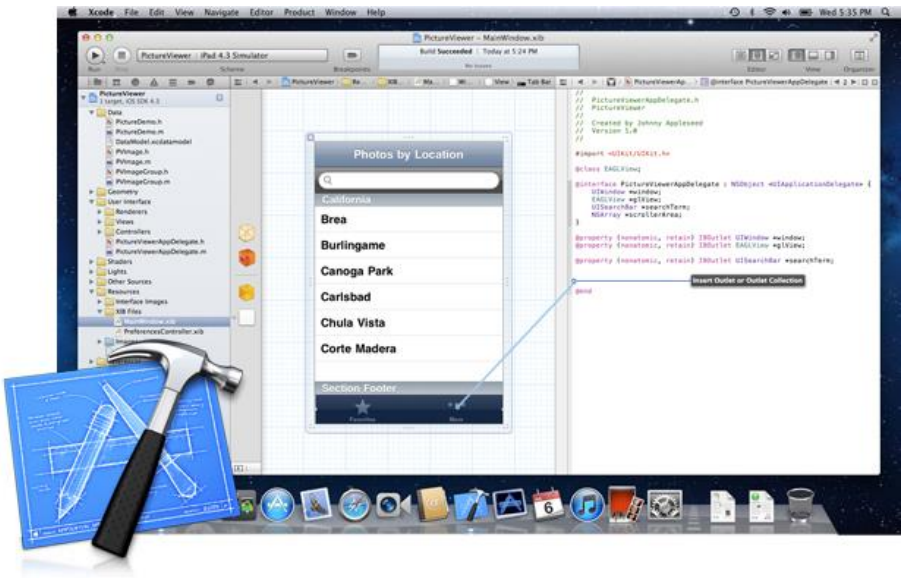

Figura 3 – L'interfaccia di XCode

Altri strumenti che permettono di assemblare l'interfaccia utente, eseguire il debug del codice generato e che forniscono una serie di programmi per l'analisi delle applicazioni sono rispettivamente Interface Builder, iOS Simulator ed Instruments.

L'**Interface Builder** è lo strumento usato per la realizzazione delle interfacce grafiche che permette di generare i file contenenti le viste delle applicazioni (file .xib, precedentemente .nib). L'utilizzo del tool è immediato. Per inserire e posizionare un elemento è sufficiente trascinarlo all'interno dell'area rappresentante una delle viste utente. Grazie a ciò, lo sviluppatore non dovrà più preoccuparsi di posizionare gli elementi via codice. Sebbene l'Interface Builder sia molto comodo da usare, non è esente da difetti, uno ad esempio è quello di non generare automaticamente il codice relativo all'oggetto utilizzato, nascondendo quindi allo sviluppatore alcune informazioni fondamentali.

L'**iOS Simulator** consente di eseguire ("provare") un'applicazione senza necessariamente disporre di un dispositivo fisico. Nonostante esso "virtualizza" il comportamento del dispositivo abbastanza fedelmente, in certi casi non aiuta a risolvere problematiche legate all'hardware presente sui dispositivi, in quanto molte funzionalità, come il GPS o le Notifiche Push, non sono presenti.

L'ambiente **Instruments** consente di analizzare le prestazioni delle applicazioni iOS durante l'esecuzione nel simulatore o nel dispositivo. Instruments raccoglie i dati e li presenta in forma grafica in una schermata chiamata timeline view. È possibile raccogliere i dati sull'utilizzo della memoria, sull'attività del disco, sull'attività di rete e le prestazioni grafiche. Lo strumento permette anche di tener traccia dei dati relativi a precedenti simulazioni, quest'ultima caratteristica consente il confronto delle varie modifiche apportate ad un'applicazione.

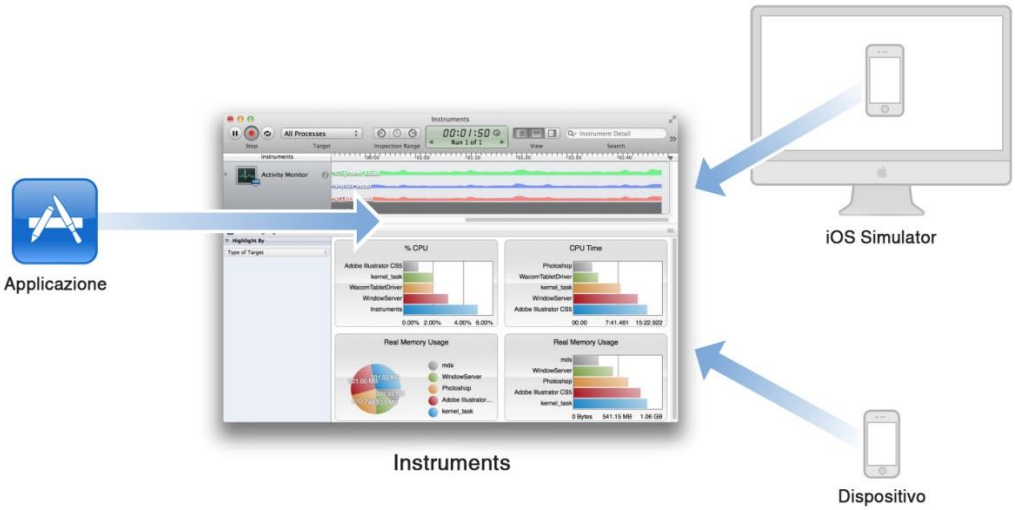

Figura 4 – Timeline view

1.4 CARATTERISTICHE DEL SISTEMA

1.4.1 Esecuzione applicazioni e multitasking

L'ambiente di runtime di iOS è progettato per avere un'esecuzione sicura e rapida dei programmi. Quest'ultimo aspetto è fondamentale se si considera che un utente, avendo a che fare con dispositivo mobile, debba poterlo usare in maniera immediata all'occorrenza. L'utente interagisce con una sola applicazione alla volta, per via delle dimensioni ridotte degli schermi dei dispositivi (rispetto a quelli dei computer), e quindi quando si lancia un'ulteriore applicazione quella precedente viene nascosta alla sua vista. Prima di iOS 4, un'applicazione veniva chiusa e rimossa dalla memoria, in seguito grazie all'avvento del multitasking, messa in secondo piano fino a quando l'utente non la rilancia nuovamente o rimossa dalla memoria, come avveniva nelle versioni precedenti. Il multitasking ha reso il riavvio delle applicazioni molto più veloce. Ciò non significa che tutte le applicazioni usate precedentemente siano rapide, a causa della memoria limitata infatti, alcune non usate di recente potrebbero essere eliminate dalla memoria in qualsiasi momento, richiedendo quindi un riavvio completo dell'app.

Con il multitasking ogni applicazione salva lo stato dell'applicazione al momento della chiusura, questo permette, al rilancio, di ripristinarne velocemente l'interfaccia, dando all'utente l'impressione che non sia stata mai chiusa. Le applicazioni messe in background risiedono nella memoria, ma non eseguono istruzioni. Per risparmiare la durata della batteria, allo scopo di evitare comportamenti impropri o perdita dei dati, il sistema gestisce opportunamente i cambi di stato. Gli stati in cui si può trovare un'applicazione sono:

- **Not running**: l'applicazione non è stata lanciata o lo era stata precedentemente, ma chiusa dal sistema;
- **Inactive**: l'applicazione è in esecuzione in primo piano, ma non riceve eventi. Generalmente l'applicazione rimane in questo stato solo per pochi istanti come transizione ad un diverso stato. Ci sono dei casi in cui un'applicazione può rimanere in questo stato più a lungo, ad esempio quando il sistema chiede l'interazione con l'utente in seguito ad una chiamata o un sms, oppure quando un utente blocca lo schermo;
- **Active**: l'applicazione è in primo piano e riceve eventi;
- **Background**: l'applicazione è in secondo piano ed esegue codice. La maggior parte delle applicazioni entrano in questo stato per poco tempo o quando stanno per essere sospese.
- **Suspended**: l'applicazione è in secondo piano, ma non esegue codice. Le applicazioni in questo stato possono essere chiuse senza preavviso dal sistema in caso di poca memoria.

La barra multitasking di iOS, accessibile con una doppia pressione del tasto home dei dispositivi, non è in realtà un task manager, ma una sorta di elenco cronologico di applicazioni aperte di recente. La prova di ciò è data dal fatto che, riavviando il dispositivo, tutte le applicazioni recenti saranno presenti sulla suddetta barra. In iOS, le applicazioni

generalmente vengono terminate ogni qual volta si preme il tasto home, infatti si può notate che non c'è traccia di una funzione relativa all'uscita dai programmi. Tecnicamente, però quel che accade è che alla pressione del tasto Home, le applicazioni restino 5 secondi nello stato di *Background* prima di passare a quello di sospese. Nello stato di sospensione (*Suspended*), l'applicazione rimane nella memoria in modo tale che possa essere rapidamente avviata dall'utente. Se ci sono troppe applicazioni sospese, iOS è abbastanza intelligente da eliminare quelle inutili spostandole nello stato *Not running*.

Ci sono alcuni casi in cui le applicazioni chiedano dei permessi speciali per l'esecuzione in background, in genere quando devono completare un compito specifico. Tali permessi consentono all'applicazione di rimanere nello stato Background per 10 minuti, invece che dei classici 5 secondi. Ad esempio, un'applicazione può dire al sistema operativo che è in fase di download, a quel punto iOS le concede 10 minuti di tempo per completare il suo compito in background, una volta trascorsi la mette nello stato di sospensione.

Ci sono poi alcune applicazioni che continuano a funzionare sempre in background, ciò influisce pesantemente sulla durata della batteria, una di queste è Mail (gestore della posta elettronica) che proprio per questo motivo è considerata una delle applicazioni più pesanti.

1.4.2 Threading

In iOS o in OS X ogni processo (o applicazione) è costituito da uno o più thread, ciascuno dei quali rappresenta una singola istanza del codice dell'applicazione. Ogni applicazione viene avviata con un singolo thread che si occupa di gestire la funzione main dell'applicazione e che può generarne altri.

Quando un'applicazione genera un nuovo thread esso diventa un'entità indipendente, all'interno dello spazio processo dell'applicazione, che ha un proprio stack di esecuzione, previsto per il runtime e separato dal kernel. Un thread può comunicare sia con quelli dell'applicazione stessa che con quelli di altri processi, eseguire le operazioni di I/O o svolgere qualsiasi altro compito per il quale sia stato generato. Poiché sono all'interno dello stesso spazio processo, tutti i thread di una singola applicazione condividono lo stesso spazio di memoria virtuale e hanno gli stessi diritti di accesso del processo padre.

In ogni programma ed a livello di sistema il threading ha un costo in termini di uso di memoria e performance. Ogni thread, infatti, richiede l'allocazione della memoria, sia all'applicazione che al kernel che immagazzina le strutture base necessarie per gestirlo e per coordinare il suo scheduling, nella memoria wired. Lo stack di un thread e di un pre-thread data, invece, è immagazzinato nello spazio di memoria del programma. La maggior parte di queste strutture sono create ed inizializzate quando viene creato il primo thread di un processo, ciò può essere relativamente costoso in termini di risorse, poiché richiede interazioni con il kernel.

Nella tabella seguente sono approssimati i costi associati alla creazione di un nuovo thread:

	Approssimazione costo	Descrizione
Strutture dati kernel	Circa 1KB	Questa memoria è usata per memorizzare le strutture dati e gli attributi del thread. Essendo gran parte allocati come memoria wired, su essi non può essere usata la tecnica del paging.
Stack	512 KB (thread secondari) 1 MB (iOS thread principale) 8 MB (OS X thread principale)	La grandezza minima allocata dello stack per thread secondari è 16 KB. Tale grandezza deve essere grande per multipli di 4 KB. Lo spazio per questa memoria viene messo da parte nella fase di creazione del thread.
Tempo di creazione	Circa 90 µs	Questo valore indica il tempo tra la chiamata iniziale per creare il thread e il momento in cui il thread inizia la sua routine.

Tabella 1 – Costo di creazione di un thread

Alcuni di essi possono essere configurati, come ad esempio la quantità di spazio allocato per i thread secondari.

I tempi di creazione di un thread possono variare notevolmente a seconda del carico del processore, della velocità del dispositivo e dalla quantità di memoria disponibile. Altro costo da tenere in considerazione quando si scrive il codice di un thread è quello di produzione, cioè il tempo necessario per implementare le sue funzioni.

Il progetto di un'applicazione che fa uso di thread, a volte può richiedere cambiamenti sostanziali nel modo di organizzare le strutture dati. Fare questi cambiamenti potrebbe essere necessario per evitare l'uso della sincronizzazione che talvolta influisce pesantemente sulle prestazioni, soprattutto se le applicazioni sono mal progettate.

La progettazione di tali strutture dati e i relativi problemi di debug, possono aumentare il tempo di sviluppo di un'applicazione che fa uso di thread. Evitare questi costi può comportare grossi problemi in fase di runtime, basti pensare a thread che impiegano troppo tempo per svolgere il loro compito o che addirittura non fanno nulla.

Run Loop

I run loop sono parte dell'infrastruttura fondamentale associata ai thread. Sono dei cicli di elaborazione che programmano il lavoro e coordinano la ricezione di eventi in entrata. Lo scopo di essi è quello di mantenere un thread occupato finché c'è un'operazione da svolgere e metterlo a "riposo" quando non c'è nulla da fare.

La gestione dei run loop non è completamente automatica. Quando si sviluppa il codice del thread, è necessario avviare il run loop al momento opportuno e gestire la risposta per gli eventi in arrivo. Sia *Cocoa* che *Core Foundation* forniscono degli oggetti che aiutano a configurare ed a gestire i run loop dei thread.

L'applicazione non ha bisogno di creare questi oggetti in modo esplicito dato che ogni thread, incluso quello principale, ha un oggetto run loop associato.

Sia in applicazioni Carbon che Cocoa, il thread principale imposta automaticamente il suo run loop e lo esegue come parte del processo di avvio dell'applicazione. Solo i thread secondari hanno bisogno di eseguire il proprio run loop in maniera esplicita.

Un run loop riceve eventi da due differenti tipi di fonti: **sorgenti Input** e **sorgenti Timer**. Le sorgenti Input si occupano di consegnare gli eventi asincroni, generalmente messaggi provenienti da un altro thread o da un'altra applicazione, mentre le sorgenti Timer eventi sincroni che si verificano in un momento programmato o in un dato intervallo. Entrambi i tipi di sorgenti utilizzano una routine di gestione che elabora ogni evento quando arriva.

La figura sottostante mostra la struttura concettuale di un run loop. Le sorgenti Input, come già detto, forniscono gli eventi asincroni per i corrispondenti gestori e causano la chiamata al metodo runUntilDate che permette di uscire dal ciclo. Le sorgenti Timer passano gli eventi alla loro routine di gestione ma non provocano l'uscita dal run loop.

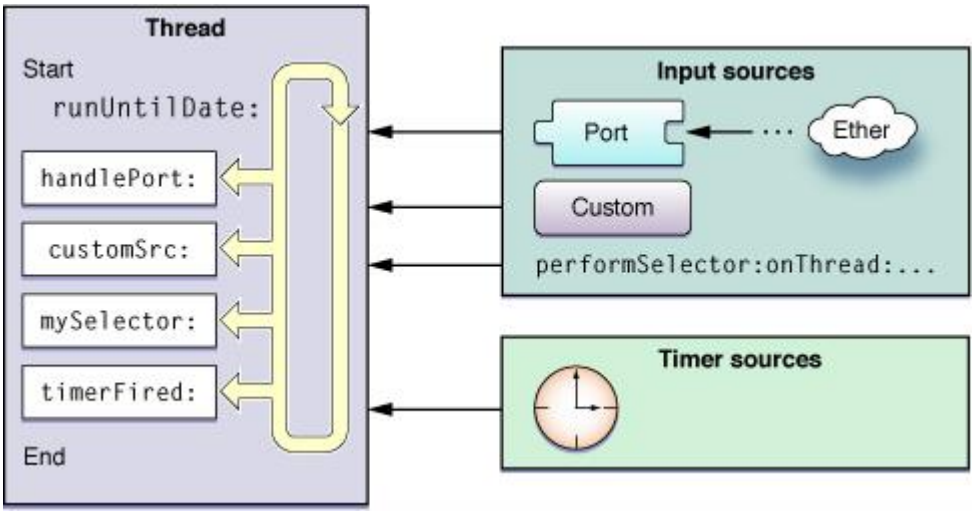

Figura 5 – Struttura di un run loop e le sue sorgenti

Oltre a gestire le fonti di ingresso, i run loop possono generare degli avvisi sul loro comportamento. Ricevono queste notifiche dagli osservatori che le usano per fare ulteriori elaborazioni sul thread. Per installare questi osservatori si utilizza il *Core Foundation*.

Grand Central Dispatch (GCD)

È una tecnologia sviluppata da Apple che ottimizza l'esecuzione delle applicazioni su sistemi multi core o su altri sistemi basati sul multiprocessing simmetrico. Essa implementa un parallelismo a livello di processi seguendo il thread pool pattern.

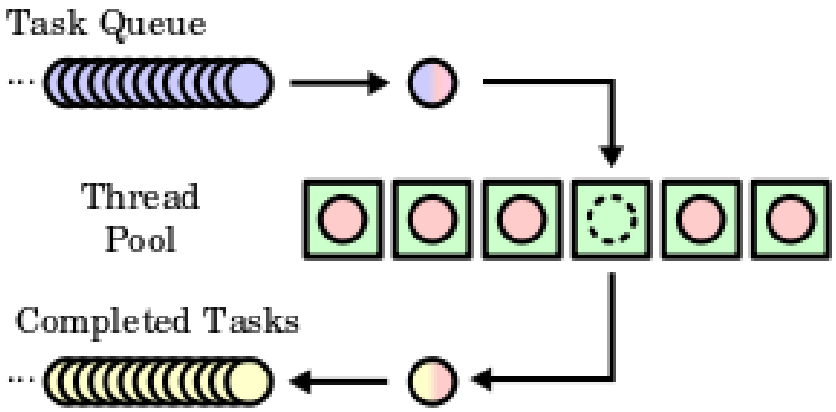

Figura 6 – Funzionamento del thread pool pattern

Il GCD consente al programmatore di delimitare delle porzioni specifiche di codice, chiamate blocchi, che possono essere eseguite in parallelo. Seguendo il pattern thread pool durante l'esecuzione del programma, i blocchi vengono messi in una coda e vengono eseguiti appena un'unità di elaborazione si rende disponibile.

Per eseguire questi blocchi, il GCD utilizza i thread, la cosa è totalmente trasparente e permette al programmatore di concentrarsi sullo sviluppo degli algoritmi, disinteressandosi della gestione del thread e della loro sincronizzazione. La creazione dei blocchi è un'operazione semplice e veloce che può essere svolta con sole 15 istruzioni mentre la creazione di un thread senza l'ausilio del GCD richiede molto più lavoro implementativo.

Un blocco creato con GCD può essere utilizzato per creare un'attività da mettere in una coda d'esecuzione o può essere assegnato ad una sorgente di eventi. Se un blocco è assegnato a quest'ultima, ogni qual volta si verifica un evento, viene attivato e messo in una coda d'esecuzione. Questa modalità di funzionamento, rispetto alla creazione di un thread al verificarsi di un evento, è considerata da Apple la più efficiente.

Il framework dispatch definisce diversi tipi di dati e funzioni:

- **Dispatch Queues:** code che contengono dei blocchi di codice o delle funzioni. La libreria crea automaticamente diverse code, con diversi livelli di priorità, ed esegue diversi task concorrenti selezionando il numero ottimale di quelli da eseguire, a seconda delle condizioni operative del momento. Un utilizzatore della libreria, può creare delle code seriali. In una coda seriale i task vengono eseguiti uno dietro l'altro e quindi l'esecuzione di un blocco è critico per l'esecuzione di quello successivo nella coda. Le code seriali possono essere utilizzate per gestire risorse condivise al posto di altre strutture come ad esempio i lock;

- **Dispatch Sources:** oggetti sui quali possono essere registrati dei blocchi di codice che vengono eseguiti in caso si scateni un evento, come ad esempio la creazione di un file o l'attivazione di un evento POSIX;
- **Dispatch Groups:** oggetti che raggruppano più blocchi. I blocchi sono aggiunti al gruppo e l'utente può utilizzare i dati elaborati quando tutti i blocchi sono stati eseguiti;
- **Dispatch Semaphores:** oggetti che permettono a chi usa la libreria di limitare l'esecuzione parallela solo ad alcuni blocchi.

1.4.3 Gestione della memoria

Uno degli aspetti che si deve tenere conto quando si sviluppa un'applicazione è la gestione corretta e senza sprechi della memoria dei dispositivi. Questo aspetto permette di rendere stabile il sistema in quanto a lungo andare, la memoria potrebbe esaurirsi e causare dei rallentamenti o addirittura dei blocchi del sistema. Bisogna tenere conto che ogni applicazione non ha l'uso esclusivo della memoria e ogni qual volta ne alloca una parte, deve in seguito liberarla per renderla disponibile alle altre applicazioni. Un programma ben scritto usa la memoria il meno possibile. I dispositivi iOS non posseggono un *Garbage Collector* (gestore automatico della memoria), la gestione della memoria è a carico degli sviluppatori ma, come vedremo a breve, grazie all'introduzione di una funzione di XCode 4.2, esiste la possibilità di automatizzare il tutto. In iOS, quindi, esistono due tecniche per gestire la memoria, una "manuale" (MRR - Manual Retain-Release) ed una "automatica" (ARC - Automatic Reference Counting).

Manual Retain-Release

In MRR lo sviluppatore si avvale di una proprietà, il `retainCount`, che indica il numero di riferimenti associati ad un oggetto allocato in memoria. Quando un qualsiasi oggetto viene allocato ed inizializzato con i classici metodi `alloc` e `init`, il suo `retainCount` ha valore 1. Per modificare in maniera esplicita il `retainCount` di un oggetto allocato, si possono usare due metodi che rispettivamente lo incrementano e decrementano di 1: `release` e `retain`.

Quando il `retainCount` di un oggetto arriva a 0, significa che quest'ultimo non è più utilizzato, quindi viene deallocato e distrutto mediante un oggetto chiamato `AutoreleasePool` che viene istanziato non appena l'applicazione viene lanciata.

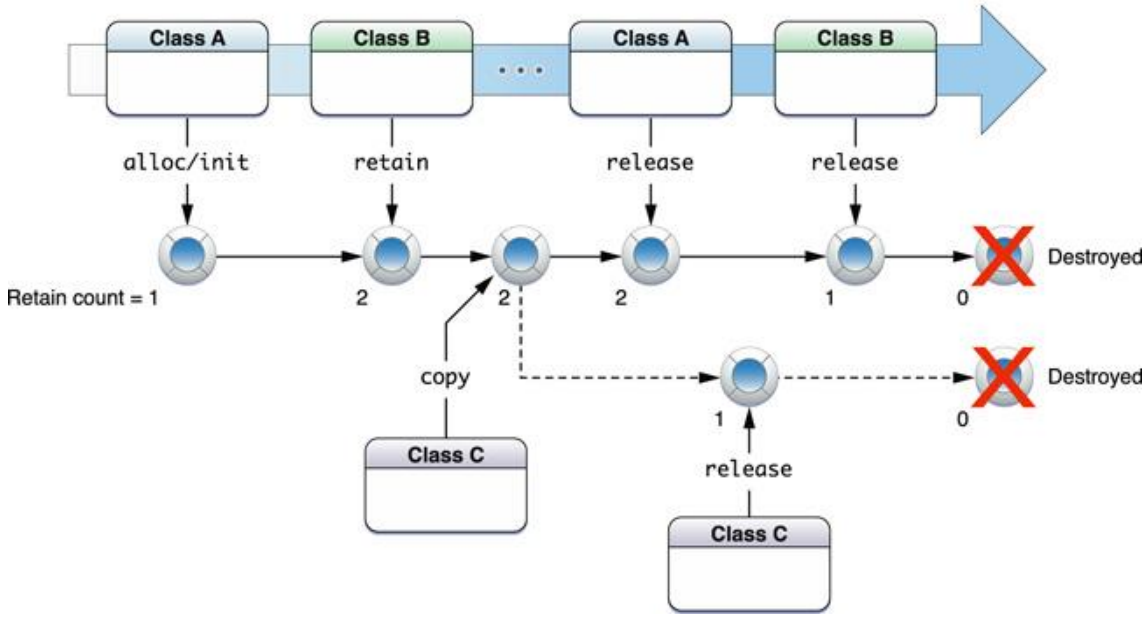

Figura 7 – Modello Release-Retain

Una cattiva ed errata gestione della memoria può portare ad uno dei seguenti problemi: un leak di memoria o un crash dell'applicazione.

Un leak di memoria si ha quando un `retainCount` di un oggetto è troppo alto. Tale oggetto, infatti, non verrà mai deallocato, in quanto possiede un `retainCount` maggiore di 1, causando uno "spreco" di memoria.

Un crash di un'applicazione può essere scaturito principalmente da un `retainCount` troppo basso che causa una deallocazione prematura dell'oggetto. Di per sé, questa deallocazione potrebbe non causare malfunzionamenti, però, se si suppone che un metodo invochi tale oggetto, l'applicazione proverà ad accedere ad un'area di memoria vuota e di conseguenza "crasherà".

Automatic Reference Counting

ARC è una tecnica, introdotta di recente, per la gestione automatica della memoria dei dispositivi. Contrariamente a quanto si possa pensare non è un garbage collector in quanto non lavora a runtime ma bensì a tempo di compilazione. Grazie ad ARC, lo sviluppatore non sarà più costretto ad invocare esplicitamente i metodi `retain` e `release` per mantenere corretto il valore del `retainCount` degli oggetti, in quanto verranno automaticamente aggiunti al codice in fase di compilazione. Ciò comporta una serie di vantaggi, per citarne qualcuno, la riduzione dei tempi di sviluppo delle applicazioni e la prevenzione degli problemi precedenti (leak di memoria e crash dell'applicazione).

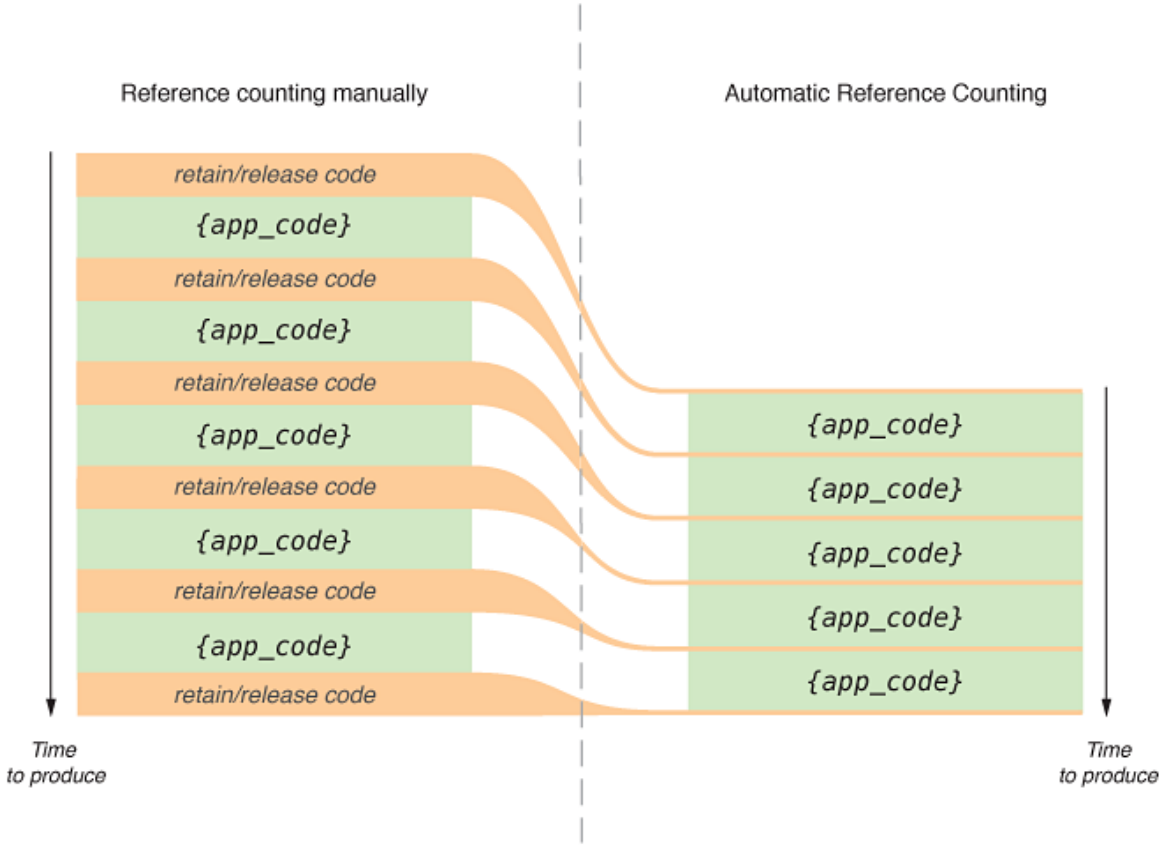

Figura 8 – Confronto tra MRR e ARC

1.4.4 Sandbox

La sicurezza in iOS è stata sin dal primo iPhone una delle priorità più importanti. Per garantire tale scopo, il sistema implementa delle funzionalità che salvaguardino l'integrità dei dati degli utenti e che impediscano alle applicazioni di interferire tra di loro.

Per ogni applicazione installata su iOS viene creata una directory sul file system (/var/mobile/Application/ID_APPLICAZIONE) al cui interno sono presenti alcune sottodirectory fondamentali (quella contenente i file dell'applicazione, Documents, Library e tmp) e definita una sandbox.

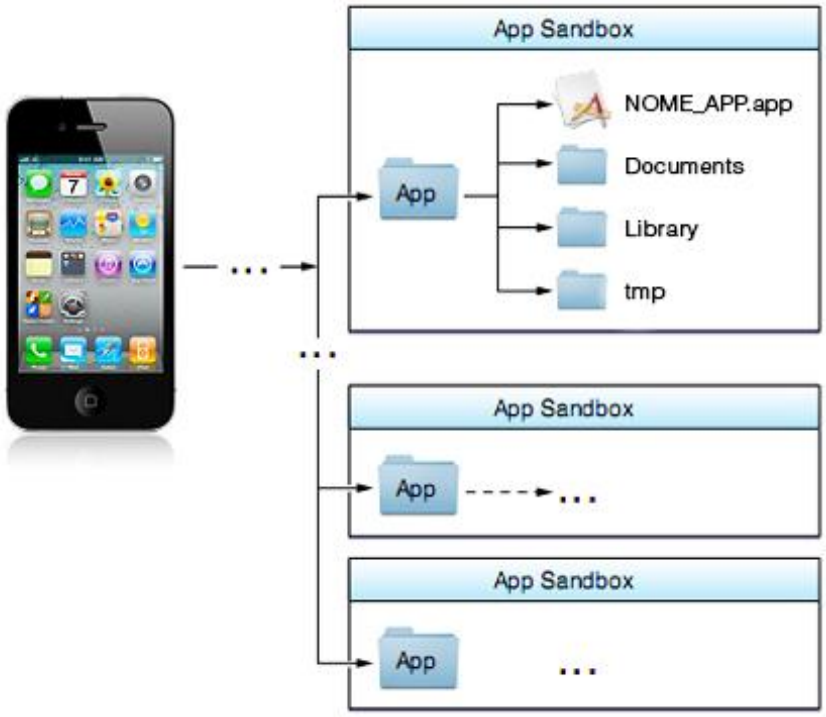

Figura 9 – Contenuto di una sandbox

NOME_APP.app: è la directory dove risiede l'applicazione

Documents: è la directory utilizzata per memorizzare i documenti dell'utente e i file dell'applicazione. Il contenuto di questa cartella è condiviso con il computer grazie all'ausilio di iTunes e permette quindi lo scambio di dati;

Library: contiene tutti i file non appartenenti all'utente, ma che servono all'applicazione. È la directory usata per memorizzare le preferenze e la cache. Anche il contenuto di questa cartella è condiviso ad eccezione della sottocartella "/Caches";

tmp: è una directory contenente i file temporanei che non hanno necessità di persistere tra i diversi lanci dell'applicazione. Il contenuto di questa cartella è provvisorio e i file al suo interno devono essere eliminati dall'applicazione quando non più necessari. Il sistema può eliminare il contenuto in maniera autonoma quando l'applicazione non è in esecuzione

Una sandbox è un insieme di controlli che limitano, per ragioni di sicurezza, l'accesso a file, servizi di sistema, risorse di rete, hardware, preferenze, etc.

In iOS, ogni applicazione ha la propria sandbox, non può accedere a quella delle altre e inoltre può crittografare i dati al suo interno al fine di renderli inaccessibili, anche a se stessa, quando il dispositivo è bloccato. Per sbloccare questi dati l'utente deve sbloccare il dispositivo inserendo un appropriato codice di accesso.

Un'applicazione senza sandbox, a differenza di quella con, ha un accesso completo sia a tutti i dati presenti nel dispositivo che a tutti i servizi di sistema, questo aspetto può rivelarsi pericoloso se si considera che un bug nell'applicazione potrebbe permettere il controllo completo del sistema.

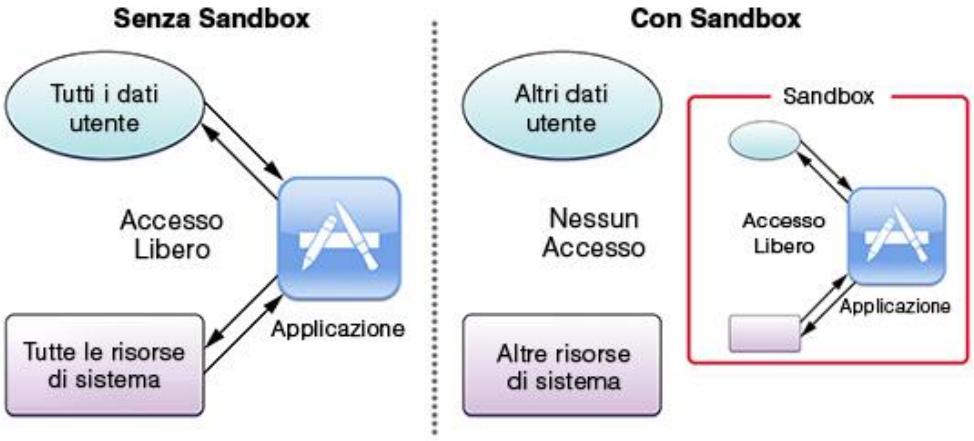

Figura 10 – Differenza tra applicazioni senza e con sandbox

1.5 CENNI STORICI

iOS venne presentato il 9 gennaio 2007 alla Macworld Conference & Expo. In tale occasione Steve Jobs afferma che il nuovo sistema si basa su MacOS X Leopard e nonostante sia ben lontano dalla versione desktop, si rileva decisamente comodo grazie al touchscreen multi touch, principale sistema di interazione vista la quasi totale assenza di tasti nei dispositivi di casa Cupertino. Il primo rilascio risale al 29 giugno dello stesso anno, in concomitanza con l'uscita del primo iPhone. La versione 1.0, inizialmente priva di nome, fu denominata "iPhone OS" e includeva esclusivamente solo le App sviluppate da Apple. Inizialmente l'unico software non sviluppato dalla casa madre erano soltanto le applicazioni web-based accessibili dal browser Safari.

La possibilità di installare applicazioni di terze parti si ebbe dalla versione 2.0, quando fu introdotto il tanto atteso App Store, il primo market che dava la possibilità di scaricare applicazioni gratuite o a pagamento.

La terza release di iPhone OS, fu presentata con l'iPhone 3GS il 17 giugno 2009, tra le più importanti novità: la registrazione di video, il copia-incolla (caratteristica richiesta dagli utenti), la navigazione GPS, il supporto a HTML5 in Safari, il tethering via USB e Bluetooth, etc.

L'evoluzione di questa versione non fu dettata solo dai progetti redatti a Cupertino, ma anche dal software sviluppato dalla comunità di "jailbreaker" pubblicato sul market alternativo Cydia (accennato nel paragrafo 1.1).

Tutti i dispositivi erano aggiornabili a iPhone OS 3, ma con delle limitazioni per la prima generazione di iPhone e iPod touch.

Nel giugno del 2010, "iPhone OS" venne rinominato in "iOS". La novità più importante fu il multitasking, cioè la possibilità di eseguire più applicazioni in contemporanea, altre migliorie introdotte furono le cartelle, FaceTime per videochiamare gratuite via Wi-Fi, lo zoom digitale nella fotocamera, la registrazione di video HD 720p, il supporto in modalità panorama aggiunto a molte applicazioni preinstallate ed altro. Con questa versione si passa ad un sistema libero da molte irrazionali limitazioni e abbastanza completo sotto il punto di vista delle funzioni.

In contemporanea con l'uscita dell'iPhone 4S, nel 2011, fu rilasciato iOS 5. Furono introdotte numerose nuove funzioni, tra cui la sincronizzazione wireless, l'integrazione con il servizio iCloud di Apple, iMessage (servizio di messaggistica gratuito) e un rinnovato e completo sistema di notifiche. Altra funzione degna di nota, fu l'introduzione dell'assistente vocale Siri, quest'ultima però disponibile solo per iPhone 4S.

Nel giugno del 2012, Apple ha presentato alla WWDC (Worldwide Developers Conference), la versione 6 di iOS nella quale spiccano un'applicazione mappe completamente rinnovata, nuove funzioni e lingue per l'assistente vocale Siri, tra cui la presenza della lingua italiana, l'integrazione al livello di sistema con Facebook e altre funzionalità e novità grafiche.

La versione 7 di iOS viene presentata il 10 giugno del 2013 con uno stile grafico completamente rinnovato e adattato allo stile "piatto" e "minimal". Tale stile presenta icone

molto più semplici e colorate. Una differenza con le precedenti versioni di iOS è la rimozione della barra di sblocco, presente fin dalla prima release, che viene sostituita da una schermata più semplice e minimalista. Altro rinnovamento sostanziale è la totale revisione del multitasking, del quale sono stati modificati look e funzionalità, rendendolo più attuale e al pari dei sistemi concorrenti. Inoltre, in dotazione agli sviluppatori, sono state presentate molte altre novità, fra le quali l'introduzione di nuove 1500 API e una nuova versione dell'ambiente di sviluppo, XCode 5.

CAPITOLO 2
IL LINGUAGGIO DI PROGRAMMAZIONE DI IOS

2.1 L'OBJECTIVE-C

Objective-C (o abbreviato ObjC) è un linguaggio di programmazione orientato agli oggetti derivante dal linguaggio C che viene utilizzato principalmente per lo sviluppo di applicazioni che girano sui sistemi operativi MacOS X e iOS.

Mantiene piena compatibilità con le strutture e la sintassi del precedente linguaggio estendendone le funzionalità. Come è facilmente intuibile, l'Objective-C non è un linguaggio di programmazione di nuova concezione, come ad esempio il Java o il C#, e quando si dice che è basato sul C, s'intende che quest'ultimo è parte integrante del linguaggio, infatti alcune parti del codice possono essere scritte in puro linguaggio C con l'aggiunta di classi e oggetti definiti tramite le estensioni di ObjC.

Le caratteristiche dell'Objective-C, in particolar modo la gestione dei messaggi, sono ispirate da Smalltalk, un linguaggio di programmazione sviluppato allo Xerox PARC che ha ispirato anche altri linguaggi come Java e C#.

L'ObjC è stato creato da Brad Cox e Tom Love alla Stepstone, azienda di piccole dimensioni che fu acquisita dalla NeXT nel 1995. Quest'ultima fu fondata dieci anni prima da Steve Jobs, dopo che questi fu costretto ad abbandonare la Apple. La NeXT non riscosse molto successo nella vendita delle sue unità ma il suo sistema operativo orientato agli oggetti, NeXTstep, e il relativo ambiente di sviluppo lasciarono il segno.

Nel 1996 Apple acquistò NeXT e in particolare l'ambiente di programmazione OPENSTEP che fu utilizzato in seguito come base per MacOS X e i suoi tool di sviluppo. In MacOS X e iOS rimane ancora traccia di NeXTstep, infatti, molte delle classi della libreria di base, che si utilizzano per lo sviluppo delle applicazioni, iniziano per NS (acronimo di NextStep).

Diversamente dai linguaggi citati precedentemente, l'Objective-C è un linguaggio debolmente tipizzato che introduce il concetto di direttiva, definizione caratterizzata dalla presenza della chiocciola iniziale (@) che serve per marcare in maniera esplicita gli elementi sintattici specifici di questo linguaggio.

Uno degli scopi della nascita di Objective-C è quello di aggiungere la programmazione orientata agli oggetti al C. Come Java e C#, il linguaggio supporta l'ereditarietà singola ed è anche possibile ottenere i vantaggi di quella multipla utilizzando metodi alternativi, ciò per mantenere semplice l'organizzazione del software, riducendo la complessità della gerarchia di classi.

Objective-C supporta inoltre il polimorfismo e la ridefinizione dei metodi. Le classi astratte non sono supportate ma possono essere simulate attraverso altri strumenti.

2.2 LA SINTASSI

L'Objective-C è, come detto precedentemente, un'estensione del linguaggio C e pertanto i sorgenti di quest'ultimo possono essere interpretati correttamente da un compilatore Objective-C. Dal C deriva gran parte della sintassi (clausole del preprocessore, espressioni, dichiarazioni e chiamate di funzioni), mentre quella relativa alle caratteristiche object-oriented è stata creata per ottenere la comunicazione a scambio di messaggi simile a quella di Smalltalk. Il modello di programmazione dell'Objective-C, quindi, è lo scambio di messaggi tra oggetti. Tale modello è differente da quello di Simula che è il primo linguaggio di programmazione object-oriented dal quale derivano sia lo Smalltalk che il C++. Questa distinzione è semanticamente importante e consiste principalmente nel fatto che in Objective-C non si chiama un metodo, ma si invia un messaggio. La sintassi seguente esprime meglio quest'ultimo concetto:

```
[ogg operazioneDaEffettuare];
```

In questo caso `ogg` rappresenta un oggetto che risponde al messaggio `operazioneDaEffettuare`, l'equivalente in C++ o in Java sarebbe:

```
ogg.operazioneDaEffettuare();
```

Qui, `ogg` rappresenta sempre un oggetto, ma `operazioneDaEffettuare()` è il metodo chiamato dall'oggetto.

Nel primo caso è possibile inviare messaggi ad un oggetto anche se esso "non è capace" di rispondere. Ciò differisce dai linguaggi tipizzati staticamente (C++, Java, etc…) in quanto, in quest'ultimi, tutte le chiamate devono essere di metodi predefiniti.

2.3 CLASSI ED OGGETTI

Uno dei concetti base della programmazione ad oggetti è quello di classe, cioè un costrutto che definisce una struttura, o schema, utilizzata per creare oggetti in fase di esecuzione. Una classe può rappresentare concetti concreti o astratti all'interno di un sistema software.

In Objective-C ogni oggetto deriva da una classe base, tipicamente un oggetto root, chiamato così per indicare che non deriva da nessun'altro. L'oggetto root dal quale derivano quasi tutte le classi di Objective-C è `NSObject`; quelle che non derivano da esso tuttavia implementano un protocollo (corrispettivo di interfaccia nel mondo java o C#) che li riconduce come comportamento a `NSObject`. Nella pratica si può tranquillamente dire che ogni oggetto deriva da `NSObject`.

Esattamente come accade nel C ogni definizione di classe prevede due parti: un header e l'implementazione.

L'header (file .h) permette di utilizzare la classe in altre applicazioni e pertanto è la parte pubblica. Include tutte le dichiarazioni delle variabili di istanza della classe, delle proprietà e delle firme dei metodi;

L'implementazione della classe (file .m) contiene il codice vero e proprio di ogni metodo e il completamento delle proprietà dichiarate nell'interfaccia, contrariamente all'header è la parte privata.

2.3.1 Variabili di istanza

Ogni oggetto ha un suo stato interno che viene rappresentato dalle variabili di istanza. Queste variabili sono considerate private per default e non possono essere modificate né lette da oggetti esterni. Per leggere o modificare il valore di queste variabili che sono le proprietà dell'oggetto, tuttavia si possono utilizzare i classici metodi setter e getter (metodi accessori).

Le variabili di istanza possono essere dei tipi scalari del linguaggio C (int, float, decimal, etc…) oppure altri oggetti Objective-C come ad esempio le stringhe, rappresentate dalla classe `NSString` (da notare la presenza dell'oggetto stringa mancante nel C). La sintassi per dichiarare una variabile di istanza è la seguente:

```
<tipo> <nomeVariabile>;
```

Dove `<tipo>` definisce la tipologia della variabile (int, float, double, …) e `<nomeVariabile>` un nome arbitrario per riconoscerla e rendere più leggibile il codice.

Esempio:

```
// Dichiara una variabile di nome contatore di tipo intero.
int contatore;
```

Inoltre, come nel C, esiste la possibilità di definire una variabile puntatore. Essa si definisce inserendo un asterisco (*) prima del nome della variabile. Quest'ultima caratteristica è fondamentale in quanto tutti gli oggetti di ObjC sono riferiti tramite puntatori.

2.3.2 Metodi di istanza e metodi di classe

In Objective-C si possono distinguere due tipologie di metodi: i metodi di istanza e i metodi di classe.

I primi si usano per effettuare operazioni su una particolare istanza della classe, come ad esempio per ottenere o modificare valori, effettuare controlli, etc. La dichiarazione di questi metodi è sempre preceduta dal segno meno (–), seguito dal tipo di ritorno tra parentesi tonde, e dal nome del metodo (è buona norma seguire, per convenzione, la sintassi CamelCase ad eccezione dell'iniziale del nome del metodo che deve essere sempre minuscolo).

I secondi (che in Java si chiamano "metodi statici") si usano per effettuare operazioni sulla classe stessa e sono quei metodi che vanno invocati direttamente dalla classe e non da una particolare istanza di essa. Un esempio di metodo di classe è il metodo `init`, cioè quello utilizzato per istanziare la classe. I metodi di classe sono dichiarati in modo analogo a quelli di istanza, cambia solamente il segno che precede la definizione che in questo caso è un più (+), e la modalità di invocazione.

La sintassi per dichiarare tali metodi è la seguente:

```
[+|-] (<tipoRitorno>) nomeMetodoParametro1: (<tipo1>) param1
                      parametro2: (<tipo2>) param2
                      …
                      parametroN: (<tipoN>) paramN;
```

Dove:

- +|- indica se il metodo è di classe o d'istanza;
- tipoRitorno è il tipo o classe di ritorno del metodo;
- nomeMetodo è l'identificativo del metodo;
- parametro# e param# contengono il nome dei parametri;
- tipo# è il tipo di parametro.

Esempi:

```
+(id) alloc;
+(id) sharedInstance;
-(void) setNome: (NSString *) unNome cognome: (NSString *) unCognome;
```

2.3.3 Ereditarietà e polimorfismo

EREDITARIETÀ:

L'ereditarietà è la caratteristica che permette ad un oggetto di ricevere metodi e proprietà da una classe già esistente, per estenderne le funzionalità o per specializzarla. Proprio per questo motivo la classe da cui si eredita è tipicamente più astratta. Ad esempio la classe `MezzoDiTrasporto` (classe base, o superclasse) può essere estesa dalla classe `Automobile` (classe derivata, o sottoclasse).

L'utilità di ereditare una classe agevola la stesura di una nuova. Lo sviluppatore non ha la necessità di scrivere altro codice sorgente in quanto quello esistente può essere riutilizzato modificandone solo qualche dettaglio. Nella classe derivata, è possibile anche ridefinire dei membri della superclasse, oppure aggiungerne di nuovi.

L'estensione di una classe base, come pure la dichiarazione di una classe, avviene nel file di interfaccia .h, grazie alla direttiva `@interface`:

```
@interface MezzoDiTrasporto: NSObject
```

In questo caso si eredità l'oggetto base `NSObject`. In effetti se dovessimo ereditare la classe `MezzoDiTrasporto`, il meccanismo sarebbe del tutto analogo:

```
@interface Automobile: MezzoDiTrasporto
```

Costruttore

In Objective-C il costruttore della classe è composto dalla chiamata di due metodi annidati `alloc` e `init`. Il primo serve per l'allocazione in memoria della classe, il secondo, quello che più somiglia al concetto di costruttore in java, si occupa di inizializzare tutte le variabili che verranno utilizzate dalla classe.

La definizione della classe `MezzoDiTrasporto` dell'esempio precedente è la seguente:

```
MezzoDiTrasporto * mezzo = [[MezzoDiTrasporto alloc] init];
```

L'oggetto super

Quando un oggetto vuole utilizzare i metodi o le variabili della classe da cui deriva, può farlo utilizzando la parola chiave `super`. Questa è un riferimento alla superclasse ed è indispensabile quando si vuole creare un metodo di inizializzazione diverso da quello standard `init` o quando occorre fare l'override di un metodo della superclasse, ad esempio `dealloc`.

L'oggetto self

L'oggetto `self` è il riferimento di una classe a se stessa. È utilizzato quando si vuole mandare un messaggio interno alla classe, cioè invocare un metodo della stessa classe, o quando si intende modificare il valore di una variabile di istanza.

Esempio:

```
NSString *stringa = [self nomeMetodoDellaClasse];
```

Override

È possibile effettuare l'override di un metodo definito nella superclasse semplicemente creando un metodo con lo stesso nome e con gli stessi parametri. All'interno di questo metodo è possibile riferirsi a quello della superclasse attraverso l'oggetto `super`.

Se volessimo effettuare l'override del metodo `init` della superclasse, dovremmo fare quanto segue:

```
- (id) init
{
    if (self = [super init]) {…}
    return self;
}
```

Overload dei metodi

In Objective-C non è possibile effettuare l'overload dei metodi come in Java; anche se è possibile ad esempio scrivere il seguente codice:

```
- (id) init;
```

```
- (id) initWithVariabile1: (int) var1 variabile2: (float) var2;
```

Qualora una classe implementi più metodi `init` occorre sempre effettuare l'override del metodo `init` (senza parametri), per evitare che si possa inizializzare la classe in modo anomalo utilizzando quello della superclasse.

Quando si effettua l'override del metodo `init` si può anche richiamare uno dei metodi `init` più specifico, passandogli dei valori di default. Prendendo come riferimento i due metodi dell'esempio precedente:

```
- (id) init
{
    return [self initWithVariabile1: 0 variabile2: 2.0F];
}
```

```
- (id) initWithVariabile1: (int) var1 variabile2: (float) var2
{
    if ( self = [super init] )
    {
        // Inizializzazione delle variabili di classe.
    }
    return self;
}
```

POLIMORFISMO:

Il polimorfismo è la capacità da parte di un oggetto di comportarsi in maniera differente a seconda del contesto dove si trova ad operare. Anche in Objective-C è presente, tuttavia non viene utilizzato come in altri linguaggi. Nella maggior parte dei linguaggi di programmazione ad oggetti, in genere è implementato in due modi diversi: 1) attraverso l'ereditarietà; 2) attraverso le interfacce. Nonostante entrambi questi metodi siano supportati in Objective-C, può essere sfruttato il fatto che sia un linguaggio dinamico e quindi, il polimorfismo, può essere implementato attraverso l'uso dei protocolli (paragrafo 2.4)

Per chiarire questo concetto si può considerare l'esempio precedente (quello relativo alle classi `MezzoDiTrasporto` e `Automobile`). Per conoscere la capacità del numero di posti di un oggetto `Automobile` si può utilizzare un metodo `numPosti` (in java sarebbe: `getNumPosti`), lo stesso lo si potrebbe utilizzare anche per gli oggetti `Aereo` e `Treno`. Da ciò ne segue che `Automobile`, `Aereo` e `Treno` da questo punto di vista sono uguali, dato che rispondono allo stesso messaggio. Objective-C è un linguaggio non fortemente tipizzato, quindi implicitamente tutti gli oggetti sono polimorfici: è possibile infatti inviare un messaggio qualsiasi ad un oggetto, senza che il compilatore se ne lamenti troppo (produce un avviso, ma non è bloccante). Se il metodo corrispondente non è presente ovviamente non succederà nulla. Il compilatore, in questi casi, si comporta in maniera permissiva perché anche se la classe non contiene un metodo adatto a rispondere al messaggio indicato, è possibile che questo sia implementato altrove, oppure reso disponibile a runtime con meccanismi dinamici.

2.3.4 Proprietà

Nelle classi spesso sono presenti metodi con il preciso e unico scopo di restituire o impostare il valore di campi della classe. Questi assumono il nome di metodi accessori (o d'accesso), detti anche getter e setter. Per convenzione questi metodi hanno prefisso set e get seguito dal nome della variabile che intendono esporre. Ovviamente, il metodo con prefisso set serve per impostare il valore della variabile, quello con prefisso get serve per recuperare il valore della variabile. Essi permettono di implementare le proprietà che sono gli attributi dell'oggetto, il cui accesso è mediato da metodi, e che quindi non viola uno dei principi fondamentali della programmazione object-oriented, l'incapsulamento.

In Objective-C 2.0 è stata introdotta una nuova sintassi per la dichiarazione delle proprietà che, nella maggior parte dei casi, evita al programmatore di scrivere i metodi getter e setter. Essa è formata da due parti, una da inserire nel file di interfaccia e l'altra da scrivere nel file di implementazione.

Successivamente il compilatore si occuperà di sostituire queste dichiarazioni con i metodi getter e setter secondo le indicazioni dello sviluppatore.

Per creare una proprietà occorre:

- Inserire una proprietà nell'interfaccia;
- Definire i metodi di accesso (sempre nell'interfaccia);
- Implementare i metodi nel file di implementazione.

Nel file di interfaccia le proprietà vengono dichiarate nel seguente modo:

```
@property (assign, readwrite) int intero;
@property (retain, readwrite) NSString *stringa;
```

Gli attributi tra parentesi tonde sono le informazioni necessarie al compilatore per scrivere correttamente i metodi setter e getter della proprietà e sono:

- **`getter = nomeGetter`**: consente di personalizzare il nome del metodo `getter`;
- **`setter = nomeSetter`**: permette di definire il nome del metodo `setter`;
- **`readwrite`**: (default) indica che la proprietà è in lettura/scrittura, e che quindi verranno generati sia `getter` che `setter`;
- **`readonly`**: Indica che la proprietà è in sola lettura, quindi verrà generato solo il metodo `getter`;
- **`assign`**: (default) indica che la proprietà è valorizzata nel `setter` con una semplice assegnazione del parametro.
- **`retain`**: specifica che al parametro ricevuto dal `setter` viene inviato un messaggio `retain`. Al valore precedente viene inviato un messaggio `release`;
- **`copy`**: chiede al compilatore di generare il codice che assegni alla variabile d'istanza della classe una copia del parametro. Al valore precedente viene inviato un messaggio `release`;

- **`nonatomic`**: specifica che i metodi accessori non sono atomici e pertanto non protetti dalla possibile esecuzione concorrente da parte di più thread.

Infine, per completare la definizione dei parametri occorre specificare nel file di implementazione le direttive `@synthesize`:

```
@synthesize intero;
@synthesize stringa;
```

Se questa parte viene omessa nel file di implementazione, il compilatore non genererà automaticamente i `setter` e i `getter`, e quindi questi metodi possono essere implementati dagli sviluppatori. Questo meccanismo può risultare utile, ad esempio, per validare i valori passati al `setter`.

2.4 PROTOCOLLI

I protocolli definiscono metodi che possono essere usati da qualunque classe. Essi vengono generalmente scritti come un lista di metodi distaccati dalla definizione di classe, tali metodi verranno implementati in seguito dall'utilizzatore. I protocolli possono essere definiti quindi come un'astrazione delle azioni che possono essere compiute, ovvero si definisce un'azione che un oggetto può compiere, come ad esempio interagire con una risorsa, ma non si conosce nello specifico come questo oggetto intende effettuare questa interazione. Grazie all'ausilio dei protocolli si è in grado di rispondere ad una specifica azione senza conoscere come questa avvenga.

I protocolli si differenziano in due categorie: protocolli formali e protocolli informali.

Protocolli formali

I Protocolli formali sono quelli che hanno una struttura ben definita all'interno di un file dedicato appositamente a quest'ultimi. Un protocollo formale è definito all'interno di due direttive `@protocol` e `@end` che ne definiscono rispettivamente l'inizio e la fine.

È possibile definire se i metodi implementati all'interno di un protocollo sono necessari o opzionali. Per far ciò si utilizzano le due parole chiavi `@required` e `@optional` prima di un metodo o di una lista di essi. Se non si specifica nessuna tipologia verrà assegnato di default il tipo `@required`. Nella pratica, però, è buona norma dichiarare sempre la tipologia ai fini di rendere il codice più leggibile.

Un esempio semplice di protocollo è il seguente:

```
@protocol NomeProtocollo

@required
- (void) metodo1;
- (void) metodo2;

@optional
- (void) metodoOpzionale;
- (void) altroMetodoOpzionale;

@end
```

Protocolli informali

I protocolli informali vengono implementati inserendo i metodi nella dichiarazione di una categoria:

```
@interface NSObject (NomeProtocollo)
- (void) metodo1;
- (void) metodo2;
@end
```

Solitamente si sceglie di creare una categoria di `NSObject`, in questo modo non sono soggetti a nessuna gerarchia e possono essere implementati praticamente ovunque. In questo caso, come nel precedente, non c'è bisogno di fornire un'implementazione del protocollo, che sarà demandata, come sempre a chi utilizza i protocolli. I protocolli informali non sono soggetti a nessun tipo di controllo da parte del runtime system.

Come si utilizza un protocollo

Un protocollo può essere utilizzato da una classe o da una categoria che ne definisce un altro. Per utilizzare un protocollo basta racchiudere il suo nome all'interno dei simboli "<", ">". È possibile inserire anche più di un protocollo, come nell'esempio.

```
@interface ClasseUtente:NSObject <Protocollo1, Protocollo2, ProtocolloN>
```

I metodi da implementare devono essere obbligatoriamente definiti come `@required`. Per far in modo che tutto funzioni correttamente bisogna implementare l'header file che contiene i riferimenti ai file dei protocolli.

Quando un oggetto utilizza un protocollo è interessante sapere se la sua implementazione è corretta, cioè se implementi tutti i metodi obbligatori del protocollo. Per far ciò esiste un metodo chiamato `conformsToProtocol:`, per usarlo basterà mandare il seguente messaggio all'oggetto, utilizzando l'operatore `@protocol`:

```
BOOL risposta = [object conformsToProtocol: @protocol(NomeProtocollo)];
```

2.5 SELETTORI ED ECCEZIONI

Selettori

Una delle caratteristiche di Objective-C sono i selettori, cioè delle variabili che contengono riferimenti a firme di metodi. Questi riferimenti, composti dal nome del metodo e da eventuali parametri, non hanno un legame ad un particolare oggetto. Quest'ultimo aspetto risulta molto importante poiché i selettori vengono impiegati per invocare metodi su oggetti di classe anche differenti.

I selettori consentono di ottenere due funzionalità: la prima, decidere il metodo da invocare su un oggetto in fase d'esecuzione (senza l'uso esplicito dei complessi puntatori a funzione del linguaggio C); la seconda, rendere facile l'interazione con i framework Apple relativi all'interfaccia utente.

Il tipo di dato che può ospitare un selettore è chiamato `SEL` che rappresenta il nome univoco di un metodo al momento della compilazione. Le variabili di questo tipo vengono valorizzate da una direttiva `@selector` che si aspetta come parametro la firma del metodo. Ad esempio:

```
SEL setNomeCognome;
setNomeCognome = @selector(setNome:cognome:);
```

I seguenti metodi, definiti nel protocollo `NSObject` e che ritornano il tipo `id`, vengono utilizzati per eseguire un selettore:

```
performSelector;

performSelectorWithObject:

performSelectorWithObject:WithObject:
```

L'oggetto indicato come parametro `withObject` viene passato al selettore in fase d'invocazione. Come è facilmente intuibile, il primo metodo non si aspetta nessun parametro, il secondo un parametro, mentre il terzo consente di passarne fino a due.

Se per esempio avessimo un oggetto `studente` al quale vorremmo passare due parametri tramite selettore potremmo scrivere quanto segue:

```
[studente performSelector: setNomeCognome
              withObject: @"Ilario" withObject: @"Genuardi"];
```

Purtroppo una limitazione dell'uso dei selettori è l'impossibilità di usarli per più di due parametri ma il problema può facilmente essere risolto passando, ad esempio, una struttura dati. Altro contesto in cui l'uso dei selettori è indispensabile è con i framework Apple, poiché questi sono utilizzati per la gestione degli eventi.

Eccezioni

Le eccezioni sono eventi che interrompono la normale esecuzione dell'applicazione. Queste vengono sollevate quando viene riscontrato un errore, cioè quando vengono compiute operazioni "illecite" in un determinato momento con un particolare dato, ad esempio operazioni di divisione per zero, chiamate ad un metodo non implementato, etc.

In Objective-C ci sono 4 direttive:

- `@try`: racchiude il codice che può lanciare un errore;
- `@catch`: gestisce un particolare errore avvenuto nel blocco `@try`;
- `@throw`: permette di lanciare un eccezione, cioè avverte il runtime system che è avvenuto un errore;
- `@finally`: contiene codice che deve essere sempre lanciato, sia che sia stato lanciato un errore, sia che questo non avvenga.

Il blocco `@catch` viene inserito sempre dopo un blocco `@try`. È possibile avere anche più direttive `@catch` in modo da catturare più tipologie di errore. Ogni eccezione lanciata è un oggetto che descrive tale eccezione ed in genere vengono utilizzate le classi `NSException` e `NSError`. Di fatto, tutte le eccezioni estendono `NSException`, ma esistono dei casi particolari in cui un'eccezione può essere rappresentata da un oggetto qualunque. Questo accade quando viene lanciata un'eccezione personalizzata (tramite `@throw`). Altri linguaggi di programmazione, come Java, hanno una gestione delle eccezioni molto somigliante all'Objective-C.

Quando si verifica un'eccezione viene analizzato ogni blocco `@catch` e viene scelto quello che gestisce l'eccezione specifica, se questa non fosse presente nella lista dei `@catch` verrà obbligatoriamente preso in considerazione l'ultimo blocco. Quindi se si utilizzano più blocchi `@catch` si deve rispettare un ordine ben preciso; le eccezioni devono andare dalla più specifica alla più generica, altrimenti non si riuscirebbe mai a catturare quella corretta.

Dopo che l'eccezione è stata gestita nel migliore dei modi si entra sempre nel blocco `@finally`, in cui solitamente sono inserite le operazioni di chiusura.

2.6 THREADING

Nel paragrafo 1.4.2 si è parlato dei thread a livello di sistema operativo, in questo paragrafo se ne parlerà dal punto di vista della programmazione. La creazione dei thread in ObjC è relativamente semplice. Per far ciò, è necessario disporre di una funzione o di un metodo che agisce da punto d'accesso del thread, dopodiché si utilizzerà una delle routine disponibili per avviarlo. Le tecniche seguenti mostrano il processo di creazione per le tecnologie di threading più comuni. I thread creati usando queste tecniche ereditano una serie di attributi di default, determinati dalla tecnologia che si utilizza.

Un primo metodo per generare i thread è quello di usare la classe `NSThread` che prevede due modi di utilizzo:

- Il primo è quello di utilizzare il metodo di classe `detachNewThreadSelector:toTarget:withObject:`
- Il secondo è quello di creare un nuovo oggetto `NSThread` e successivamente chiamare il metodo `start` (questo metodo è supportato solo in iOS e in MacOS X 10.5 e seguenti).

Entrambe queste tecniche creano thread distaccati dall'applicazione, ciò significa che le risorse associate ad ogni thread verranno recuperate automaticamente dal sistema quando ogni thread termina.

Dato che il metodo `detachNewThreadSelector:toTarget:withObject:` è quello supportato da tutte le versioni di OS X, si trova spesso nella maggior parte delle applicazioni Cocoa che fanno uso di thread.

Per inizializzare un nuovo thread, è sufficiente fornire:

- Il nome del metodo (specificato come selettore) che si desidera utilizzare come punto d'accesso del thread;
- L'oggetto che definisce tale metodo;
- I dati che si desiderano passare al thread.

L'esempio seguente mostra una chiamata di questo metodo.

```
[NSThread detachNewThreadSelector: @selector(myThreadMainMethod:)
                         toTarget: self withObject: nil];
```

Da MacOS X 10.5, è stato aggiunto il supporto (presente anche su iOS) per la creazione di oggetti `NSThread` senza l'immediata generazione del corrispondente thread. Ciò ha consentito di ottenere ed impostare vari attributi del thread prima che questi venga inizializzato.

Per inizializzare un oggetto `NSThread` si utilizza il metodo `initWithTarget:selector:object:` che ottiene le stesse informazioni del metodo `detachNewThreadSelector:toTarget:withObject:` e le utilizza per generare una nuova istanza di `NSThread`. Un thread inizializzato non è stato ancora avviato, per farlo bisogna chiamare il metodo `start`, come illustrato nell'esempio:

```
NSThread *myThread = [[NSThread alloc] initWithTarget: self
                                              selector: @selector
                                                  (myThreadMainMethod:)
                                                object: nil];
[myThread start];  // Qui viene avviato il thread
```

Se il thread di un oggetto `NSThread` è attualmente in esecuzione, si può utilizzare il metodo `performSelector:onThread:withObject:waitUntilDone:` per inviargli dei messaggi, questo è un modo conveniente che permette la comunicazione fra i thread. I messaggi inviati con questa tecnica vengono eseguiti direttamente dall'altro thread come parte della sua normale elaborazione di run loop.

Altro metodo per generare i thread è quello di sfruttare il legame tra i linguaggi C ed Objective-C, usando la POSIX Thread API. Questa tecnologia può essere utilizzata praticamente da qualsiasi tipo di applicazione (comprese Cocoa e applicazioni Cocoa Touch) e risulta più conveniente in caso di scrittura di software multipiattaforma. La routine di POSIX utilizzata per creare nuovi thread si chiama `pthread_create`.

Il codice seguente mostra due funzioni personalizzate per la creazione di un thread utilizzando le chiamate POSIX. La funzione `LaunchThread` crea un nuovo thread la cui routine principale è implementata nella funzione `PosixThreadMainRoutine`. Poiché POSIX crea thread "joinable" per default, questo esempio modifica gli attributi del thread per crearne uno indipendente.

```c
#include <assert.h>
#include <pthread.h>

void *PosixThreadMainRoutine(void *data) {
    // Fai qualcosa qui.
    return NULL;
}

void LaunchThread() {
    // Crea il thread usando le routine POSIX.
    pthread_attr_t attr;
    pthread_t posixThreadID;

    int returnVal;

    returnVal = pthread_attr_init(&attr);
    assert(!returnVal);

    returnVal = pthread_attr_setdetachstate(&attr,
                                            PTHREAD_CREATE_DETACHED);
    assert(!returnVal);

    int threadError = pthread_create(&posixThreadID, &attr,
                                     &PosixThreadMainRoutine, NULL);

    returnVal = pthread_attr_destroy(&attr);
    assert(!returnVal);

    if (threadError != 0) {
        // Riporta un errore.
    }
}
```

La funzione `LaunchThread` del codice precedente, avrebbe creato un nuovo thread indipendente. Naturalmente, i nuovi thread creati utilizzando questo codice non farebbero nulla perché uscirebbero quasi subito dopo il loro avvio. Per rendere le cose più interessanti, è necessario aggiungere del codice nella funzione `PosixThreadMainRoutine`.

Le applicazioni basate su C, possono far comunicare i thread attraverso l'uso di porte, condizioni o memoria condivisa. Tra il thread principale dell'applicazione e gli altri thread, si dovrebbe quasi sempre costituire una sorta di comunicazione, per far in modo che il thread principale possa controllarne lo stato e chiuderli in maniera pulita al termine dell'applicazione.

2.7 DIFFERENZE TRA OBJECTIVE-C E C++

L'Objective-C e il C++ sono due linguaggi nati nel 1983 come linguaggi di programmazione orientata ad oggetti, successori del C. Il primo, inizialmente rimasto parecchio ai margini, data la mancanza di supporto per piattaforme diverse da casa Apple, è divenuto popolare grazie al recente e crescente sviluppo di applicazioni per MacOS X e dispositivi iOS. Il secondo, invece, ha avuto maggior successo grazie alla sua potenza e versatilità.

Il progetto e l'implementazione dell'Objective-C e del C++ rappresentano due diversi approcci all'estensione del C.

L'Objective-C, a differenza del C++, aggiunge al C solo delle caratteristiche object-oriented e nella sua versione "base" (senza framework) non offre lo stesso in termini di librerie standard. Per ovviare questa lacuna, viene però corredato da una libreria, basata sul modello di OpenStep, Cocoa o GNUstep, la quale fornisce funzionalità simili a quelle offerte dalla libreria standard di C++.

Oltre alla programmazione strutturata del C, il C++, dal canto suo, supporta direttamente la programmazione ad oggetti, la programmazione generica e la meta-programmazione. Inoltre, ha una estesa libreria standard che include numerose classi container.

Un'altra notevole differenza consiste nel fatto che l'Objective-C fornisce un maggior supporto runtime alla riflessione rispetto a C++. In Objective-C si può interrogare un oggetto riguardo alle sue stesse proprietà, ad esempio se possa o meno rispondere ad un dato messaggio, mentre in C++ ciò sarebbe impossibile, a meno di ricorrere a librerie esterne. Comunque è possibile chiedere se due oggetti sono o meno dello stesso tipo (inclusi i tipi predefiniti) e se un oggetto è istanza di una data classe (o superclasse).

L'uso della riflessione fa parte di una più ampia distinzione tra caratteristiche dinamiche (run-time) e statiche (compile-time) dei linguaggi. Sebbene sia Objective-C che C++ implementino un misto di entrambe le caratteristiche, l'Objective-C è decisamente più orientato verso le decisioni dinamiche, mentre C++ verso quelle effettuate al momento della compilazione.

CAPITOLO 3
LA PIATTAFORMA PRESTASHOP: SERVIZI E FUNZIONALITÀ

Con il termine commercio elettronico, in inglese e-commerce, si intende quell'insieme di transazioni commerciali fra produttore e consumatore, finalizzate allo scambio di informazioni direttamente correlate alla vendita di beni e servizi tramite computer o dispositivi mobili, e reti telematiche. La storia del commercio elettronico risale agli anni '70, quando attraverso l'utilizzo di reti private, le aziende potevano scambiarsi informazioni di tipo commerciale, cosicché fornitori ed acquirenti potessero comunicare tra loro aggiornandosi costantemente. Ciò imponeva il limite di usare formati di comunicazione e documenti standard solo su reti private, creando una sorta di circolo chiuso e negando l'accesso a chiunque non ne facesse parte. Grazie allo sviluppo di Internet si è passati ad un vero e proprio mercato senza confini geografici, infatti, i siti web si trasformano in veri e propri negozi virtuali in cui acquistare ogni genere di prodotto.

Il produttore ha la possibilità di diffondere il proprio messaggio promozionale elettronicamente, modificandolo a piacere e aggiornandolo in tempo reale con pochissimi costi aggiuntivi. Internet, inoltre, offre l'opportunità al produttore di interagire direttamente con il consumatore e permette alle aziende di essere più competitive nel mercato. Sul web la fase preliminare di ordine e l'eventuale pagamento vengono effettuati "online" ma il bene acquistato viene poi spedito al domicilio o alla sede dell'acquirente mediante corrieri o altra forma postale.

Inizialmente per creare un proprio negozio virtuale era necessario ricorrere ad uno sviluppatore di siti web che creasse appositamente le pagine relative alla vendita dei prodotti ed a tutti i servizi correlati, ma ultimamente grazie al crescente sviluppo delle piattaforme di e-Commerce, "chiunque" sappia usare un pc può crearlo in pochi minuti, anche non conoscendo un minimo di programmazione, ciò grazie agli strumenti ed ai template messi a disposizione. Tra le più importanti piattaforme di commercio elettronico si possono citare: Magento, Joomla con VirtueMart, WordPress con WP-ecommerce e PrestaShop. Proprio su quest'ultima, si basa lo sviluppo del Framework e dell'App, argomenti di questo elaborato.

3.1 LA PIATTAFORMA PRESTASHOP

PrestaShop è un software Content Management System (CMS) nato nel 2007, gratuito ed open source realizzato completamente in PHP e funzionante su database MySQL. Esso è dedicato a tutti coloro vogliano creare un proprio negozio online. Ciò che lo differenzia principalmente dagli altri prodotti della categoria, come ad esempio Joomla! o WordPress, è il fatto che nasce per essere utilizzato soltanto nella realizzazione di siti di commercio elettronico. Questa sua particolarità lo ha portato nel tempo ad acquistare una notevole notorietà, contando circa 2000000 di download e vari riconoscimenti.

PrestaShop 1.5 (la versione utilizzata dal framework) contiene oltre 310 funzioni ed è in continuo sviluppo. Tutte le caratteristiche sono completamente gratuite e possono essere installate e disinstallate con un solo clic.

Ad oggi è uno dei CMS per l'e-Commerce maggiormente apprezzato, sia dagli sviluppatori che dagli utenti finali, per la semplicità d'uso e l'estrema intuitività. I temi PrestaShop sono realizzati con il motore di template Smarty, il quale permette una netta separazione tra contenuti, grafica e programmazione. La documentazione ufficiale, difatti, è divisa in due sezioni ben distinte: una per i web designer ed una per gli sviluppatori. Questa impostazione rende PrestaShop un CMS assai malleabile, che può essere adattato senza problemi ai progetti più disparati.

PrestaShop è stato concepito per essere modulare, cioè costituito da moduli di terze parti ed è proprio per questo che è estremamente personalizzabile. Un modulo è un'estensione che permette a qualsiasi sviluppatore di aggiungere delle funzionalità, come la possibilità di selezionare più prodotti o comunicare con altri servizi di e-Commerce (guide d'acquisto, piattaforme di pagamento, logistica, ed altro...)

La compagnia sviluppatrice di questa piattaforma offre a corredo del software più di 100 moduli liberi, sul sito ufficiale. Inoltre, sono anche disponibili a pagamento più di 1600 componenti aggiuntivi sviluppati sia dalla stessa compagnia che da membri della comunità.

3.1.1 Architettura tecnica di PrestaShop

L'architettura della piattaforma PrestaShop è di tipo 3-tier, ovvero basata su 3 livelli di astrazione:

- Object/data: L'accesso al database è controllato attraverso i file della cartella "classes";
- Data control: I contenuti utente sono controllati dai file della cartella principale;
- Design: Tutti i file del tema si trovano nella cartella "themes".

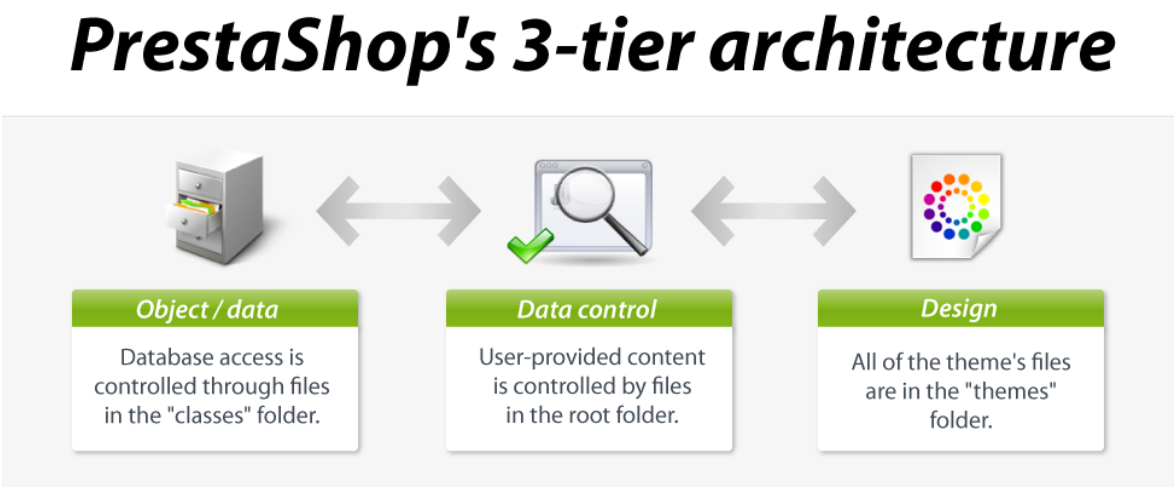

Figura 11 – Architettura della Piattaforma

Questo modello architetturale si basa sullo stesso principio del pattern Model-View-Controller (MVC), ma in maniera più semplice ed accessibile.

Il concetto cardine, che probabilmente viene ripetuto parecchie volte, è proprio la facilità di PrestaShop. Proprio per queste ragioni il team di sviluppo ha scelto di non utilizzare un framework PHP (come ad esempio Zend Framework, Symfony o CakePHP), ciò per consentire una migliore leggibilità del codice con conseguente velocizzazione dell'editing.

Tutto questo permette anche di ottenere prestazioni migliori, poiché il software è costituito dalle sole linee di codice e non contiene librerie generiche supplementari.

L'architettura a 3 livelli offre molti vantaggi:

- Il codice della piattaforma è più semplice da leggere;
- Gli sviluppatori possono aggiungere e modificare il codice in maniera più rapida;
- I Graphic designer e gli HTML integrator possono lavorare con i file della cartella "themes" senza dover conoscere o leggere una sola riga di codice PHP;
- Gli sviluppatori possono lavorare sui dati addizionali e sui moduli usati dagli HTML integrator.

Model:

Una **model** rappresenta il comportamento dell'applicazione e fornisce i metodi per accedere ai dati. Si occupa ad esempio dell'elaborazione di quest'ultimi, o dell'interazione con il database, gestendoli in maniera opportuna e garantendone l'integrità.

View:

Una **view** è l'interfaccia con cui l'utente interagisce. Il ruolo principale è quello di visualizzare i dati forniti dalla model, ma ha anche il compito di gestire tutte le azioni da parte dell'utente (la selezione degli elementi, la pressione sui pulsanti, lo scrolling, etc.), ed inviare questi

eventi al controller. La vista non fa alcuna trasformazione, essa visualizza solo il risultato dell'elaborazione eseguita dalla model.

Controller:

Il **controller** gestisce gli eventi di sincronizzazione tra la model e la view. Riceve tutti gli eventi generati dall'utente e "attiva" le azioni da eseguire. Se un'azione richiede ad esempio la modifica dei dati, il controller "chiederà" alla model di effettuare l'operazione, la model infine notificherà alla view che i dati sono stati modificati.

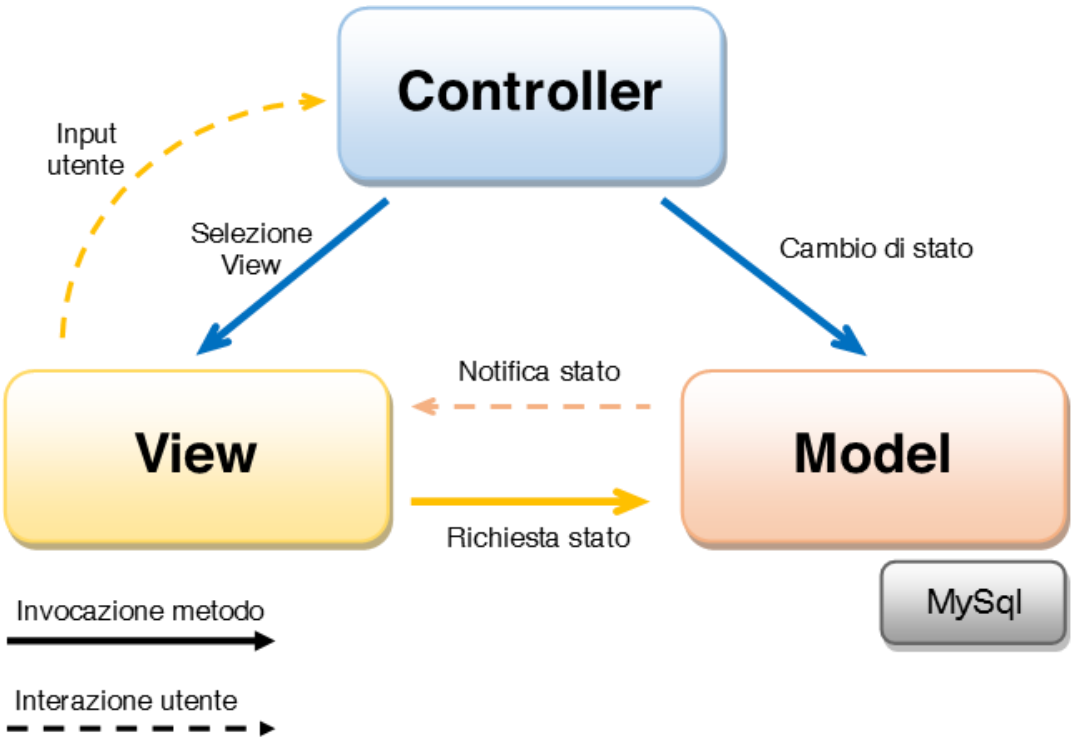

Figura 12 – Schema di interazione Model-View-Controller (Lato Server)

3.1.2 Funzionalità

Lo scopo principale di ogni applicazione di commercio elettronico è quello di permettere, agli utenti che la utilizzano, l'acquisto di beni o servizi, senza recarsi in un negozio fisico, ma semplicemente usando un computer o un qualsiasi altro dispositivo in grado di connettersi ad internet. Il tutto deve avvenire in modo semplice, veloce e sicuro. Ovviamente qualunque tipo di vendita comporta una serie di funzioni che permettano di completare una transazione, come ad esempio la scelta del metodo di spedizione e la scelta del tipo di pagamento. Quanto detto è valido per gli acquirenti, ma un'applicazione di e-Commerce deve tenere in considerazione anche chi vende, garantendo delle funzionalità per una corretta gestione del negozio virtuale.

Partendo proprio da questi punti possiamo analizzare nel dettaglio le funzioni che la piattaforma PrestaShop offre:

Gestione del catalogo:

Permette di gestire un elenco di prodotti dinamico grazie ad un back-office completo. Indipendentemente dal numero di prodotti venduti dal negozio, questa interfaccia di amministrazione consente di organizzare facilmente gli inventari più complessi con un solo clic. Ciò comporta una rapida importazione ed esportazione dei prodotti, impostazione degli attributi, ordinamento dei prodotti, sconti per quantità, e molto altro.

Caratteristiche:

- Gestione delle scorte;
- Possibilità di esportare un prodotto in eBay o in un altro store;
- Scelta del numero di prodotti visualizzabili per pagina;
- Vendita incrociata di prodotti;
- Vendita di prodotti scaricabili (software, videogiochi, etc.);
- Numero illimitato di prodotti, categorie ed attributi;
- Confronto tra prodotti;
- Potente Layered Navigation;
- Gestione avanzata del magazzino.

Visualizzazione dei prodotti:

Permette di mostrare i prodotti in maniera unica e fornisce ai clienti numerose opzioni per la visualizzazione di quelli desiderati. I commercianti, grazie a questa funzione possono invogliare i clienti all'acquisto, mettendoli a proprio agio con zoom, caratteristiche e viste multiple del prodotto.

Caratteristiche:

- Visualizzare le quantità disponibili;
- Possibilità di associare più immagini per ogni prodotto;
- Zoom sui prodotti;
- Prodotti correlati;
- Aggiungere il prodotto ad una lista;
- Recensioni sul prodotto;
- Funzione per inviare un prodotto ad un amico.

Gestione del sito:

I venditori possono godere di un sofisticato editor dei contenuti, la gestione degli “screen” dei prodotti e la modifica delle lingue per tutto il back-office. PrestaShop permette inoltre di aggiornare con un solo click la versione del software, al fine di rendere il negozio compatibile con quella più recente.

Caratteristiche:

- Permessi di amministrazione e utenti;
- Design completamente personalizzabile grazie all'utilizzo dei modelli;
- Personalizzazione del logo sulla fattura, dalla favicon e dell'immagine d'intestazione;
- Scelta della lingua di visualizzazione;
- Modifica del contenuto attraverso il CMS;
- Gestione delle prestazioni (velocità, cookie memorizzati nella cache);
- Aggiornamento one-click;
- Integrazione Webservice - CRM, ERP, etc;
- Personalizzazione del negozio senza modificare il core;
- Gestione di multi-store.

Ottimizzazioni per i motori di ricerca:

Una delle migliori forme di marketing è Search Engine Marketing che permette ai venditori di rendere il proprio negozio visibile a tutti coloro fossero interessati ai contenuti offerti dallo stesso. PrestaShop offre funzioni di ottimizzazione del sito e la garanzia di indicizzazione nei maggiori motori di ricerca. Tutto ciò è possibile grazie all'utilizzo delle parole chiave e dei tag. Grazie a questa funzione, un negozio può essere visto, ad esempio, sulla prima pagina di Google in maniera semplice ed efficace.

Caratteristiche:

- Facile interazione con i motori di ricerca;
- Personalizzazione dell'URL;
- Google Site Map;
- Meta-informazioni per prodotti e categorie;
- Mappa del sito generata in maniera automatica;
- URL dedicato per ogni prodotto per prevenire contenuti duplicati;
- Tag per ogni prodotto;
- Notifiche e-mail sullo stato di consegna.

Acquisti:

Tra le caratteristiche offerte, c'è anche quella di avere il checkout tutto in una pagina in cui è possibile personalizzare i campi per raccogliere tutte le informazioni necessarie alla vendita. Dalla scelta dei prodotti alla spedizione, il processo di checkout di PrestaShop facilita l'acquisto ai clienti.

Caratteristiche:

- Checkout tutto in una pagina;
- Possibilità per i clienti di creare un account o di continuare come ospite;
- Possibilità di confezionare regali e inserire dei messaggi di augurio;
- Carrelli salvati con tempo di scadenza.

Spedizione:

PrestaShop permette di avere moduli flessibili per il trasporto, completamente integrati con i principali vettori di spedizione. Fornisce ai clienti opzioni per spedizioni affidabili e la personalizzazione dei messaggi d'avviso.

Caratteristiche:

- Sconti sulla spedizione;
- Applicazione di costi aggiuntivi (tasse) sul prezzo o sul peso;
- Fatturazione separata e indirizzi di spedizione;
- Integrazione con USPS, FedEx, UPS, Canada Post;
- Aggiunta di destinazioni e vettori illimitati.

Pagamenti:

Sono integrate numerose opzioni di pagamento che possono essere installate con un solo clic. I venditori possono scegliere quali mettere a disposizione.

Caratteristiche:

- Pagamenti tramite carta di credito, assegno, bonifico bancario, etc;
- Tasse configurate automaticamente in base a stato e provincie;
- Conversione automatica delle valute;
- Integrazione con Authorize.Net (provider che consente ai commercianti di accettare pagamenti con carte di credito o assegni elettronici attraverso la loro pagina web);
- Differenziazione dei prezzi in base alle zone di spedizione;
- Creare o modificare un ordine attraverso il back-office.

Commercializzazione:

PrestaShop offre una varietà di strumenti di marketing e promozionali. Questi strumenti vengono messi a disposizione perché la crescita di PrestaShop è strettamente correlata alla crescita dei suoi commercianti. Più commercianti di successo usano PrestaShop, più la piattaforma migliora in termini di servizi, funzionalità ed importanza.

Caratteristiche:

- Automatizzazione dell'invio di e-mail promozionali;
- Sottoscrizione a newsletter;
- Programmi fedeltà;
- Possibilità di inviare le pagine con i prodotti rilevanti agli amici;
- Prodotti visti di recente;
- Buoni promozionali.

Account cliente:

La soddisfazione dei cliente è la chiave per mantenerli fedeli e far aumentare le vendite nell'arco di tutto l'anno. Fornire ai clienti la possibilità di effettuare tutte le operazioni in maniera semplice, come ad esempio il checkout one-click, dal proprio account personale rende la loro esperienza più piacevole e su misura alle loro esigenze.

Caratteristiche:

- Account personale completo di tutto;
- Possibilità di inviare messaggi tramite il proprio account;
- Gestione delle restituzioni;
- Possibilità di rivendere gli oggetti acquistati presso altri store;
- Servizio clienti post vendita.

Traduzioni:

Ogni singolo negozio grazie alla comunità di Prestashop, che ha membri in oltre 150 paesi, può essere tradotto in molte lingue diverse. La piattaforma mette a disposizione oltre 40 traduzioni disponibili e servizi di geolocalizzazione. I clienti possono, quindi, scegliere le lingue desiderate, importare ed esportare i pacchetti delle stesse e usufruire di strumenti di traduzione on-line.

Sicurezza:

PrestaShop mette a disposizione tutto ciò che un commerciante ha bisogno per essere protetto. Implementa delle funzionalità che permettano di effettuare tutte le operazioni in maniera sicura. Una connessione sicura, infatti, è fondamentale per effettuare pagamenti online senza il rischio di incorrere in spiacevoli sorprese.

Caratteristiche:

- Conformità PCI (Payment Card Industry, determina l'integrità della rete e transazioni end-to-end sicure, supportando operazioni di retail efficienti);
- Certificazione SSL;
- Back-office sicuro;
- Password e cookie crittografati;
- Analisi degli ordini FIA-NET (Società francese che propone servizi rivolti ai siti di e-Commerce e certifica quelli più affidabili).

Localizzazione/Tasse:

Il sistema di monitoraggio avanzato di PrestaShop è in grado di rilevare la posizione in cui si trova un cliente e di calcolare le relative tasse o promozioni previste dal commerciante. La configurazione dei tassi di cambio, inoltre, permette ai clienti di scegliere la loro valuta preferita indipendentemente da quella con cui viene venduto un prodotto.

Caratteristiche:

- Numero tasse illimitato;
- Formato indirizzo in base al paese;
- Numero illimitato di valute;
- Sincronizzazione dei tassi di cambio;
- Tasse differenziate per stato, provincia o paese;
- Fuso orario impostato per ubicazione;
- Formattazione valuta;
- Configurazione Eco Tax (incentivi).

Analisi e reporting:

Il reporting è la chiave per il monitoraggio e l'ottimizzazione delle prestazioni. I commercianti monitorizzano le vendite e le interazioni del visitatore per capire come migliorare la qualità dei servizi offerti.

Caratteristiche:

- Esportare elenchi di newsletter;
- Notifiche di aggiornamento nel back-office;
- Monitorare l'attività dei visitatori;
- Visualizzare i profili dei clienti;
- Reporting di ordini e vendite;
- Statistiche di affiliazione;
- Statistiche delle newsletter.

3.2 ARCHITETTURA DEL FRAMEWORK

Il progetto su cui si basa questa tesi prevede lo sviluppo di un framework che permetta di comunicare ed interagire con la piattaforma PrestaShop. Il framework si occuperà di effettuare le richieste al web server, mediante metodi REST, e presentare i dati ricevuti (file xml o immagini) all'app che lo utilizza. Lo schema d'interazione è quello illustrato in figura e, come si può facilmente notare, è caratterizzato da tre componenti:

- Applicazione;
- Web Server di PrestaShop;
- Framework.

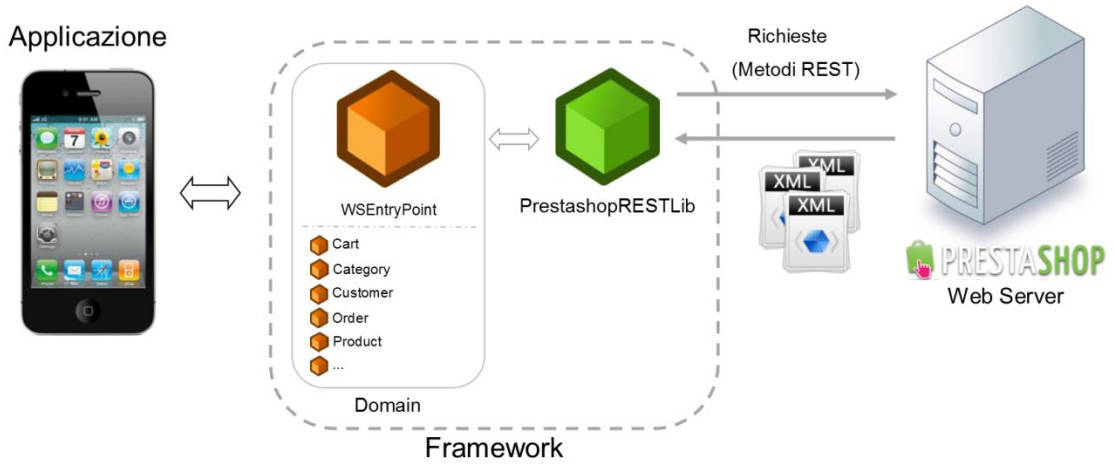

Figura 13 – Schema di interazione con il framework

Per capire meglio questo schema si può ricorrere ad un semplice scenario d'uso.

L'utente interagendo con l'app richiede, ad esempio, l'indice delle categorie dei prodotti venduti da uno store generico. A questo punto verrà invocato il metodo relativo al recupero delle categorie (`getCategories`) della classe `WSEntryPoint` che, a sua volta, chiamerà in causa la classe `PrestashopRESTLib`. Quest'ultima, richiederà al server i nomi delle categorie in questione, in questo caso mediante un `GET` HTTP (metodi REST). Se la richiesta è corretta, il web server ritornerà il file XML delle categorie, viceversa un errore. Una volta ricevuto il file, la classe `PrestashopRESTLib` lo passa a `WSEntryPoint` che creerà degli oggetti `Category` (classe dell'insieme `Domain` rappresentante le categorie) per ogni categoria contenuta nel file XML.

Tutti questi oggetti verranno ritornati all'applicazione sotto forma di un `NSMutableArray` di `Category`. Infine, l'applicazione si occuperà di visualizzare in forma tabellare l'elenco delle categorie precedentemente richieste dall'utente.

FRAMEWORK:

Il componente principale di questa architettura è il framework, esso è composto da una classe `PrestashopRESTLib` il cui compito è quello di comunicare con il web server, e da un insieme di classi, `Domain`, che hanno il compito di effettuare il mismatch dei dati ricevuti dalla libreria `PrestashopRESTLib`, rendendoli "comprensibili" all'applicazione.

Alla base del framework avremo le funzionalità di basso livello, come la gestione della comunicazione con la piattaforma, man mano che si sale, invece, le funzionalità di alto livello come pagamenti, caching e quant'altro.

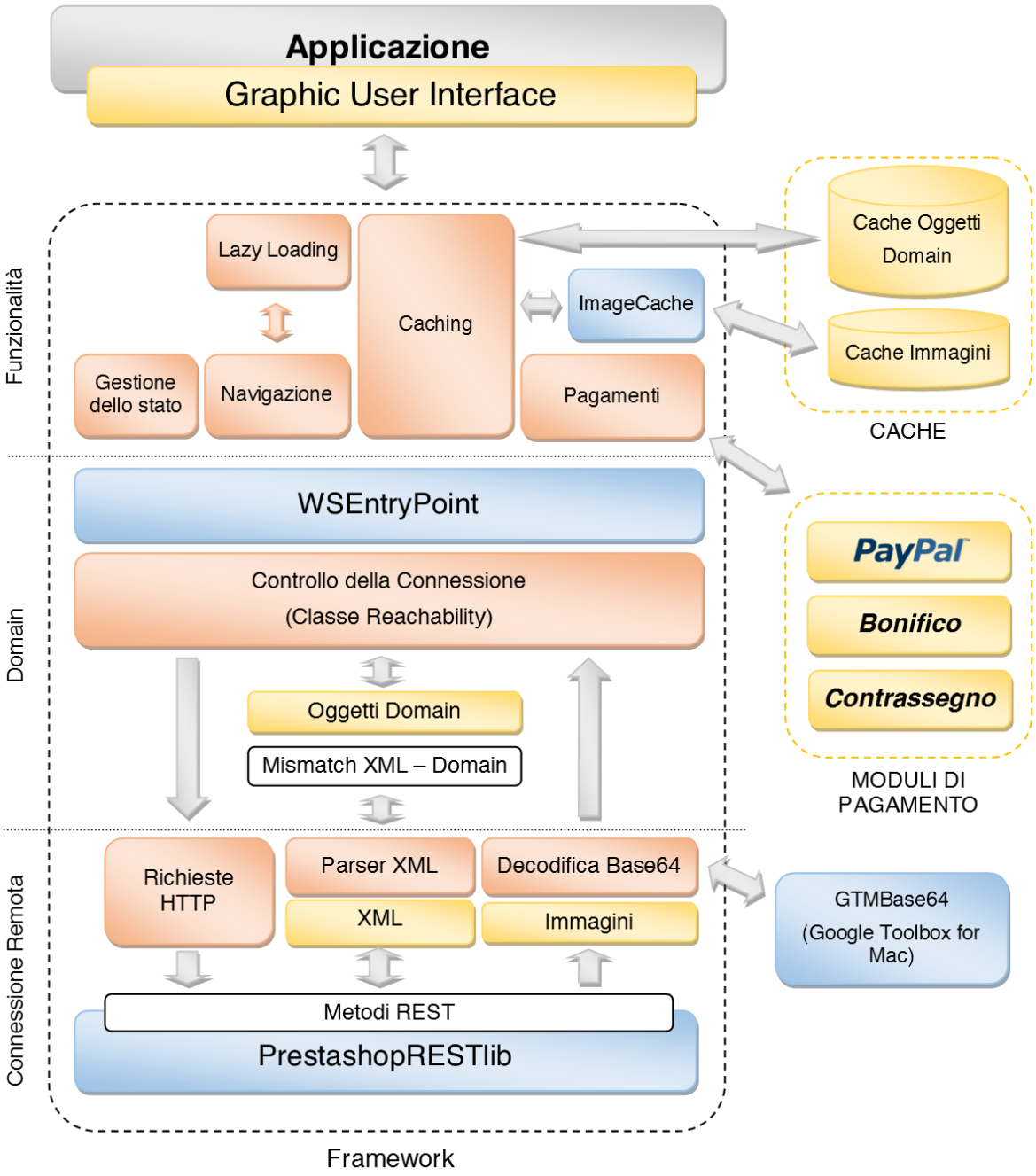

Figura 14 – Modello Architetturale

A livello architetturale, il framework è caratterizzato da tre livelli di astrazione: Connessione Remota, Domain e Funzionalità.

Alla base ci sarà il livello **Connessione Remota** che avrà tutte le funzionalità di basso livello che vengono utilizzate dai livelli sovrastanti e principalmente dalla classe `WSEntryPoint`. Tali funzionalità sono del tutto trasparenti agli occhi dello sviluppatore in quanto non verranno mai utilizzate direttamente dallo stesso. Questo livello ricopre un ruolo importante, senza le funzionalità dello stesso, infatti, non si potrebbe instaurare la connessione remota con la piattaforma.

Il livello **Domain** implementa tutte le tecnologie sfruttate dalle funzionalità di alto livello dell'applicazione. L'elemento predominante di questo livello è la classe `WSEntryPoint` che usufruisce delle caratteristiche fornite dal livello inferiore per svolgere i propri compiti. Quasi tutti i metodi della classe `WSEntryPoint`, infatti, svolgono compiti remoti, ciò nonostante essa non comunica direttamente con il web server, ma delega il livello "Connessione Remota" che si occupa di gestire le richieste e le risposte provenienti dal server.

Il livello che sta più in alto prende il nome di **Funzionalità**. È quello che caratterizza le astrazioni delle funzioni offerte dal framework, come la gestione dello stato per identificare un utente, Il caching per velocizzare l'app, la scelta del tipo di pagamento, etc.

Nonostante tutti i livelli facciano parte della medesima architettura, in questo capitolo si analizzerà soltanto quello relativo alla "Connessione Remota", gli altri due verranno analizzati nel capitolo successivo. Tale scelta è sensata per distinguere le funzioni già offerte dalla piattaforma con le estensioni che verranno proposte nel progetto. La classe `PrestashopRESTLib`, infatti, altro non è che un porting della libreria PHP `PSWebServiceLibrary` (fornita da Prestashop), nel linguaggio Objective-C. Il porting di questa libreria è necessario per rendere compatibili i servizi offerti da Prestashop con le API di un'applicazione nativa per iOS e quindi con i metodi implementati nella classe `WSEntryPoint`.

3.2.1 Il livello "Connessione Remota"

La classe `PrestashopRESTLib` è il componente che sta alla base dell'architettura e quello più importante del livello **Connessione Remota**. Il suo compito è fare da tramite tra il server remoto e il framework. Nella pratica, con abuso di linguaggio, è una sorta di "gateway", cioè l'elemento da cui passano tutte le richieste e le risposte remote. Per far ciò, come si può notare anche dal nome, utilizza i metodi REST (paragrafo 3.3.2), cioè metodi che svolgono operazioni di lettura e scrittura sul server utilizzando le specifiche del protocollo HTTP.

Con riferimento alla figura sottostante, al di sopra della classe ci sono le operazioni svolte dalla stessa, blocchi in rosso, mentre in giallo sono raffigurati gli oggetti con cui essa interagisce, immagini e file XML.

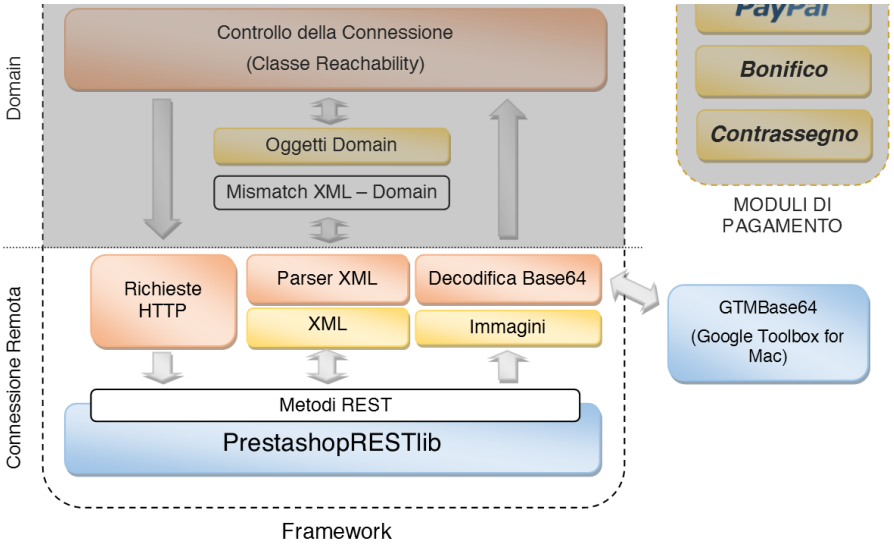

Figura 15 – Connessione Remota (Architettura)

Il blocco denominato “Richieste HTTP” è relativo alle operazioni di creazione dei messaggi HTTP rappresentanti le richieste dell’utente provenienti dai livelli più alti. La comunicazione tra le richieste HTTP e la classe `PrestashopRESTLib` è unidirezionale verso quest’ultima perché stiamo parlando di richieste verso il server remoto e non delle risposte ricevute dallo stesso. D’altro avviso, invece, le comunicazioni con XML ed immagini. Nel primo caso, cioè con i file XML, la comunicazione avviene in entrambi sensi perché, oltre a recuperare questi file, una richiesta può svolgere operazioni di scrittura sul server che richiedono proprio l’invio di un file XML. Un esempio è la creazione di un nuovo account dove s’invierà al web server l’XML contenente i dati dell’utente. Questo file verrà generato dalle classi dell’insieme `Domain`, a partire da un template dichiarato nella classe `WSEntryPoint`. Nel caso delle immagini, invece, la comunicazione è solo in entrata dato che il framework non implementa operazioni di upload.

Avendo a che fare sia con file XML che con immagini, la classe `PrestashopRESTLib` implementa anche operazioni ausiliarie, come il processo di validazione di un file XML (parsificazione) e la decodifica delle immagini in base64, quest’ultima possibile grazie all’uso della libreria di funzioni GTM (Google Toolbox for Mac), messa a disposizione da Google. Queste operazioni sono fondamentali per il corretto funzionamento di tutto il framework dato che i file XML devono essere formattati in maniera corretta e soprattutto integri nella loro struttura: in certi casi potrebbe verificarsi, infatti, che un file arrivi corrotto a causa di problemi di rete, ciò impedirebbe all’applicazione di funzionare correttamente. Analogo ragionamento nel senso opposto dove, se il file non sarà ben strutturato, sarà il server a non riconoscerlo, con conseguente annullamento dell’operazione ed invio del messaggio d’errore.

Riguardo le immagini, esse verranno ricevute codificate in base64, quindi per essere utilizzate e riconosciute bisogna decodificarle opportunamente. Le immagini decodificate verranno poi passate ai livelli superiori.

3.3 LA LIBRERIA PrestashopRESTLib

La classe `PrestashopRESTLib`, come già detto precedentemente, è quella che gestisce la comunicazione con il web server e riceve le risposte dallo stesso. In essa sono presenti i dati di accesso alle API del negozio virtuale, come l'url e la chiave di autenticazione, e tutti i metodi necessari ad effettuare le richieste, ricevere le risposte e altri metodi relativi alla gestione della connessione ed al debugging.

I vari metodi che compongono la classe sono raggruppati in sezioni: Init, Connection, Debug, XML, Add, Get, Head, Edit, Delete, Image e Utility.

3.3.1 Init, Connection, debug, xml parsing

INIT:

I metodi init sono quei metodi che permettono di inizializzare un'istanza di una classe. Nel capitolo 2 abbiamo visto che, in objective-c, una classe deriva, salvo eccezioni, dalla superclasse `NSObject` e quindi ne eredita tutti i metodi. Questi possono essere sovrascritti per adattarli alle proprie esigenze. In questo caso, i metodi `initWithUrl:key:` e `initWithUrl:key:debug:`, ereditano il metodo `init` della superclasse `NSObject` e permettono di istanziare la classe `PrestashopRESTLib` passandogli tre parametri: `url`, `key` e `debug`. Il parametro `url` rappresenta l'indirizzo logico del negozio, il parametro `key` la chiave di autenticazione e la variabile booleana debug se settata a `YES` permette di stampare a video il log delle operazioni. Il primo metodo non riceve il parametro debug in quanto lo setta a `YES` per default, il secondo, invece, dà la possibilità di scegliere se attivarlo o no.

CONNECTION:

La sezione Connection comprende i metodi che controllano lo stato della connessione ed effettuano le richieste remote: `checkStatusCode:`, `checkWebServiceVersion:`, `executeRequestWithUrl:params:`, `formatHeaderStringWithHeader:status:`.

- `checkStatusCode:` traduce il codice del messaggio HTTP ricevuto dal server. Il metodo riceve il parametro `statusCode` di tipo `NSString`, con un codice di tre cifre, e ritorna una stringa contenente il significato del codice di errore. L'unico caso in cui il metodo non ritorna nulla è quando si riceve il codice 200. Quest'ultimo caso ha senso in quanto la risposta del server ha avuto esito positivo e l'operazione è stata effettuata correttamente. Il metodo `checkStatusCode:` "traduce" solo i seguenti codici di stato: 200, 201, 204, 400, 401, 404, 405 e 500. Nel caso in cui venga ricevuto un codice differente verrà comunque ritornato un messaggio di tipo "`unexpected HTTP status`".

- `checkWebServiceVersion:` controlla se la versione di Prestashop è compresa tra la 1.5.0.0 e la 1.5.9.9. Questo controllo è necessario per garantire il corretto funzionamento del framework, infatti, in caso di versione differente potrebbero verificarsi problemi di incongruenza con conseguenti crash dell'applicazione che lo utilizza.

- `executeRequestWithUrl:params:` è il metodo che si occupa sia del setup della connessione che dell'effettuare la richiesta al server. Essendo forse il metodo più importante (tutte le richieste passano da lui), merita un'analisi approfondita sul suo funzionamento. Il metodo riceve un parametro di tipo `NSDictionary` che conterrà delle coppie chiave-valore, utili a settare alcuni parametri relativi alla connessione, ad esempio metodo HTTP, autenticazione, timeout, codifica, etc.
  In caso di mancanza di qualche parametro di connessione, sarà comunque lo stesso metodo ad occuparsi di settare quelli mancanti attraverso le sue variabili di default:

  - `HTTP_AUTHUSERPSWD`: impostato con la variabile di classe `_key`, contiene la stringa di autenticazione con il server;
  - `HTTP_METHOD`: impostato a `GET` (funzione usata più spesso);
  - `HTTP_TIMEOUT`: 10 secondi, tempo di validità della richiesta.

  La prima parte del metodo, quindi, si occuperà di instaurare la connessione.
  Il servizio usato per la transazione HTTP è il basic authentication che permette di accedere a risorse protette da autenticazione in maniera semplice, data la totale assenza di cookie o di identificativi di sessione.
  La richiesta al web server è di tipo: `http://usr:pwd@www.esempioindirizzoweb.com/path` con `usr` rappresentante l'username e `pwd` la password.
  Nel nostro caso il campo `pwd` è vuoto, verrà usato solo il parametro `usr` che conterrà la stringa di autenticazione codificata in base64 come richiesto dal servizio.
  In seguito, `executeRequestWithUrl:params:` selezionerà i parametri HTTP relativi al metodo che lo ha invocato. Se la richiesta viene effettuata da un metodo per creare una risorsa sul web server, il parametro `HTTP_METHOD` sarà impostato con `POST`, `GET` in caso di richiesta di una risorsa, e così via.
  Dopo questa fase preliminare verrà effettuata la richiesta attraverso il metodo `NSURLConnection`.
  La risposta conterrà il messaggio HTTP contenente i dati e l'esito della richiesta; questi verranno ritornati dal metodo `executeRequestWithUrl:params:` sotto forma di un `NSDictionary` (variabile `result`), contenente tre voci:

  - `HTTP_RESPONSE_CODE` = Codice di stato;
  - `HTTP_RESPONSE_BODY` = Body della risposta;
  - `HTTP_RESPONSE_HEADER` = Header del messaggio HTTP.

  Il body della risposta può contenere dati eterogenei (immagini, file XML, etc.) in base al tipo di richiesta effettuata, per cui è opportuno gestirli in maniera corretta. Il tipo di dato ricevuto è ricavabile dal contenuto del parametro `content` del protocollo HTTP (`text/xml` per file testuali, `image` per immagini). L'ultima parte del metodo

`executeRequestWithUrl:params:` si occuperà proprio di trattare questi dati. Se il dato ricevuto è un file XML l'intero body verrà copiato nella variabile `HTTP_RESPONSE_BODY`, la cosa cambia però se si tratta di un'immagine, infatti quest'ultima essendo un insieme di dati binari e non un file di testo, come nel caso precedente, ha la necessità di essere codificata in base64 e standardizzata nel formato UTF-8 per essere utilizzata correttamente dagli altri metodi del framework.

- `formatHeaderStringWithHeader:status:` l'header del messaggio ricevuto dal web server è formattato secondo lo standard HTTP, questo metodo restituisce una stringa più leggibile per la fase di debug, effettuando semplicemente delle sostituzioni di caratteri al testo originario. Un esempio del risultato di questo metodo è il seguente:

```
{
    "Access-Time" = 1377797844;
    Connection = "Keep-Alive";
    "Content-Encoding" = gzip;
    "Content-Length" = 7324;
    "Content-Sha1" = 09f195b9d683ca94939a6a3609c4eb9d82d8135a;
    "Content-Type" = "text/xml;charset=utf-8";
    Date = "Thu, 29 Aug 2013 17:37:24 GMT";
    "Execution-Time" = "0.038";
    "Keep-Alive" = "timeout=15, max=100";
    "PSWS-Version" = "1.5.2.0";
    Server = "Apache/2.2.16 (Debian)";
    Vary = "Host,Accept-Encoding";
    "X-Powered-By" = "PrestaShop Webservice";
}
```

Figura 16 – Header messaggio HTTP ricevuto dal server

```
HTTP STATUS: 200 [nessun errore]
Access-Time: 1377797844
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 7324
Content-Sha1: 09f195b9d683ca94939a6a3609c4eb9d82d8135a
Content-Type: text/xml charset=utf-8
Date: Thu, 29 Aug 2013 17:37:24 GMT
Execution-Time: 0.038
Keep-Alive: timeout=15, max=100
PSWS-Version: 1.5.2.0
Server: Apache/2.2.16 (Debian)
Vary: Host,Accept-Encoding
X-Powered-By: PrestaShop Webservice
```

Figura 17 – Risultato del metodo: `formatHeaderStringWithHeader:status:`

Dai due log si può notare l'eliminazione delle parentesi graffe, delle virgolette e dei punto e virgola finali, l'inserimento dello stato della risposta HTTP con relativo codice d'errore e significato tra parentesi quadre, e sostituito il carattere "=" con il carattere ":".

DEBUG:

La sezione debug offre dei metodi che permettono di stampare a video gli esiti delle operazioni che vengono svolte a run-time. Esse sono utili agli sviluppatori per risolvere problematiche legate ad errori di programmazione o conoscere lo stato delle operazioni svolte dall'applicazione. I metodi in questione sono due: `appendDebug:` e `appendDebug:content:`. La loro funzione è semplicemente quella di aggiungere alla variabile d'istanza `_debugLog` della classe `PrestashopRESTLib`, una stringa contenente il log delle operazioni. Questa stringa rappresenta il "registro" di tutte le operazioni svolte dall'app.

XML:

Il metodo `parseXMLWithRequest:`, l'unico di questa sezione, è quello che ha il compito di gestire il file XML ottenuto dal web server. La prima parte del codice si occupa di analizzare lo stato relativo alla richiesta. Se la risposta ha un codice HTTP 200, vuol dire che non si sono verificati errori e che quindi il file XML, contenente la risorsa richiesta, è stato ricevuto correttamente; in caso di codice diverso, ad esempio HTTP 404 (risorsa mancante), siamo in presenza di un'eccezione che ovviamente dovrà essere gestita e visualizzata nel log. Una delle funzioni della seconda parte del codice è proprio quella di gestire queste eccezioni, ma quella più importante è comunque quella di parsificare il file ricevuto (controllo di possibili errori di forma). Infine, il metodo `parseXMLWithRequest:` ritornerà una variabile di tipo `DDXMLElement` che, se la parsificazione non ha rilevato errori, conterrà la root del file XML, altrimenti un valore `nil`. Nella scelta del parser per questo framework, si è tenuta in considerazione la possibilità di averne uno di tipo DOM, facile da usare, che potesse, oltre che leggere gli elementi e gli attributi del file XML, anche modificarne i contenuti, ragion per cui la scelta è ricaduta sulla libreria open-source `KissXML`, basata su `libXML2`, considerata una delle migliori.

3.3.2 Metodi REST

REST, acronimo di REpresentational Transfer State, è un paradigma per la realizzazione di applicazioni Web che permette la manipolazione delle risorse per mezzo dei metodi GET, POST, PUT e DELETE del protocollo HTTP. Per capire meglio di cosa stiamo parlando si può ricorrere ad un esempio pratico: un utente digita sulla barra degli indirizzi del suo browser l'indirizzo internet relativo alla risorsa a cui vuole accedere; questa risorsa è tipicamente un file HTML che risiede su un computer remoto; supponiamo che l'utente digiti

http://www.unibo.it e prema invio, il browser richiede la pagina descritta da quell'indirizzo tramite protocollo HTTP, la richiesta arriva fino al server dell'Università di Bologna che risponde fornendo il contenuto della pagina al browser. Ciò che accade è che il browser ha inviato la richiesta HTTP: "GET http://www.unibo.it".

GET è uno dei metodi previsti dal protocollo HTTP ed è utilizzato per il reperimento delle risorse web. In questo caso viene invocato interagendo con la barra degli indirizzi, ma nella pratica viene invocato anche ogni qualvolta l'utente clicca su un link ipertestuale.

Oltre a digitare gli indirizzi internet e cliccare sui link, è possibile interagire con siti internet e applicazioni Web in altre maniere, ad esempio quando un utente inserisce i propri dati all'interno di un form per registrarsi ad un nuovo servizio. A differenza della semplice navigazione, quando si compila un form e lo si invia (di solito per mezzo di un apposito pulsante sistemato in fondo al form stesso), l'operazione modifica lo stato del server, ad esempio aggiungendo il nostro nominativo alla lista delle persone iscritte ad un determinato servizio. Questa operazione d'inserimento di dati non utilizza il metodo GET, ma il metodo POST.

Utilizzando i metodi GET e POST e l'indirizzo di una determinata risorsa web si possono reperire le risorse da un sistema remoto: nel nostro esempio una pagina internet, dall'altra fornire una nuova risorsa, il nominativo introdotto nel form.

Il protocollo HTTP prevede inoltre la possibilità di operare sulle risorse con due altri metodi: PUT per modificare e DELETE per cancellare una risorsa remota. Questi due metodi non sono disponibili per mezzo dei comuni browser, ma fanno parte del protocollo HTTP alla pari dei metodi GET e POST.

I metodi REST spesso vengono confusi con le operazioni CRUD (CREATE, READ, UPDATE, DELETE) dei database per le loro analogie, ma sono notevolmente differenti a livello architetturale.

Quanto detto finora è utile per introdurre i metodi REST della classe `PrestashopRESTLib`, basata proprio su di essi. Ogni tipologia di metodo ne ha uno principale che esegue le operazioni, gli altri servono solamente ad impostare i parametri, richiamando i metodi principali attraverso la parola chiave `return`. In seguito, per evitare confusione, i metodi principali verranno evidenziati in grassetto.

ADD:

Sono quei metodi che permettono di aggiungere una risorsa sul web server ed infatti utilizzano il metodo POST del protocollo HTTP. Essi vengono invocati dall'applicazione quando si vuole creare un nuovo utente (`Customer`), un carrello della spesa (`Cart`) e un ordine (`Order`).

`addWithResource:postXml:` chiama il metodo **`addWithUrl:resource:postXml:`** utilizzando soltanto il nome della risorsa e il parametro `postXml` che conterrà il file XML (formato stringa) da caricare sul server.

addWithUrl:resource:postXml: Imposta i parametri di connessione ed effettua la richiesta tramite il metodo executeRequestWithUrl:params:. Se riceve un parametro url con valore nil, lo assegnerà con la stringa di default del tipo _url + "/api/" + il nome della risorsa.

Esempio: http://prestashop.molluscobalena.it/api/carts

Nel codice seguente si può notare che i parametri di configurazione HTTP_METHOD e HTTP_POSTFIELD, impostati relativamente a POST e xml=%@, vengono immagazzinati all'interno di un NSMutableDictionary che verrà passato al metodo che esegue la richiesta vera e propria. Il valore di HTTP_METHOD ovviamente sarà @"POST" perché si sta effettuando un'operazione di scrittura, quello di HTTP_POSTFIELD conterrà il file XML da inviare, preceduto dal prefisso @"xml=".

```
NSMutableDictionary *params = [NSMutableDictionary dictionary];
[params setValue: @"POST" forKey: @"HTTP_METHOD"];
[params setValue: [NSString stringWithFormat: @"xml=%@", postXml] forKey:
    @"HTTP_POSTFIELD"];

NSDictionary *request = [self executeRequestWithUrl: requestUrl params:
    params];
```

GET:

I metodi sottostanti definiscono le operazioni di richiesta, e quindi lettura, delle risorse. Ricordando quanto detto precedentemente, la richiesta è di tipo "GET <indirizzo della risorsa>", quindi, come è facilmente intuibile, questi metodi si occuperanno di creare la stringa url relativa all'indirizzo della risorsa.

I metodi getWithResource:, getWithResource:options:, getWithResource:idt: e getWithResource:idt:options: vengono utilizzati solo per chiamare il metodo **getWithUrl:resource:idt:options:** settando solo alcuni parametri.

Il metodo **getWithUrl:resource:idt:options:** è quello che si occupa della creazione dell'indirizzo url relativo all'oggetto remoto richiesto, di aggiungere all'url eventuali parametri e passare la stringa risultante al metodo executeRequestWithUrl:params: che effettuerà la richiesta. I parametri che possono essere aggiunti all'url sono: @"filter", @"display", @"sort" e @"limit".

@"filter" permette di filtrare i contenuti del file xml, al fine di restringere il campo di interesse, ad esempio tutti i prodotti con i prezzi compresi tra 20€ e 100€;

@"display" visualizza solo gli elementi del file xml richiesti. Nel caso in cui si richiedesse una risorsa di tipo Product, volendone visualizzare solo nome e prezzo, si dovrà settare questo parametro con il valore @"[name,price]" (la virgola separa gli elementi);

@"sort" ordina gli elementi, ad esempio in maniera alfabetica crescente;

`@"limit"` permette di impostare il limite massimo di elementi da visualizzare, utile in presenza di file con tanti record. Se si richiede una lista di prodotti che ne contiene 2000, grazie a questo parametro è possibile limitarne il numero.

Nelle figure sottostanti vengono mostrati degli esempi di risultato, in base a degli url con relativi parametri.

```
http://prestashop.molluscobalena.it/api/products
```

```
- <prestashop>
  - <products>
      <product id="1" xlink:href="http://prestashop.molluscobalena.it/api/products/1"/>
      <product id="2" xlink:href="http://prestashop.molluscobalena.it/api/products/2"/>
      <product id="3" xlink:href="http://prestashop.molluscobalena.it/api/products/3"/>
      <product id="4" xlink:href="http://prestashop.molluscobalena.it/api/products/4"/>
      <product id="5" xlink:href="http://prestashop.molluscobalena.it/api/products/5"/>
      <product id="6" xlink:href="http://prestashop.molluscobalena.it/api/products/6"/>
      <product id="7" xlink:href="http://prestashop.molluscobalena.it/api/products/7"/>
      <product id="8" xlink:href="http://prestashop.molluscobalena.it/api/products/8"/>
    </products>
  </prestashop>
```

```
http://prestashop.molluscobalena.it/api/products?filter[price]=[20,100]
```

```
- <prestashop>
  - <products>
      <product id="2" xlink:href="http://prestashop.molluscobalena.it/api/products/2"/>
      <product id="6" xlink:href="http://prestashop.molluscobalena.it/api/products/6"/>
    </products>
  </prestashop>
```

I prodotti con id 2 e 6 hanno il prezzo compreso tra 20 e 100 €

```
http://prestashop.molluscobalena.it/api/products?display=[id,name]&limit=2&sort=[name_ASC]
```

```
- <prestashop>
  - <products>
    - <product>
        <id>6</id>
      - <name>
        - <language id="6" xlink:href="http://prestashop.molluscobalena.it/api/languages/6">
            Custodia portafoglio in pelle Belkin per iPod nano - Nero/Cioccolato
          </language>
        </name>
      </product>
    - <product>
        <id>1</id>
      - <name>
          <language id="6" xlink:href="http://prestashop.molluscobalena.it/api/languages/6">iPod Nano</language>
        </name>
      </product>
    </products>
  </prestashop>
```

HEAD:

I metodi HEAD hanno un comportamento analogo ai metodi GET ma a differenza di quest'ultimi non richiedono la risorsa, ma solamente l'header del messaggio HTTP relativo alla risposta del server. Sebbene possano sembrare inutili, permettono di effettuare operazioni di controllo, come ad esempio se una risorsa esiste o no, ciò senza impegnare pesantemente la connessione. Quest'ultima affermazione diventa sensata quando si richiede una risorsa con molti dati, infatti non avrebbe senso scaricare l'intero file per leggere soltanto l'header del messaggio.

I metodi `headWithResource:`, `headWithResource:options:`, `headWithResource:idt:`, `headWithResource:idt:options:`, come nel caso dei metodi GET, richiamano il metodo **`headWithUrl:resource:idt:options:`** settando solo alcuni parametri mentre quest'ultimo effettuerà la richiesta. Una differenza a livello di settaggi la si può notare nel parametro `HTTP_METHOD` che questa volta avrà valore `@"HEAD"` al posto di `@"GET"`.

EDIT:

Spesso accade che si vogliano modificare i dati contenuti in una risorsa, ad esempio un indirizzo scritto in maniera errata o lo stato di un ordine. In questi casi, dato che la risorsa è già presente non ha senso crearne una nuova, ragion per cui è preferibile modificare soltanto i contenuti desiderati. I metodi che si occupano di effettuare queste modifiche prendono il nome di `editWithUrl:postXml:`, `editWithResource:idt:postXml:` ed **`editWithUrl:resource:idt:postXml:`**.

I primi due al solito settano i parametri per l'invocazione di quest'ultimo metodo. Una particolarità, che si nota dalle firme dei metodi, è il parametro postXml sempre presente. Questo parametro è fondamentale perché si tratta del file XML contenente la modifica che verrà effettuata sul web server.

I parametri della richiesta `HTTP_METHOD` e `HTTP_POSTFIELD` verranno settati rispettivamente a `@"PUT"`, per indicare che si tratta di una modifica dei dati, e con la stringa del contenuto XML, ricordando che `HTTP_POSTFIELD` è il campo utilizzato per inoltrare i dati.

DELETE:

Gli ultimi metodi REST, `deleteWithUrl:` e `deleteWithResource:idts:`, servono per cancellare una risorsa dal server. Sono metodi che permettono di settare parzialmente i parametri del metodo principale **`deleteWithUrl:resource:idts:`**. Il primo metodo invoca quest'ultimo passandogli semplicemente l'url della risorsa, il secondo passandogli nome della risorsa più un array di valori stringa che rappresentano uno o più id delle risorse. Diversamente da quanto accade negli altri metodi REST, con i DELETE è possibile cancellare anche più risorse contemporaneamente, per questo motivo il parametro `idts` è un array, la cui dimensione rappresenta il numero delle risorse da cancellare.

Se ad esempio si invocasse `deleteWithResource:idts:` con parametro `resource = @"products"` e parametro `idts` l'array di stringhe `@"12"`, `@"23"`, `@"34"`, verrebbero eliminati i prodotti con id 12, 23, 34.

Il metodo **`deleteWithUrl:resource:idts`** è quello che si occupa della richiesta di eliminazione tramite il solito `executeRequestWithUrl:params:`. Il funzionamento è molto semplice: se viene invocato tramite con il parametro `url` esegue la richiesta senza fare operazioni preliminari, dato che per url s'intende la stringa della risorsa completa. Se, invece, viene invocato con `resource` e array di id, questa volta si occuperà di formare la stringa relativa alla (o alle) risorsa da cancellare. In quest'ultima invocazione si possono distinguere due casi: l'eliminazione di una sola risorsa e l'eliminazione di due risorse.

Per eliminare una singola risorsa l'url risultante è del tipo:

`<indirizzo>/api/<risorsa>/<id>`

Nell'altro caso:

`<indirizzo>/api/<risorsa>/?id=[id1,id2,…,idn]`

Esempio:

Eliminazione dell'ordine con id 4:	`http://prestashop.molluscobalena.it/api/orders/4`

Eliminazione degli ordini con id 4, 5, 6:	`http://prestashop.molluscobalena.it/api/orders/?id=[4,5,6]`

Oltre all'url, il metodo `executeRequestWithUrl:params:` ha bisogno di un altro parametro per funzionare correttamente che è quello relativo ai settaggi della connessione HTTP (`params`). Trattandosi di una richiesta di eliminazione il parametro `HTTP_METHOD` verrà settato a `@"DELETE"`.

Per concludere, i metodi DELETE come risultato restituiscono una variabile booleana rappresentante l'esito dell'operazione richiesta. Essi ritorneranno `YES` in caso di esito positivo, `NO` viceversa.

3.3.3 Download immagini

Quando si scarica un prodotto da una categoria, oltre ad informazioni come prezzo e descrizione si scaricheranno anche una o più immagini associate che permettono di avere un'anteprima del prodotto. In un'applicazione di commercio elettronico, le immagini costituiscono un mezzo importante nella scelta di un acquisto da parte di un utente, essendo difatti l'unico modo per vedere un oggetto.

La libreria PrestashopRESTLib implementa, quindi, quei metodi atti a scaricare le immagini. Tutti i metodi questa sezione restituiscono un oggetto NSData che altro non è che l'insieme dei bit che costituiscono l'immagine codificata in Base64.

Le immagini richieste possono essere di due tipi: anteprima o immagine completa.

Nello specifico il metodo getImageWithUrl: recupera l'immagine completa (cioè quella con risoluzione maggiore) associata al parametro url.

getProductImageWithIdProduct:idImage: recupera l'immagine attraverso due parametri l'id prodotto e l'id immagine. Il primo parametro identifica il prodotto, il secondo identifica l'immagine relativa. Il secondo parametro ha senso, perché, come già detto, un oggetto può essere rappresentato da più immagini.

getProductImageWithIdImage:full: permette di recuperare l'immagine associata ad un id e di scegliere la tipologia mediante il parametro booleano full.

I tre metodi precedenti richiameranno tutti il metodo getImageWithUrl:resource:idt1:idt2:full: che è il metodo principale, cioè quello che si occupa di settare la connessione ed effettuare la richiesta vera e propria.

Questi metodi, seppur inseriti in una sezione a parte, sono anch'essi metodi REST, infatti, ricordando quanto detto nei precedenti paragrafi, consentono di recuperare una risorsa, nella fattispecie un'immagine.

Da ciò si deduce che il parametro HTTP_METHOD verrà settato a @"GET".

Diversamente da quanto accade per la richiesta di una risorsa di tipo XML, l'url di un'immagine sarà del tipo:

```
<URL_NEGOZIO>/api/images/products/idProd/idImage
```

idProd è il numero rappresentante l'id del prodotto, idImage quello relativo all'id dell'immagine. Questo indirizzo rappresenta l'immagine originale, esiste anche la possibilità di richiedere al web server un'immagine ridimensionata, in questo caso l'url della richiesta varia secondo lo schema:

```
<URL_NEGOZIO>/id#-xxx/image.jpg
```

rappresenta l'id numerico dell'immagine richiesta, xxx una stringa che identifica la caratteristica dell'immagine. L'url <URL_NEGOZIO>/id83-small/image.jpg, ad esempio, permette di richiedere un'immagine piccola associata all'id 83, rinominata image.jpg, che potrà essere utilizzata ad esempio come anteprima. La caratteristica small è una grandezza dichiarata nelle risorse image_type. Se questa grandezza è 54x72, small sarà associato a questa size.

Per gli scopi del framework verranno utilizzati solo gli url del tipo:

```
<URL_NEGOZIO>/id#/image.jpg
<URL_NEGOZIO>/id#-large/image.jpg
```

Il primo rappresenta l'immagine completa, il secondo l'immagine ridimensionata utile per le anteprime.

3.3.4 Utility

Quando si invia una richiesta di scrittura al server con contenuto xml, è necessario che quest'ultimo sia formattato secondo lo standard. A tal fine l'ultimo metodo della classe `PrestashopRESTLib` si occupa di inserire, all'inizio della stringa rappresentante il file xml, l'identificativo della versione e la codifica dei caratteri utilizzata dallo stesso, utili per il riconoscimento del file e il corretto funzionamento della piattaforma.

Il metodo in questione è `xmlToStringWithRoot:`. Esso riceve la stringa xml (parametro root) e la restituisce con l'aggiunta della prima riga identificativa: `<?xml version="1.0" encoding="UTF-8"?>` (in questo caso si può notare come la versione utilizzata sia la 1.0 e la codifica UTF-8).

CAPITOLO 4
PROGETTO DI ESTENSIONI iOS-ORIENTED DEL FRAMEWORK PRESTASHOP

4.1 ARCHITETTURA DEL FRAMEWORK (ESTENSIONI)

Nel capitolo precedente abbiamo analizzato l'architettura del framework di Prestashop soffermandoci sugli aspetti di basso livello del tutto trasparenti allo sviluppatore. Nello specifico si è analizzato il livello "Connessione Remota", il cui componente principale è la classe `PrestashopRESTLib`, responsabile dell'instaurazione e della gestione della comunicazione remota. In questo capitolo verranno proposte delle estensioni alle attuali funzionalità offerte da Prestashop, verranno analizzati, quindi, gli oggetti dell'insieme `Domain` e la classe `WSEntryPoint`, elementi utili per aggiungere nuove funzionalità alla piattaforma e che permettono ad un'app nativa di utilizzare il framework nel suo insieme, dato che fungono da "interfaccia" per lo stesso. Uno sviluppatore per utilizzare il framework di Prestashop non utilizzerà i metodi della classe `PrestashopRESTLib` dato che questi vengono utilizzati solo ed esclusivamente dalla classe `WSEntryPoint`.

4.1.1 I livelli "Domain" e "Funzionalità"

La seconda parte del modello architetturale del framework di Prestashop è incentrata sulla classe `WSEntryPoint` che coordina tutte le funzioni dell'applicazione con quelle dei livelli inferiori ed estende le funzioni di base dalla piattaforma Prestashop.

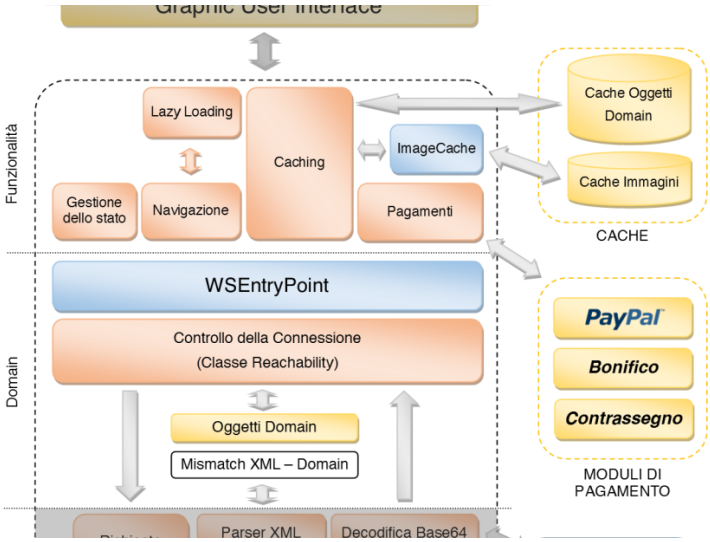

Figura 18 – Domain e Funzionalità (Architettura)

DOMAIN:

Ricapitolando la classe `PrestashopRESTLib` esegue le richieste remote, riceve o invia file XML e ritorna le immagini decodificate. Questi compiti, dal punto di vista del livello "Domain", possono essere rappresentati da quattro casi:

Il primo caso è quello che si verifica quando viene effettuata, attraverso la GUI dell'applicazione, una richiesta remota. Esso precede tutti gli altri, dato che il modello d'interazione è cliente/servitore e quindi l'iniziativa verrà sempre presa dall'utente.

Il successivo caso si verifica quando viene ricevuto un file XML correttamente formattato. Tale file dovrà essere letto e convertito in un tipo di dato che sia facilmente manipolabile dall'applicazione. Se viene ricevuto un elenco di categorie, queste non potrebbero essere utilizzate perché il file XML è un file testuale e l'applicazione non è in grado di leggerlo, o meglio non è in grado di riconoscere tutti gli elementi contenuti nello stesso (in questo caso le categorie di prodotti). Per rendere queste risorse realmente manipolabili viene effettuato un processo di conversione che prende il nome di mismatch di dati, questo processo è simile a quello che viene effettuato quando un'applicazione interagisce con un database. D'altro canto, un file XML ha molte analogie con le tabelle dei database anche se completamente diversi nella struttura. Il processo di conversione dei dati, quindi, permette di convertire una risorsa XML in un oggetto definito da una classe associata e definita nell'insieme `Domain`. Queste classi implementano dei metodi che permettono di accedere alle proprietà di ogni oggetto. In un oggetto `Cart` (carrello della spesa), ad esempio, ci saranno quei metodi che permettono di leggere i prodotti contenuti nello stesso, aggiungere o rimuovere un prodotto, etc.

Quando un utente effettua una richiesta di creazione o di modifica di una risorsa sul server, ci troviamo nel terzo caso. Le operazioni che verranno svolte sono la creazione della stringa di richiesta e la formattazione e compilazione di un file XML, nel quale saranno presenti i dati da creare o modificare. La creazione della richiesta è analoga per tutti i casi mentre la seconda è più particolare. La formattazione e la compilazione di un file XML è l'operazione inversa alla conversione del file in un oggetto di `Domain`. Il mismatch dei dati qui viene effettuato partendo da quest'ultimo oggetto o da un template XML i cui elementi sono incompleti, fino ad arrivare ad un file XML ben formattato. Successivamente questo file verrà affidato ai livelli sottostanti che lo invieranno al server tramite il body del messaggio HTTP, generato a partire dalla richiesta.

La ricezione dell'immagine decodificata, ultimo caso, comporta il salvataggio nella cache o la visualizzazione della stessa in una vista dell'applicazione. Fortunatamente le immagini in Objective-C possono essere rappresentate attraverso un `NSData` (dati binari) o mediante un `UIImage` (dato immagine). Entrambi questi tipi di dato sfruttano i metodi delle loro classi per essere gestiti, sgravando il compito a `WSEntryPoint` che potrà utilizzarli senza operazioni preliminari.

Tra la classe `WSEntryPoint` e le operazioni sottostanti troviamo il "controllo della connessione". Questa operazione viene effettuata dalla classe `Reachability`, messa a disposizione da Apple, che permette di controllare lo stato di connettività di un dispositivo. L'utilizzo di questa è importante. Avendo a che fare, infatti, con un applicativo che richiede l'accesso alla rete, l'utente deve essere notificato in caso di assenza della stessa, poiché altrimenti si ritroverebbe davanti a delle schermate vuote, cioè prive di dati, data l'irreperibilità di quest'ultimi.

FUNZIONALITÀ:

Il livello “Funzionalità” è rappresentato da blocchi che riassumono una serie di operazioni svolte dai uno o più metodi della classe `WSEntryPoint` o da altre classi esterne, come `imageCache`, e rappresentano le astrazioni delle funzioni offerte dal framework. La gestione dello stato, ad esempio, è una funzionalità non fornita direttamente dal framework, ma possibile mediante la cooperazione di vari metodi. Tale funzionalità la si può ottenere dalla combinazione tra le classi `LoginViewController` e `NewCustomerViewController`, che gestiscono le viste dell’applicazione, ed i metodi Customer di `WSEntryPoint`.

Le funzioni descritte nel livello “Funzionalità” della figura 18 sono: gestione stato, navigazione nell’applicazione, pagamenti e caching.

La **gestione dello stato** permette all’applicazione di estendere le sue funzionalità ad utenti che hanno effettuato l’accesso. Gli utenti senza credenziali non potranno sfruttare appieno le funzioni offerte dal framework. Il motivo di questa affermazione è legato al fatto che alcuni metodi della classe `WSEntryPoint` dipendono dall’id di un utente. Basti pensare che il carrello della spesa è strettamente collegato ad una persona, così come gli ordini ed gli indirizzi di spedizione.

Il blocco **Navigazione** racchiude tutti gli spostamenti che vengono effettuati dall’utente all’interno dell’app. Per spostamento non s’intende semplicemente il cambiamento di vista, ma l’insieme dei metodi invocati per comporre l’intera schermata che verrà visualizzata dall’utente. Sopra il blocco navigazione troviamo il blocco **lazy loading**. Questo blocco rappresenta una delle funzioni utili a rendere efficiente l’app. Al momento ci limitiamo a dire che il suo scopo principale è quello di annullare i tempi di attesa quando si scaricano liste di prodotti molto grandi, sbloccando l’applicazione che altrimenti rimarrebbe sospesa fino al completamento del download di tutti i prodotti contenuti nella lista. Questa funzione, tuttavia, non viene sempre utilizzata dal blocco navigazione, ma può essere considerata un’estensione dello stesso.

Il framework fa uso di due tipi di cache: una per le immagini e l’altra per gli oggetti di `Domain`. Tale funzionalità è rappresentata dal blocco **Caching**. Sempre dalla figura 18, si nota che questo blocco comunica proprio con le due tipologie di cache, con la differenza che quella per le immagini utilizza una classe a parte, `imageCache`, e dato che a differenza della prima è persistente, cioè non viene distrutta all’uscita dell’applicazione, necessita di metodi creati ad hoc per gestirla.

Il blocco **Pagamenti** è quello che si occupa di coordinare la comunicazione con i moduli di pagamento esterni. I moduli utilizzati dal framework sono tre: Contrassegno, Bonifico Bancario e PayPal. Si considerano “esterni” perché devono essere abilitati nelle impostazioni del negozio virtuale creato sulla piattaforma Prestashop.

4.2 LA CLASSE WSEntryPoint

La classe `WSEntryPoint` implementa quei metodi utilizzati per reperire gli elementi che verranno mostrati nell'interfaccia grafica o per svolgere le operazioni richieste dall'utente, come la registrazione, il login, l'aggiunta di oggetti al carrello, la navigazione nel catalogo prodotti, e tanti altri.

Questa classe è molto importante perché in essa sono presenti le credenziali di accesso alle API del web server di un generico store. Questi dati sono contenuti nelle variabili d'istanza della classe definite nel file header `WSEntryPoint.h` e quelle più rilevanti sono l'indirizzo url del negozio online (`WS_URL`), la chiave di accesso alle API (`WS_KEY`), l'id dello shop (`idShop`) e del gruppo (`idShopGroup`) (quest'ultimo in caso di multistore), e l'istanza della classe `PrestashopRESTLib` (`psWebService`).

4.2.1 I metodi

I metodi della classe `WSEntryPoint`, come del resto quelli di `PrestashopRESTLib`, sono organizzati anch'essi in sezioni per essere facilmente reperibili allo sviluppatore che li utilizza. La prima parte dell'implementazione della classe è costituita da quelli che permettono di istanziarla, di recuperarne un'istanza e altri metodi ausiliari che permettono di controllare la connettività di un dispositivo e svolgere altri compiti non riferibili alle sezioni dei metodi seguenti. Le sezioni di metodi comprendono quelli relativi agli account utente, ai prodotti, al carrello, agli ordini ed alle informazioni del negozio virtuale.

Iniziamo l'analisi della classe, partendo dalle firme dei metodi iniziali:

```
- (id)init;
+ (WSEntryPoint *) getInstance;
- (void) checkNetwork;
- (void) invalidateCachesAfterOrder;

+ (NSString *) getInnerTextWithElement: (DDXMLElement *) element tagName:
    (NSString *) tagName defaultValue: (NSString *) defaultValue;
+ (NSString *) getInnerTextWithElement: (DDXMLElement *) element tagName:
    (NSString *) tagName languageSpecific: (BOOL) languageSpecific defaultValue:
    (NSString *) defaultValue;
```

Il primo metodo, `init`, è ovviamente quello che serve per istanziare la classe. È un metodo ridefinito a partire da quello di `NSObject` perché svolge delle operazioni preliminari non implementate da quello della superclasse. Queste operazioni riguardano l'inizializzazione sia delle variabili d'istanza della classe che delle varie cache, compresa quella per le immagini dei prodotti. All'interno del corpo di questo metodo si trova anche l'inizializzazione della classe `PrestashopRESTLib` che, ricordando quanto detto prima, permette di effettuare la connessione con il server remoto. Dal precedente capitolo, l'`init` della classe

PrestashopRESTLib veniva invocato passandogli tre parametri: l'url, la chiave e la variabile booleana debug. Trascurando quest'ultima, le prime due si ottengono dalle variabili d'istanza della classe WSEntryPoint, WS_URL e WS_KEY. La stringa d'invocazione sarà dunque:

```
psWebService = [[PrestaShopWebservice alloc] initWithUrl: WS_URL key:
    WS_KEY debug: YES];
```

Il metodo getInstance è un metodo di classe (preceduto da un +) che permette di recuperare un'istanza della classe WSEntryPoint. È molto utile quando si vuole invocare un metodo della stessa. Senza di esso, per invocare il medesimo si dovrebbe istanziare un altro oggetto WSEntryPoint, ma ciò non avrebbe senso perché, a parte il fatto che si occuperebbe inutilmente memoria, il metodo chiamato non potrebbe utilizzare la cache precedente, dato che quest'ultima verrebbe inizializzata nuovamente.

Il metodo checkNetwork funge da controllo della connettività. È un metodo void, cioè che non restituisce nulla, ed il suo compito è quello di visualizzare una finestra di dialogo qualora la connessione fosse assente. È importante perché avvisa l'utente che l'app non può funzionare correttamente, visto che, in questo caso, non ci sarebbe comunicazione con il server remoto. Il metodo in questione implementa anche funzionalità di controllo del tipo di connessione, cioè se un dispositivo è connesso tramite rete mobile o WiFi. Questa funzionalità, però, non è richiesta per il corretto funzionamento del framework, ma implementata per usi futuri. Il controllo della connessione viene effettuato per mezzo della classe Reachability di Apple.

invalidateCachesAfterOrder permette di svuotare la cache dopo che un ordine viene effettuato con successo. L'unica operazione che svolge è semplicemente quella di settare tutti gli oggetti della cache a nil.

Dato che Prestashop è una piattaforma internazionalizzabile, non si esclude la possibilità che gli elementi dei file XML, relativi alle risorse richieste, siano localizzati per più lingue. Le classi getInnerTextWithElement:tagName:defaultValue: e getInnerTextWithElement:tagName:languageSpecific:defaultValue: permettono proprio di estrarre la stringa localizzata di un elemento, qualora fosse presente, viceversa, verrà restituita una stringa valida per tutte le lingue. Il primo metodo richiama il secondo settando a default il parametro languageSpecific a NO; Il secondo si occupa di estrarre la stringa relativa al tag (parametro tagName) dell'elemento XML (parametro element). Infine, il parametro defaultValue è la stringa che verrà ritornata dai metodi, qualora il valore dell'elemento richiesto non esistesse.

Metodi Customer

La prima sezione della classe `WSEntryPoint` è quella dei metodi Customer, o metodi utente.

```
- (Customer *) checkCredentialsWithEmail: (NSString *) email password:
    (NSString *) password;
- (BOOL) registerNewCustomerWithFirstName: (NSString *) firstName lastname:
    (NSString *) lastName password: (NSString *) password email: (NSString *)
    email idGender: (NSString *) gender birthday: (NSString *) birthday;
```

All'interno di questa sezione, come si può notare dalle precedenti firme, ci sono due metodi: `checkCredentialsWithEmail:password:` e `registerNewCustomerWithFirstName:lastName:password:email:idGender:birthday:`.

Il primo metodo permette di effettuare il controllo delle credenziali di un utente. È usato come metodo di login, infatti, riceve proprio i due parametri: `email` e `password`. Quando questo metodo viene invocato viene richiesto al web server il file XML associato al parametro email. Se l'XML è presente, il web server ritornerà la risorsa `Customer` viceversa un'eccezione ("file not found"). Una volta ottenuto quest'ultimo, verrà confrontato il parametro `password`, dopo l'opportuna codifica, con l'elemento già codificato del file ricevuto. Se il confronto ha esito positivo, cioè se le stringhe coincidono, il metodo ritornerà un oggetto `Customer` di che servirà per effettuare il login ed inizializzare l'oggetto `Cart`.

Il secondo metodo permette di registrare un nuovo utente, quindi riceverà tutti i parametri necessari per completare il template XML `customerTemplate`, di tipo `NSString`, definito sempre all'interno della classe `WSEntryPoint`. La richiesta di registrazione, essendo un'operazione di scrittura, richiederà un file XML da inviare al web server. Il motivo di utilizzare un template è quello di avere un file XML correttamente strutturato ed inoltre si può sfruttare il fatto che il linguaggio objective-c permette di creare una nuova stringa sostituendo il carattere di escape `%@` con altre stringhe. L'operazione è analoga a quella che avviene quando s'invoca `printf` nel linguaggio c.

Esempio in objective-c:

```
NSString *customerTemplate = @"<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
            "<prestashop xmlns:xlink=\"http://www.w3.org/ 1999/xlink\">"
                "<customer>"
                    "<firstname><![CDATA[%@]]></firstname>"
                    "<lastname><![CDATA[%@]]></lastname>"
                    "<email><![CDATA[%@]]></email>"
                "</customer>"
            "</prestashop>";
```

Esempio di template

```
NSString *firstName = @"Ilario";
NSString *lastName = @"Genuardi";
NSString *email = @"mia@email.it";
NSString *customerXml = [NSString stringWithFormat: customerTemplate,
    firstName, lastName, email];
```

Invocazione: Vengono sostituiti i caratteri `%@` di `customerTemplate` con le stringhe `firstName`, `lastName` ed `email`.

```
@"<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
    "<prestashop xmlns:xlink=\"http://www.w3.org/ 1999/xlink\">"
        "<customer>"
            "<firstname><![CDATA[Ilario]]></firstname>"
            "<lastname><![CDATA[Genuardi]]></lastname>"
            "<email><![CDATA[mia@email.it]]></email>"
        "</customer>"
    "</prestashop>";
```

Risultato: Contenuto di `customerTemplate`

Quest'esempio mostra come viene formato il file XML da inviare, tuttavia il template delle figure non è fedele all'originale usato dalla classe, in quanto sono presenti molti più elementi, ma il concetto però è lo stesso. Altri metodi di altre sezioni utilizzano lo stesso principio, ma verranno analizzati in seguito. Dopo la creazione di questa stringa e prima dell'invio della stessa, il metodo `registerNewCustomerWithFirstName:lastName:password:email:idGender:birthday:` effettua il controllo sulla correttezza di tutti i parametri, confrontandoli con espressioni regolari (RegEX). Se i parametri sono tutti corretti il file verrà inviato, altrimenti verrà sollevata la solita eccezione, differenziata per tipologia di errore. Una volta ricevuto il file XML, il server remoto comunicherà al framework l'esito dell'operazione: registrazione effettuata, email già registrata o file XML non corretto.

Metodi Product

I metodi della sezione Product sono quei metodi che si occupano di recuperare categorie, prodotti, associare le immagini ai vari prodotti e altri metodi che svolgono operazioni correlate.

```
- (NSMutableArray *) getCategories;
- (NSMutableArray *) getProductsWithCategory: (id) category listener:
    (ProductsViewController *) listener;
- (Product *) getPartialProductWithId: (NSString *) productId;
- (Product *) getFullProductWithId: (NSString *) productId;
- (Product *) fetchPartialProductWithId: (NSString *) productId;
- (BOOL) fetchFullProductWithProduct: (Product *) product;
- (ProductOption *) getProductOptionWithId: (NSString *) productOptionId;
- (NSMutableDictionary *) getProductOptionNames;
- (SpecificPrice *) getProductSpecificPricesWithId: (NSString *) productId;
- (SpecificPrice *) getCombinationSpecificPricesWithId: (NSString *)
    combinationId;
- (void) downloadSpecificPrices;
- (float) getTaxRateWithId: (NSString *) taxId;
- (void) downloadTaxRates;
```

Partendo dalle firme dei metodi precedenti, `getCategories` si occupa di scaricare le varie categorie di prodotti o sottocategorie. Il metodo ritorna un `NSMutableArray` contenente gli oggetti `Category`. Gli oggetti `Category`, come si vedrà nel prossimo paragrafo, contengono tutte le informazioni di una categoria di prodotti, in particolare se una categoria comprende sottocategorie. Lo sviluppatore può riutilizzare questo metodo anche per recuperare le sottocategorie di prodotti.

`PrestaShop` organizza le categorie in livelli. I livelli 0 e 1 non sono definiti, il 2 rappresenta la categoria principale il 3 la sottocategoria. Per recuperare i vari livelli, il metodo utilizza un filtro nella richiesta url `filter[level_depth]`. Nel paragrafo 3.3.2 un esempio di utilizzo dell'opzione `filter[]`.

Il metodo `getProductsWithCategory:listener` è particolarmente interessante perché utilizza una funzionalità del framework, il lazy loading (paragrafo 4.3). Si occupa dello scaricamento dei prodotti associati ad una categoria. Ogni prodotto scaricato verrà visualizzato dall'interfaccia utente singolarmente, cioè aggiunto alla vista una volta ricevuto, il parametro `listener` è proprio la classe che gestisce quest'interfaccia. Dalla firma del metodo si può notare che Il primo parametro, `Category`, è definito di tipo `id` (parola chiave del linguaggio objective-c che definisce il tipo astratto di parametro, da non confondere con gli identificativi). Il motivo di utilizzare il tipo astratto, e non direttamente un oggetto `Category`, è legato ad una problematica di gestione del tipo di parametro proprio del linguaggio. Tale problematica non verrà analizzata in questo testo dato che non è attinente.

La cache generata da questo metodo contiene liste di prodotti parziali o complete. Il motivo di ciò è legato al fatto che questo metodo può essere interrotto esternamente, settando la variabile `stopDownloadProducts` a `YES`. Se il metodo termina senza interruzioni ritornerà una lista completa, viceversa quella parziale che comunque potrà essere completata in seguito, qualora l'utente richiedesse nuovamente i prodotti associati alla stessa categoria.

`getPartialProductWithId:`, `getFullProductWithId:`, `fetchPartialProductWithId:` e `fetchFullProductWithProduct:` rappresentano quei metodi utili per recuperare i prodotti dal server remoto.

`getPartialProductWithId:` cerca l'id del prodotto nella cache e se non presente invoca `fetchPartialProductWithId:` che si occupa di recuperare il prodotto dal web server, associargli l'immagine di anteprima e recuperare la quantità di prodotti disponibili nel negozio online.

`getFullProductWithId:` è analogo a `getPartialProductWithId:`, ma a differenza di quest'ultimo permette di recuperare tutte le informazioni legate ad un prodotto. Inizialmente questo metodo ricerca l'id nella cache, se presente controlla che la proprietà `isComplete` dell'oggetto `Product` sia `YES`, se non lo è richiama il metodo `fetchFullProductWithProduct:` che permette di reperire tutti i dati e le immagini complete associate al prodotto richiesto. Quest'ultimo metodo verrà richiamato anche nel caso l'id non sia contenuto nella cache. Da notare che il metodo non restituisce un prodotto, ma solo l'esito, questo perché viene invocato sempre da `getFullProductWithId:`.

Gli altri metodi che completano la sezione implementano alcune funzionalità per associare più informazioni ad un prodotto. Molti di essi, infatti, vengono invocati all'interno di

`fetchFullProductWithProduct:`. Senza soffermarci troppo sul loro funzionamento verrà descritto solo il loro compito:

- `getProductOptionWithId:` ritorna un oggetto `ProductOption` in base all'id del prodotto;
- `getProductOptionNames` ritorna una lista con i nomi delle opzioni legate ad un prodotto;
- `getProductSpecificPricesWithId:` recupera lo sconto del prodotto il cui id viene passato come parametro;
- `getCombinationSpecificPricesWithId:` recupera il prezzo scontato della combinazione in base all'id di quest'ultima;
- `downloadSpecificPrices` scarica i prezzi scontati dei prodotti in promozione;
- `getTaxRateWithId:` recupera una tassa da applicare al prezzo del prodotto in base all'id;
- `downloadTaxRates` recupera tutte le tasse da applicare ai prezzi dei prodotti.

Metodi Cart

```
- (Cart *) getCartWithId: (NSString *) cartId;
- (Cart *) createNewEmptyCartWithCustomerId: (NSString *) customerId;
- (DDXMLElement *) updateCartWithId: (NSString *) cartId cartXml: (NSString *)
    cartXml;
- (NSMutableArray *) getCartRowHelpersWithCart: (Cart *) cart;
- (NSString *) addAddress: (Address *) address;
- (NSMutableArray *) getCarriers;
- (float) getCastOnDeliveryPrice;
- (NSMutableArray *) getAddressesWithCustomerId: (NSString *) customerId;
- (State *) getStateWithId: (NSString *) stateId;
- (NSMutableArray *) getActiveStates;
- (Country *) getCountryWithId: (NSString *) countryId;
- (NSMutableArray *) getActiveCountries;
- (Country *) getActiveCountryWithIndex: (int) index;
- (NSMutableArray *) getActiveStatesForCountry: (NSString *) countryId;
- (State *) getActiveStatesForCountry: (NSString *) countryId index: (int)
    index;
```

I prodotti che un utente intende acquistare, e che quindi seleziona dal "catalogo" messo a disposizione da un negozio online, verranno raccolti all'interno di un carrello della spesa virtuale. L'operazione è simile a quella che avviene quando un individuo si reca al supermercato e prende un prodotto dallo scaffale. Nel nostro caso, per raccogliere i prodotti scelti da uno store, la classe `WSEntryPoint` mette a disposizione i metodi per gestire e creare un carrello virtuale, definito dall'oggetto `Cart`.

I metodi di questa sezione svolgono operazioni analoghe ai quelli delle altre, ma con la differenza che variano i tipi di dato richiesti. Ad esempio, come abbiamo visto precedentemente, un prodotto può essere recuperato mediante il suo id. Questa operazione è del tutto simile a quella che avviene per recuperare un carrello della spesa. Ovviamente ci sono delle differenze a livello di implementazione. Per quanto detto, in questa sezione ci occuperemo solo delle differenze sostanziali, trascurando, quindi, le cose già affrontate nei metodi delle altre.

Il primo metodo, `getCartWithId:`, si occupa di recuperare un'istanza del carrello in base all'id.

`createNewEmptyCartWithCustomerId:`, permette di creare un nuovo oggetto `Cart` a partire dall'id dell'utente. Il nuovo carrello viene creato a partire dal template `emptyCartTemplate`, definito all'interno di `WSEntryPoint`. L'operazione è simile alla registrazione di un nuovo utente con la differenza che il template sarà completato dall'id dell'utente, passato come parametro, e dagli id del negozio (`idShop`) e del gruppo (`idShopGroup`), variabili d'istanza della classe `WSEntryPoint`.

`updateCartWithId:cartXml:`, aggiorna il contenuto del carrello rappresentato dall'id passato come parametro. Elemento interessante di questo metodo è il secondo parametro, `cartXml`, che altro non è che il file XML che verrà inviato al server per modificare il contenuto del carrello. Bisogna sempre ricordare che la piattaforma Prestashop crea e modifica le sue risorse mediante il passaggio di un file XML ben formattato. Apparentemente si può pensare che sarebbe più opportuno utilizzare una variabile locale del carrello ed inoltrarla al server remoto solo quando si effettua l'ordine, ciò però toglierebbe la sincronicità tra l'applicazione e il server, generando problemi di aggiornamento delle risorse remote, come ad esempio la quantità disponibile di un prodotto.

Il metodo `getCartRowHelpersWithCart:` permette di estrarre, da un oggetto `Cart` passato come parametro, la lista degli oggetti `CartRow` che rappresentano i prodotti inseriti nel carrello.

`addAddress:` è il metodo che dà la possibilità ad un utente di associare uno o più indirizzi di spedizione al suo account. Il metodo riceve un parametro di tipo `Address` che verrà opportunamente convertito in file XML ed inoltrato al server.

`getCarriers`, recupera la lista dei corrieri, o meglio dei metodi di spedizione, disponibili per un determinato indirizzo.

`getCastOnDeliveryPrice` è il metodo che si occupa di calcolare le spese aggiuntive qualora un utente decidesse di pagare in contrassegno.

`getAddressesWithCustomerId:`, permette di scaricare gli indirizzi di spedizione associati ad un id utente.

I metodi seguenti si occupano di recuperare le nazioni e le provincie disponibili. Tali metodi sono utili ai fini di calcolare le spese di spedizione, differenziate appunto per zone, corrieri disponibili e tasse da applicare ai prezzi.

`getStateWithId:` e `getCountryWithId:` sono due metodi che restituiscono rispettivamente un oggetto `State` rappresentante una provincia e un oggetto `Country` rappresentante una nazione. Entrambi gli oggetti vengono recuperati mediante il loro id.

`getActiveStates` e `getActiveCountries` restituiscono la lista delle provincie e delle nazioni attive per un negozio. Non tutti gli store, infatti, offrono servizi per tutte le zone, alcuni ad esempio non spediscono in zone disagiate, altri operano solo su un unico territorio nazionale. Nel caso di `getActiveCountries`, un negozio che spedisce solo in Italia avrà

soltanto una nazione attiva e la lista risultante conterrà proprio un unico oggetto, di tipo `Country`, rappresentante appunto l'Italia.

`getActiveCountryWithIndex:`, permette di accedere ad un elemento della lista restituita da `getActiveCountries` specificandone l'indice. Tale metodo può essere utilizzato in situazioni che richiedono iterazione, ad esempio quando si utilizza un "picker", cioè un selettore con funzionalità simili ai menu a tendina usati nei form delle pagine web o in alcune applicazioni.

`getActiveStatesForCountry:`, ritorna la lista di provincie associate ad una nazione. Questa lista in alcuni casi può essere vuota, ciò è dovuto al fatto che una nazione può non essere divisa in provincie. Se consideriamo il caso della Svizzera, che non ne possiede, tale lista sarà vuota.

`getActiveStatesForCountry:index:` è il metodo corrispettivo di `getActiveCountryWithindex:`, ma associato alla lista restituita da `getActiveStatesForCountry`.

Metodi Order

```
- (NSString *) addOrderWithXMLString: (NSString *) orderXml;
- (BOOL) editOrderStateWithId: (NSString *) orderId stateId: (NSString *)
    stateId;
- (BOOL) editOrderTotalPaidRealWithId: (NSString *) orderId totalPaidReal:
    (NSString *) totalPaidReal;
- (float) getShippingCostWithCarrierId: (NSString *) carrierId countryId:
    (NSString *) countryId stateId:(NSString *) stateId;
```

I metodi Order sono quelli che si occupano di effettuare un ordine e gestirne lo stato. `addOrderWithXMLString:` è il metodo che identifica la sezione ed è quello che permette d'inoltrare un ordine al server. Il parametro `orderXML` rappresenta il file XML che verrà inviato. L'operazione svolta è semplicemente quella di inviare il file al server ed attendere la risposta da quest'ultimo. Dalla firma del metodo si nota che il valore di ritorno è una stringa. Dato che l'operazione può assumere solo due esiti, può sembrare strano che questo valore sia una stringa anziché un valore booleano, ma il motivo di utilizzare tale tipo di dato è sensato in quanto il server ritornerà l'id dell'ordine se accettato, quindi è consigliabile questo approccio poiché in questo caso avremo due informazioni al posto di una: l'id dell'ordine ed indirettamente l'esito dell'operazione (la ricezione dell'id implica che l'ordine sia stato effettuato con successo). Tra le altre cose, quanto detto assume maggior importanza se si pensa che altri metodi di questa sezione hanno bisogno dell'id dell'ordine per essere utilizzati ed è conveniente, quindi, averlo a disposizione.

Il metodo `editOrderStateWithId:stateId:` permette di modificare lo stato di un ordine mediante il suo id e l'id dello stato che identifica se un ordine è in attesa di pagamento, in lavorazione, etc. Per far ciò, richiede al server la risorsa rappresentate l'ordine che alla ricezione verrà convertita, grazie al metodo `xmlToStringWithRoot:` di `PrestashopRESTLib`, in un file XML di tipo `NSString`. Successivamente il metodo `editOrderStateWithId:stateId:` estrarrà l'elemento `current_state`, ovvero lo stato

corrente dell'ordine, e lo sovrascriverà con il valore del parametro `stateId`. Infine, invierà il file appena modificato al server che si occuperà di aggiornare la risorsa con in nuovi dati. Il valore di ritorno del metodo è di tipo booleano e rappresenta l'esito dell'operazione.

`editOrderTotalPaidRealWithId:totalPaidReal:`, svolge un'operazione molto simile al metodo precedente, con la differenza che verrà modificato, al posto di `current_state`, l'elemento `totalPaidReal`, cioè il prezzo totale dell'ordine.

L'ultimo metodo di questa sezione, `getShippingCostWithCarrierId:countryId:stateId:`, è quello che si occupa di calcolare le spese di spedizione, differenziate in base al tipo di spedizione scelto ed alla zona dove verrà spedito un ordine. Il valore ritornato da questo metodo è di tipo float e rappresenta proprio il costo di spedizione. I tre parametri `carrierId`, `countryId`, `stateId` sono rispettivamente l'id del corriere, quello della nazione e della provincia se presente. Quest'ultimo parametro è infatti facoltativo, dato che una nazione può non averne. In questo caso il valore che dovrà essere passato sarà `@"0"`.

Altri metodi

Un negozio di e-Commerce mette a disposizione prodotti scontati o espone gli ultimi arrivi in primo piano. I metodi di questa sezione si occupano proprio di ciò.

```
- (NSMutableArray *) getNewArrivalsWithListener: (ProductsViewController *)
    listener;
- (NSMutableArray *) getSpecialOffersWithListener: (ProductsViewController *)
    listener;
```

`getNewArrivalsWithListener:` e `getSpecialOffersWithListener:` sono due metodi che accettano un parametro `listener` di tipo `ProductViewController`, esattamente come il metodo `getProductsWithCategory:listener:`. Avendo a che fare con il download di un insieme di prodotti, implementano anch'essi il lazy loading.

Il metodo per scaricare i nuovi prodotti non fa altro che ritornare un array contenente i prodotti inseriti nell'ultimo periodo. Per riconoscere se un prodotto è un "nuovo arrivo" o no, vengono confrontati due valori. Il primo valore viene recuperato dal file di configurazione di PrestaShop. Esso è un intero che rappresenta l'intervallo di tempo, espresso in secondi, che viene utilizzato per considerare un prodotto "nuovo arrivo". Il secondo valore è rappresentato dalla differenza tra l'istante "attuale" e la data di aggiunta del prodotto. Quindi se il primo valore è maggiore o uguale al secondo il prodotto verrà considerato "nuovo arrivo", viceversa vecchio. Un "nuovo arrivo", quindi, per essere considerato tale deve essere contenuto nell'intervallo definito in figura.

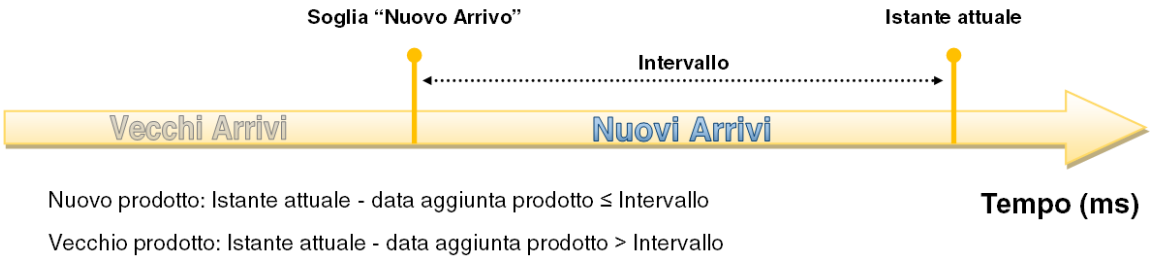

Figura 19 – Diagramma temporale "Nuovi Arrivi"

Tutti i valori precedenti sono convertiti in millisecondi per ragioni di compatibilità.

L'altro metodo di questa sezione è quello utilizzato per scaricare i prodotti in offerta. L'operazione svolta da esso è più semplice, rispetto al metodo precedente, perché non fa altro che richiedere gli id degli oggetti con i prezzi scontati. Successivamente restituirà anch'esso un array di prodotti definiti da tali id.

Metodi CMS

L'ultima sezione della classe `WSEntryPoint` comprende un solo metodo:

```
- (NSMutableDictionary *) getShopAbouts;
```

Questo metodo fornisce tutte le informazioni sul negozio virtuale, a partire dalla descrizione del negozio fino ad arrivare ai recapiti. Il metodo in questione restituisce un oggetto `NSMutableDictionary` contenente coppie organizzate per tipologia. Ad esempio la voce "pagamento" sarà associata alla descrizione dei metodi di pagamento accettati e la voce "spedizione" a quella dei metodi di spedizione. Tipicamente queste descrizioni vengono raggruppate per essere visualizzate in un'unica schermata, al fine di agevolare l'utente che può leggerle e ritrovarle facilmente. Queste descrizioni vengono restituite sotto forma di stringhe HTML, ragion per cui è opportuno convertirle in stringhe UTF-8 per evitare la visualizzazione di caratteri non desiderati.

4.2.2 Gli Oggetti (Customer, Cart, Product, Order, etc)

Le classi che assieme `WSEntryPoint` fanno parte dell'insieme `Domain` sono le rappresentazioni delle tipologie di oggetti XML richiesti al web server di Prestashop.

Questi oggetti e le loro proprietà per essere "comprese", dall'applicazione che li utilizza, devono essere convertiti in tipi di dato del linguaggio Objective-C.

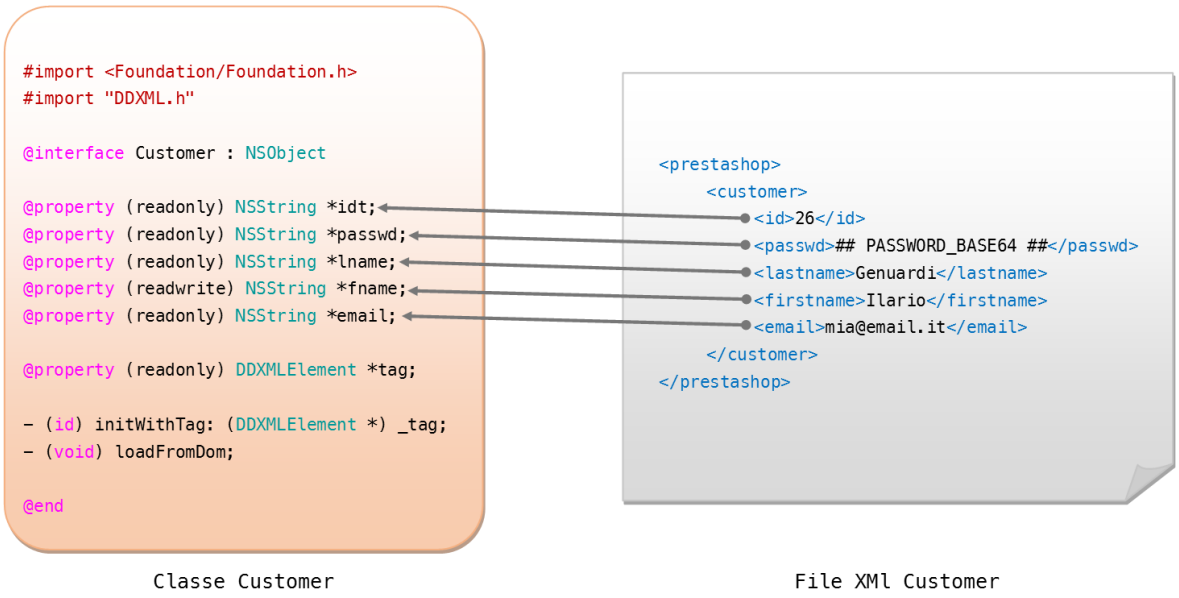

Figura 20 – Esempio di mismath elemento Customer

Finora è stato illustrato il processo di reperimento dei file XML, ma non di come questi vengano convertiti in oggetti Objective-C.

Nella figura precedente un esempio di associazione tra un file XML ed una classe. Le frecce indicano la relazione dei vari elementi del file XML con le relative proprietà della classe. L'XML Customer con id 26 diventerà, quindi, un'istanza della classe `Customer`, le cui variabili avranno gli stessi valori degli elementi XML.

Ma come avviene il tutto? Quando si richiede un file XML, l'elemento root di quest'ultimo verrà ritornato dalla classe `PrestashopRESTLib` sotto forma di `DDXMLElement`. Questo tipo di dato rappresenta il file XML. Senza un'opportuna estrazione degli elementi, però, questo dato risulterebbe inutile ai fini dell'applicazione, perché questi non potrebbero essere utilizzati. Per questo motivo si utilizzano proprio gli oggetti di `Domain`. Le classi rappresentanti questi oggetti hanno tipicamente due metodi: `initWithTag:` e `loadFromDom`. Il primo ridefinisce l'`init` della classe e permette di istanziarla passandogli una variabile `DDXMLElement`; il secondo, invocato alla fine del precedente metodo, estrae gli elementi XML, li converte e li assegna alle variabili della classe associata. Non tutti gli elementi del file XML verranno associati, soltanto quelli necessari ai fini del framework (nella figura, non sono rappresentati tutti gli elementi reali di un file XML di tipo customer, ma solo quelli utili).

Esistono varie tipologie di oggetti in `Domain`, ognuna delle quali è riferita ad un file XML ben preciso. Nella tabella uno schema:

Oggetto Domain	Descrizione	Proprietà
Address	Indirizzo di spedizione. Questo oggetto è utilizzato negli ordini come indirizzo di destinazione o di fatturazione. Viene utilizzato anche per calcolare le spese di spedizione. Un utente può averne associati più di uno.	idt, idCustomer, alias, company, lastName, firstName, address1, address2, postCode, city, phone, phoneMobile, vatNumber, idCountry, idState;
Carrier	Corriere espresso o altro metodo di spedizione. Si riferisce al metodo di spedizione scelto dall'utente.	idt, name, delay, idTaxes, nameTag, priceTag;
Cart	Carrello della spesa. La lista dei prodotti scelti dall'utente.	idt, idAddressDelivery, idAddressInvoice, idCurrency, idCustomer, idLang, idShop, idShopGroup, idCarrier, dateAdded, dateUpdated, cartRows, cartTag;
CartRow	Sono gli oggetti contenuti nella proprietà `cartRows` di `Cart`. Ognuno di essi rappresenta un prodotto aggiunto al carrello. Le sue proprietà sono l'id del prodotto, l'id della combinazione (per prodotti con scelte multiple) e la quantità scelta.	idProduct, idCombination, quantity;
CartRowHelper	Questo oggetto viene utilizzato per la rappresentazione dettagliata del prodotto nel carrello che verrà visualizzata dalla vista dell'app. A differenza degli altri, non deriva da un file XML, ma dai parametri passati al suo init.	product, cartRow, title, reference, price, image;
Category	Categoria. Rappresenta una lista di sotto-categorie (`subCategories`) o di prodotti.	idt, name, subCategories, categoryTag;
Combination	Combinazione. Alcuni prodotti possono avere varie configurazioni, questo oggetto rappresenta proprio le combinazioni di quest'ultime. Ad esempio un iPhone 4 può avere due colori (nero, bianco) e due dimensioni di memoria (16GB, 32GB). Ogni combinazione rappresenta una delle quattro coppie Nero-16GB, Nero-32GB, Bianco-16GB e Bianco-32GB.	idt, price, minQuantity, quantity, reference, options, imagesId, specificPrice, tag;

Country	Nazione. Questo oggetto si riferisce ad uno stato (Italia, Svizzera, Germania, etc). La proprietà `hasStates` è un valore booleano che indica se lo stato è diviso in provincie.	idt, name, hasStates, tag;
Customer	Utente. Rappresentazione di un account utente. Utilizzato per gestire i login e logout in un'applicazione.	idt, fname, lname, email, passwd;
Order	Ordine. Nella pratica è l'oggetto utilizzato per finalizzare la vendita. Tra le sue proprietà tutti i totali dei prezzi (totale da pagare, prezzo con e senza IVA, e costi di spedizione).	idt, idAddressDelivery, idAddressInvoice, idCart, idCurrency, idLang, idCustomer, idCarrier, module, idCurrentState, idShop, idShopGroup, payment, totalPaid, totalPaidReal, totalProductsNoTaxes, totalProductsWithTaxes;
Product	Prodotto. È la rappresentazione di un oggetto messo in vendita in uno store. Spesso utilizzato per definire l'oggetto di default, cioè quello senza combinazioni.	idt, name, price, defaultImageId, defaultCombinationId, description, images, combinations, isComplete, specificPrice, taxRate, taxId, productTag;
ProductOption	Opzione prodotto. Definisce una singola opzione di un prodotto. Nell'esempio dell'oggetto `Combination`, un'opzione è il colore. In questo caso l'oggetto iPhone potrà avere le proprietà: `name` = @"Corore"; `value` = @"Nero";	optionId, name, value;
SpecificPrice	Prezzo scontato. Differenziato in base a quantità e promozioni.	idProduct, idCombination, fromQuantity, reduction, reductionType, tag;
State	Provincia. Si riferisce ad una provincia. La proprietà idZone è necessaria per il calcolo delle tariffe dei corrieri.	idt, name, idZone, tag;

Tabella 2 - Schema Oggetti Domain

4.3 SCARICAMENTO ASINCRONO (LAZY LOADING)

In certi casi può capitare che l'utente si ritrovi davanti ad una schermata che apparentemente rimane bloccata. Questo accade quando l'applicazione svolge delle operazioni in background che richiedono parecchio tempo per essere completate. L'utente potrebbe spazientirsi e ciò verrebbe visto come una mancata efficienza dell'applicazione. Nel caso in cui è richiesto lo scaricamento della lista dei prodotti di una categoria, l'operazione potrebbe risultare particolarmente onerosa, in termini di tempo, se quest'ultima ne contiene molti. Tra l'altro, come detto precedentemente, l'applicazione rimarrebbe bloccata sino a quando tutti i prodotti non vengano scaricati.

Supponendo, ad esempio, che una categoria contenga 40 prodotti e che per scaricarne ognuno mediamente si impieghino 0,5 secondi, l'operazione richiesta impiegherebbe circa 20 secondi. Per le considerazioni fatte, questo tempo non è accettabile.

Per sbloccare l'applicazione può essere utilizzato un thread asincrono che continui a lavorare in background, dando libertà all'utente di continuare la navigazione. Questo thread da solo non è sufficiente a rendere l'applicazione efficiente, infatti, ci sarebbe sempre il problema della visualizzazione dei prodotti che verrebbero aggiunti alla vista solo quando il thread finisce di svolgere il proprio compito.

Una caratteristica particolarmente interessante del framework è quella di aggiungere i prodotti man mano che questi vengono scaricati. Questa caratteristica prende il nome di lazy loading (caricamento pigro).

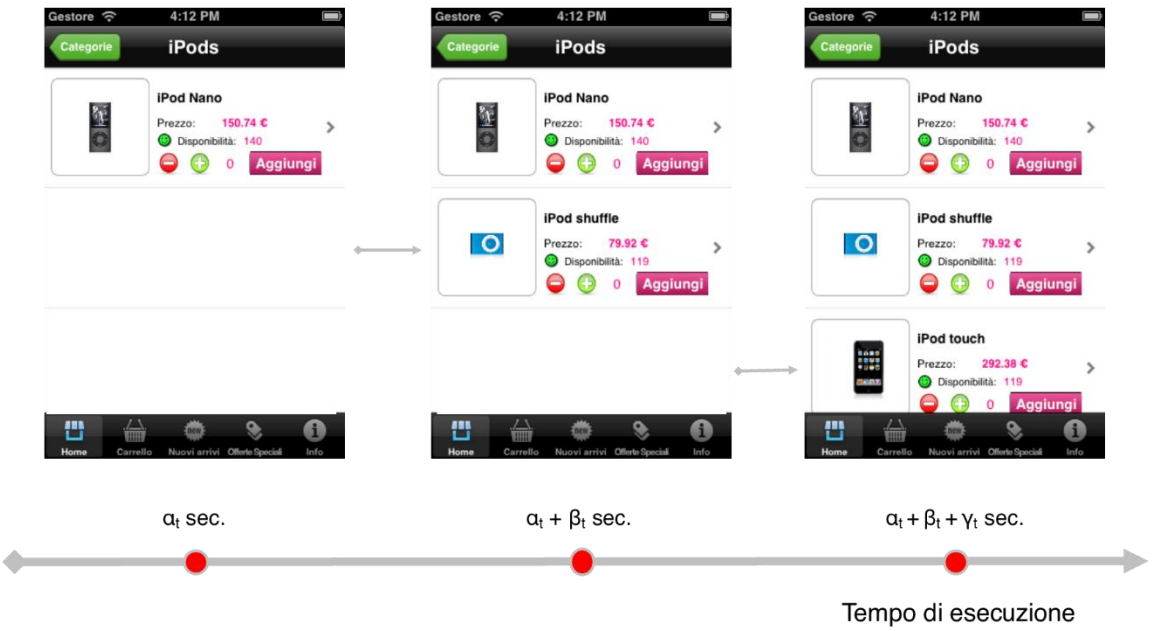

Figura 21 – Lazy Loading: schema temporale

In iOS i thread vengono gestiti dal GCD (paragrafo 1.4.2). Il problema principale del GCD, però, è che i thread asincroni una volta avviati non possono essere terminati da metodi esterni. Fortunatamente esiste una maniera per aggirare il problema.

Ritornando al framework quindi, la richiesta dei prodotti, è un'operazione asincrona che permette alla vista di non rimanere bloccata. Se i prodotti di una categoria, però, sono tanti e l'utente esce dalla vista, il thread che esegue il metodo `getProductsWithCategory` rimane in esecuzione. Nel caso un utente "entra ed esca" da più categorie prima che i thread abbiano finito il loro compito, questi aumenterebbero causando un blocco dell'applicazione, dovuto all'esaurimento di risorse. Ciò può essere risolto mediante l'utilizzo di una variabile di istanza booleana, `stopDownloadProducts`, che se settata a `YES` (true) faccia interrompere il ciclo `for` del metodo `getProductsWithCategory` e di conseguenza il thread associato. Nello specifico la variabile viene settata a `NO` (false) quando il metodo viene invocato ed a `YES` quando si esce dalla vista.

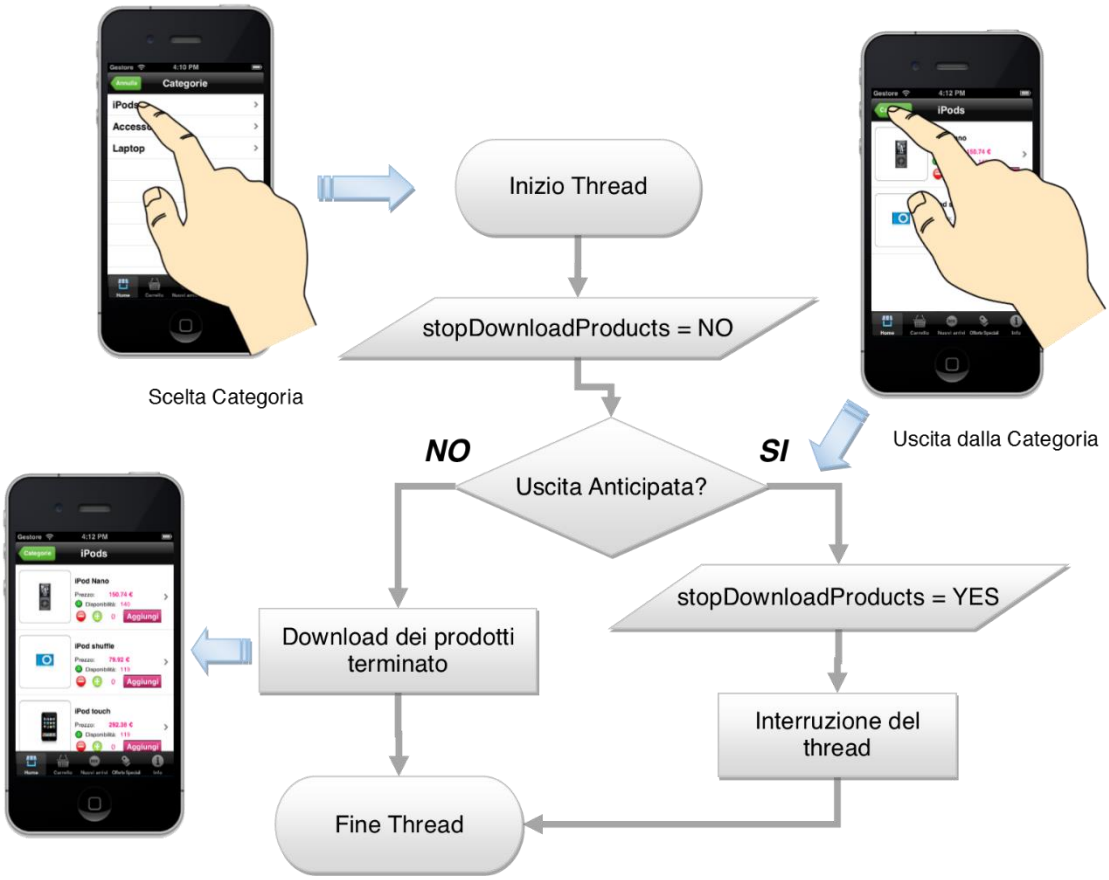

Figura 22 – Ciclo di vita del thread

Apparentemente può sembrare che il problema venga risolto, ma non lo è; infatti, se `getProductsWithCategory` viene interrotto, la lista relativa ai prodotti per categoria richiesta alla cache (`productsCacheByCategory`) non è `nil` (`NULL`) e quando si rientrerà nella categoria, il metodo ritornerà solo una lista parziale. La soluzione adottata per risolvere questo problema è quella di introdurre un `NSMutableDictionary` contenente delle coppie "*id category*", "*status*", dove il primo valore è l'id della categoria e il secondo una stringa che può essere `@"partial"` o `@"full"`. Grazie a questo approccio tutto fila liscio,

infatti, se l'id della categoria è associato a @"full", il metodo ritorna semplicemente la lista di prodotti precedentemente salvati nella cache, viceversa, se è @"partial", recupera la lista parziale e continua a scaricare i prodotti mancanti. Quanto detto implica una corretta gestione delle risorse senza inutili sprechi di banda, memoria e impiego del processore.

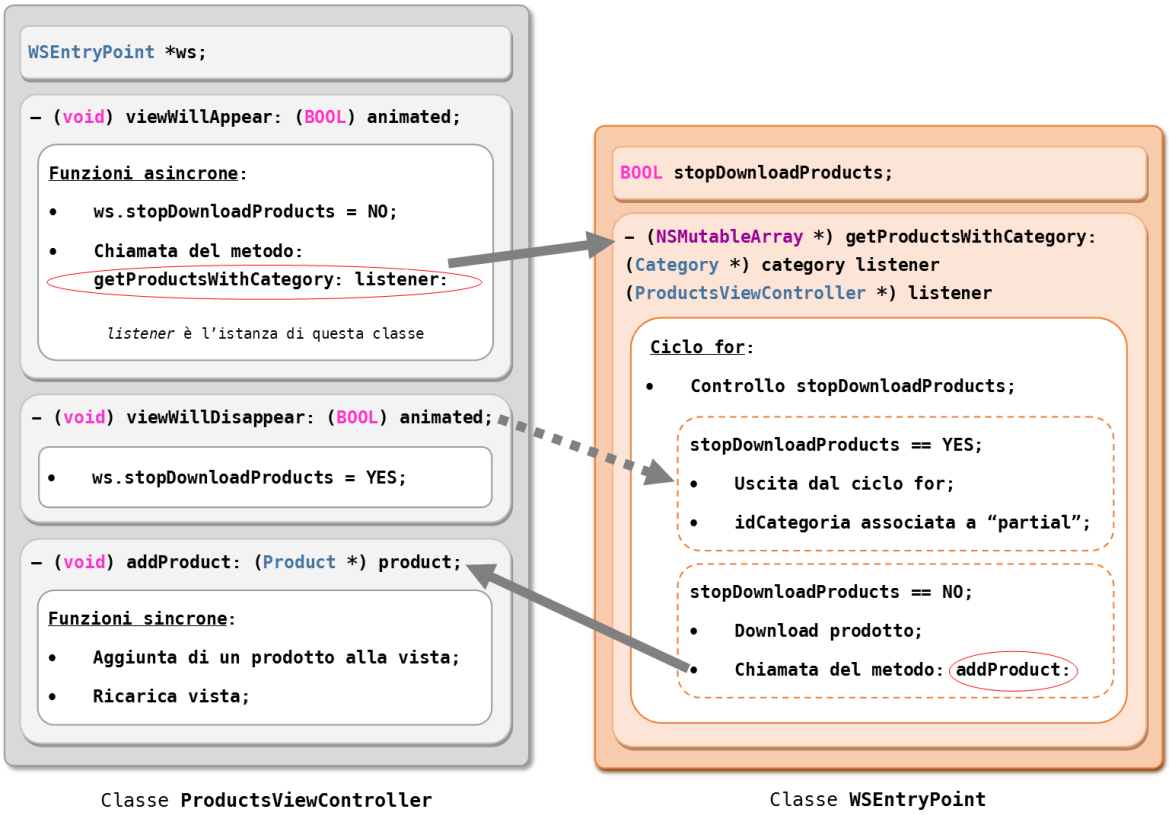

Figura 23 – Funzionamento del Lazy Loading

4.4 CACHING

L'utilizzo di una memoria cache, o comunemente cache, è una delle tecniche informatiche che permettono di immagazzinare dati per poi usarli in un secondo momento. Nonostante sia una caratteristica molto importante, che permette di velocizzare e ridurre l'utilizzo delle risorse di un dispositivo, non è visibile agli occhi di un utilizzatore.

Esistono vari tipi di cache, che si distinguono tra loro per il contesto dove vengono utilizzate. Nel corso di questo elaborato non verranno descritti, si analizzeranno solo quelli relativi al framework di PrestaShop.

Nel framework esistono due tipologie di cache, una per gli oggetti del gruppo `Domain` e l'altra per le immagini. La prima è costituita da delle variabili d'istanza della classe `WSEntryPoint`, la seconda è fisicamente un'area di memoria sita all'interno della sandbox dell'applicazione e gestita dalla classe `ImageCache`.

Quando è necessario accedere ad un dato verrà prima cercato nella cache. Se questo dato è presente e valido, verrà utilizzata la sua copia precedentemente salvata, viceversa, verrà scaricato dal web server e memorizzato nella cache, nel caso possa servire in futuro.

4.4.1 Cache oggetti domain

Gli oggetti `domain`, come visto nel paragrafo 4.1.2, sono delle classi che permettono di memorizzare un tipo di dato specifico. Questo dato può essere una categoria, un ordine o l'indirizzo civico di un utente. La cache associata a questi elementi è differente da quella delle immagini (paragrafo successivo). Questa si basa sulle sole variabili di tipo `NSMutableArray` o `NSMutableDictionary`. Il motivo di usare variabili di tipo "mutable" è legato al fatto che ogni cache viene aggiornata costantemente ogni qualvolta l'utente effettua un'operazione di richiesta. Per questo motivo, l'aggiunta di nuove voci impone che queste variabili siano modificabili (in objective-c esistono gli stessi tipi di dato non editabili: `NSArray` e `NSDictionary`). `NSMutableArray` è praticamente una lista di oggetti, che a differenza di altri linguaggi come Java, può essere eterogenea, cioè può contenere oggetti di vario tipo (`int`, `float`, `NSString`, etc); `NSMutableDictionary`, invece, una lista di coppie nome-valore, corrispettivo di `HashMap` di Java.

Le cache di tipo `NSMutableArray` sono tipicamente quelle associate a raggruppamenti di oggetti. Ad esempio, la cache dei "nuovi prodotti", se richiesta la visualizzazione degli stessi, previo precedentemente immagazzinamento, li restituisce tutti e non solo uno specifico. Ciò ha senso perché in certi casi è necessario utilizzare l'intera lista e non solo uno di essi. Se volessimo accedere ad un unico oggetto, comunque, c'è un'altra cache che si occupa di tale compito.

Le cache di tipo NSMutableArray sono, quindi:

- categoryCache (categorie di prodotti);
- newProductsCache (nuovi prodotti);
- specialOffersCache: (offerte speciali);
- carriersCache: (metodi di spedizione).

Le cache di tipo NSMutableDictionary sono legate ad insiemi di oggetti che possono essere utilizzati singolarmente. Proprio per questo motivo sono costituite da delle coppie nome-valore, dove il nome sarà l'id associato ad un oggetto e il valore sarà l'istanza della classe che lo rappresenta. Riprendendo l'esempio precedente, se volessimo accedere ad un singolo elemento della cache newProductsCache potremmo farlo chiamando in causa la cache productsCache ed utilizzare l'id prodotto, dato che ques'ultimo è stato comunque immagazzinato in entrambe le cache quando è stato scaricato la prima volta.

Le cache di tipo NSMutableDictionary sono:

- productsCache (prodotto tramite il suo id);
- productsCacheByCategory (prodotti tramite l'id della categoria);
- productOptionsName (nome opzione prodotto tramite l'id dell'opzione);
- productOptionsCache (opzione prodotto tramite il suo id);
- productsSpecificPrices (prezzo di un prodotto tramite l'id del prodotto);
- combinationsSpecificPrices (prezzo di una combinazione del prodotto tramite l'id della combinazione);
- taxRates (tasse statali tramite il loro id);
- addressesCache (indirizzi civici tramite l'id dell'utente);
- countriesCache (nazione tramite il suo id);
- statesCache (provincia tramite il suo id);
- countriesStatesCache (provincie tramite l'id di una nazione);
- productsByCategoryFullOrPartial (@"full" o @"partial" tramite l'id della categoria).

4.4.2 Cache Immagini

La classe che gestisce la cache delle immagini prende il nome di ImageCache. Essa è separata dal WSEntryPoint poiché la cache immagini, a differenza della cache degli oggetti Domain che viene eliminata ogni qualvolta l'applicazione viene chiusa, memorizza i dati nella memoria del dispositivo. La classe ImageCache implementa, quindi, i metodi necessari per coordinare le operazioni. Questi metodi possono essere implementati anche all'interno del WSEntryPoint, ma ciò lo appesantirebbe ulteriormente di linee di codice, rendendolo meno leggibile.

Salvare le immagini è necessario per velocizzare l'applicazione, dato che rispetto agli oggetti di Domain sono più pesanti in termini di dati.

Per immagazzinare e successivamente recuperare le immagini dalla memoria, la classe `ImageCache` conterrà dei metodi specifici.

La classe in questione contiene le variabili d'istanza: `cacheDir`, `imagesIndex`, `cacheSize` e `maxCacheSize`:

- `cacheDir` è una stringa contenente l'indirizzo della cache immagini (nello specifico "`<rootAppPath>/Library/Caches/Products Images`");
- `imagesIndex` è un `NSDictionary` di coppie id-valore contenente due variabili: `imageId` (identificativo dell'immagine) e `imageSize` (grandezza in bit);
- `cacheSize`, definita di tipo `long`, rappresenta lo spazio occupato dalle immagini nella cache;
- `cacheSizeMax`, anch'essa definita come `long`, è la grandezza massima della cache.

Sfruttando il fatto che l'objective-c deriva dal c, alcune variabili costanti sono definite nel preprocessore, grazie alla direttiva `#define`.

```
#define FILE_EXTENSION @"jpg"
#define CACHE_MAX_SIZE_BYTES_EXTERNAL (50 * 1024 * 1024) // 50 MB
#define CACHE_MAX_SIZE_BYTES_INTERNAL (10 * 1024 * 1024) // 10 MB
#define FLUSH_RATE 10
```

Figura 24 – Le variabili costanti della classe `ImageCache`

I metodi della classe `ImageCache` sono 4 e sono definiti tutti come metodi d'istanza.

Il primo, `initWithAppCacheDir:andMaxCacheSize:`, permette di istanziare la classe mediante l'uso di due parametri: `appCacheDir` che definisce un nome arbitrario da assegnare alla directory e `_maxCacheSize` che definisce la grandezza massima della cache. In linea generale, il metodo si occupa d'istanziare la classe, rigenerando la cache ogni qualvolta l'applicazione viene lanciata. Ciò è indispensabile perché, ricordando quanto detto prima, gli oggetti di `Domain` vengono distrutti quando si chiude l'applicazione e quindi è fondamentale ricreare i vari indici della cache;

Il secondo, `hasImage:`, controlla se l'id dell'immagine è contenuto all'interno dell'indice, quindi se l'immagine di un prodotto è presente o no nella cache. Esso restituisce una variabile booleana che indica l'esito del controllo;

Il terzo, `getCachedImage:`, recupera un'immagine in base all'id fornito come parametro. I casi che possono verificarsi quando s'invoca questo metodo sono due: 1) L'immagine non è presente nella cache; 2) L'immagine è presente. Nel primo caso il metodo ritorna un valore `nil`, nel secondo, invece, la situazione cambia: se l'immagine è disponibile verrà restituita normalmente, altrimenti, se non è presente, ma esiste il suo id nell'indice della cache, verrà sollevata un'eccezione che verrà opportunamente gestita dallo stesso metodo.

Infine, il quarto e ultimo metodo, `putImage:andImageData:`, permette di inserire un'immagine nella cache. Questo metodo riceve due parametri: l'id dell'immagine e i dati

ad essa associati di tipo `NSData`. Quando si salvano dei dati, bisogna innanzitutto controllare che la grandezza del file da memorizzare non superi lo spazio disponibile riservato alla cache. Una funzione di `putImage:andImageData:` è proprio quella di controllare se l'immagine può essere salvata senza operazioni preliminari, cioè se lo spazio a disposizione è abbastanza grande. In questo caso l'immagine viene salvata senza problemi, viceversa, salvata previa eliminazione delle immagini più vecchie per aumentare lo spazio disponibile.

4.5 ANALISI A RUN-TIME, CONSUMI MEMORIA, RITARDI…

L'efficienza e la reattività di un'app sono alcune delle caratteristiche che qualunque programmatore deve tenere in considerazione al fine di presentare un ottimo prodotto all'utilizzatore finale.

Per far ciò deve analizzare tutti quegli elementi che permettano di migliorare le performance di un'app, a partire dall'ottimizzazione del codice fino ad arrivare a risolvere tutte quelle problematiche che rallentino l'esecuzione a runtime.

Uno degli aspetti principali quando si sviluppa un'app, che utilizza la rete, è quello di considerare che non tutti i ritardi dipendono dall'applicazione, alcuni infatti possono essere generati dalla congestione di qualche nodo della rete, da errori di ricezione o invio dei pacchetti, dal server non rintracciabile, etc.

Nel corso di questo paragrafo si prenderanno in esame tutti i dati che riguardano l'app di test, sviluppata per testare il framework descritto precedentemente, iniziando dai tempi di avvio e l'impegno delle risorse, fino ad arrivare ai dati ricavati dall'analisi di vari scenari d'uso.

Figura 25 – Scenario di avvio

Grazie a varie ottimizzazioni ed all'ausilio delle storyboard, l'intervallo di tempo tra l'avvio dell'app e l'operatività della stessa è abbastanza piccolo. Il seguente grafico mostra proprio il tempo necessario affinché l'applicazione sia pronta all'utilizzo dell'utente.

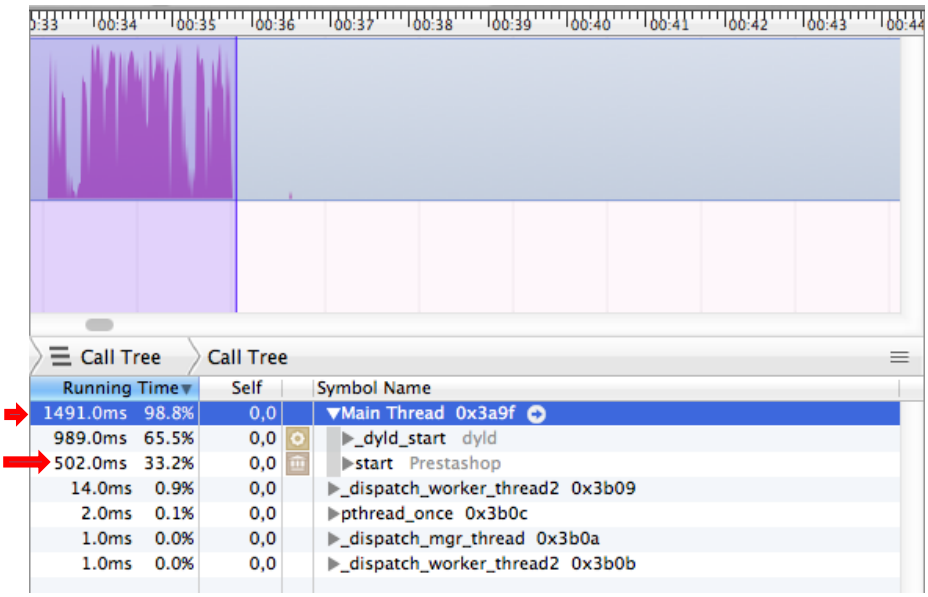

Figura 26 – Tempo di avvio dell'app

Tale tempo, mediamente, è di circa 1500 ms e varia in base ai carichi di lavoro del dispositivo. Fortunatamente iOS, a livello di sistema, è molto efficiente e quindi in tutti i casi questo tempo non aumenta considerevolmente. La figura 26 mostra un'analisi sul tempo d'avvio dell'app che, in questo caso, è 1491 ms: 989 ms rappresenta il tempo che impiega iOS per generare un qualunque main thread di un'app (circa 1 secondo) mentre i restanti 502 ms quelli utilizzati per inizializzare sia le viste che gli oggetti del framework. In quest'ultimo tempo è incluso anche quello necessario per ricreare la cache immagini che, indipendentemente dal numero di immagini presenti, impiega circa 25 ms, esso è possibile ricavarlo dalla lettura dei log dell'applicazione (i ritardi delle viste dei seguenti scenari d'uso vengono ricavati in questa maniera, cioè tramite l'inserimento nel codice di stringhe, visualizzate nel log, che permettono di estrarre il tempo di caricamento).

La figura seguente, invece, mostra l'allocazione degli oggetti in memoria all'avvio dell'applicazione. Il dato più rappresentativo è quello evidenziato da un riquadro rosso e indica lo spazio totale allocato, 1,13 MB.

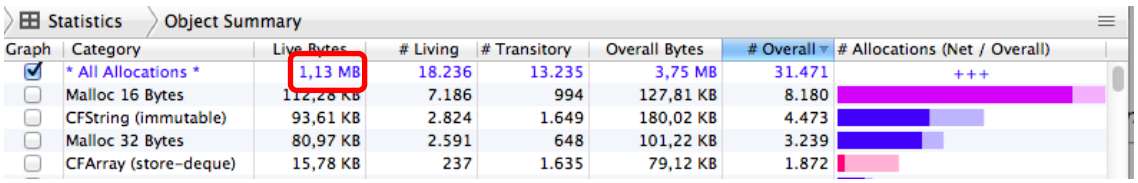

Figura 27 – Allocazione di memoria all'avvio

SCENARI UTENTE:

La prima serie di scenari che andremo ad analizzare sono quelli relativi al mantenimento dello stato di un utente. Questa funzionalità ricopre un ruolo fondamentale nell'applicazione dato che alcune funzioni dell'app, come il carrello della spesa, non sono utilizzabili da utenti anonimi. In termini di consumo di memoria ed impiego delle risorse, questi scenari non generano eventi particolarmente onerosi (nell'ordine di 50-70 ms).

Nel caso in cui un utente voglia effettuare il login potrà farlo premendo il bottone "Accedi" sulla Navigation bar (la barra superiore) della home del programma. Qui si presenterà la schermata di accesso dove immettere email e password. Alla conferma dei dati, se l'operazione ha avuto un riscontro positivo, comparirà la finestra di dialogo che avviserà l'utente dell'avvenuto accesso. Premendo su "OK" si ritornerà automaticamente alla home. Adesso l'applicazione riconosce l'utente connesso, ed inoltre, d'ora in avanti, il carrello della spesa sarà abilitato, ovviamente fino a quando l'utente non effettua il logout.

Figura 28 – Scenario di login

Se un utente, invece, non è in possesso di credenziali e vuole registrarsi potrà farlo usando l'apposito bottone nella schermata di login. Anche in questo caso i tempi e i consumi sono irrisori, l'unica differenza la si ha nella comparsa della vista di registrazione che impiega qualche millisecondo in più di quella di accesso. Ciò è dovuto alla presenza di più oggetti, ma questo non influenza minimamente le prestazioni, e l'utente, dati i tempi trascurabili, non si accorgerà di nulla. La schermata di registrazione implementa controlli sulla correttezza dei dati e che siano stati inseriti tutti quelli obbligatori. Per garantire la registrazione dei soli utenti maggiorenni, il selettore della data di nascita non permette di inserire le date comprese nell'intervallo di 18 anni precedente all'attimo in cui si sta selezionando la data. Anche in questo caso, se la registrazione ha successo, comparirà la finestra di dialogo e premendo su "OK" si ritornerà alla home.

Figura 29 – Scenario registrazione nuovo utente (la freccia il movimento del dito verso l'alto)

SCENARI NAVIGAZIONE:

Dalla home è possibile esplorare il catalogo dei prodotti di uno store semplicemente premendo il bottone "Categorie" che avvierà l'operazione di recupero delle stesse. Una categoria può contenere sottocategorie o l'elenco dei prodotti. Nel primo caso, premendo su un nome, verrà effettuata la medesima operazione di recupero delle categorie, nel secondo verrà mostrata una schermata vuota che aggiunge ogni oggetto alla vista man mano che questo viene scaricato (lazy loading). Da un'analisi accurata dei dati, si vede che la memoria allocata cresce sia entrando in una categoria che contiene sottocategorie sia in una che contiene la lista dei prodotti. Il perché di questo aumento è semplice, ciò è dovuto alla cache, che essendo formata da tipi di dato "mutable", quindi non completamente allocati, richiede ulteriore spazio quando si aggiungono nuovi dati. Verrebbe da chiedersi cosa succede se la cache si riempie troppo, la risposta è semplice, il sistema operativo si occupa di tutto ed attua dei meccanismi per liberare le risorse.

Figura 30 – Scenario esplorazione categorie

I tre grafici seguenti mostrano l'efficienza dell'app e alcuni confronti: nel primo, rappresentante lo scenario di figura 30, si nota che l'app libera automaticamente la memoria (frecce rosse) e che l'allocazione cresce per via dell'aumento della cache (il valore finale della memoria in questo caso è 4,94 MB); nel secondo (figura 32) si può notare la differenza, in termini di tempo, tra l'uso o no della cache; nell'ultimo (figura 33) abbiamo il confronto tra l'utilizzo dello scaricamento asincrono dei prodotti (con lazy loading) con quello sincrono (senza lazy loading).

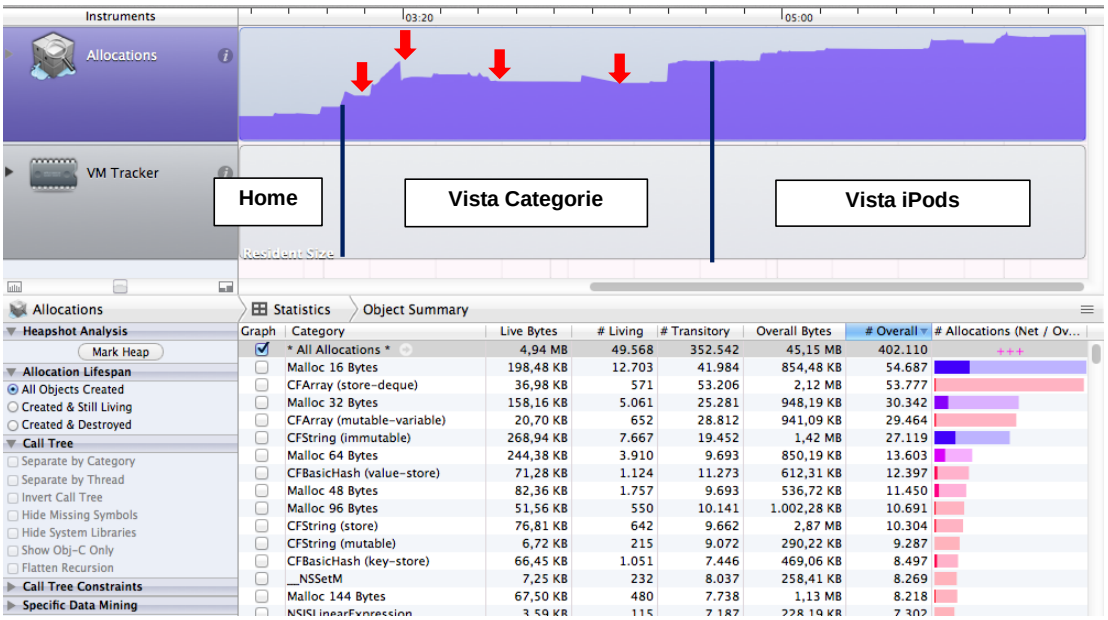

Figura 31 – Gestione efficiente della memoria e crescita allocazione cache

Figura 32 – Tempi di accesso alle liste di prodotti per categoria senza cache (Rosso) e con cache (Verde)

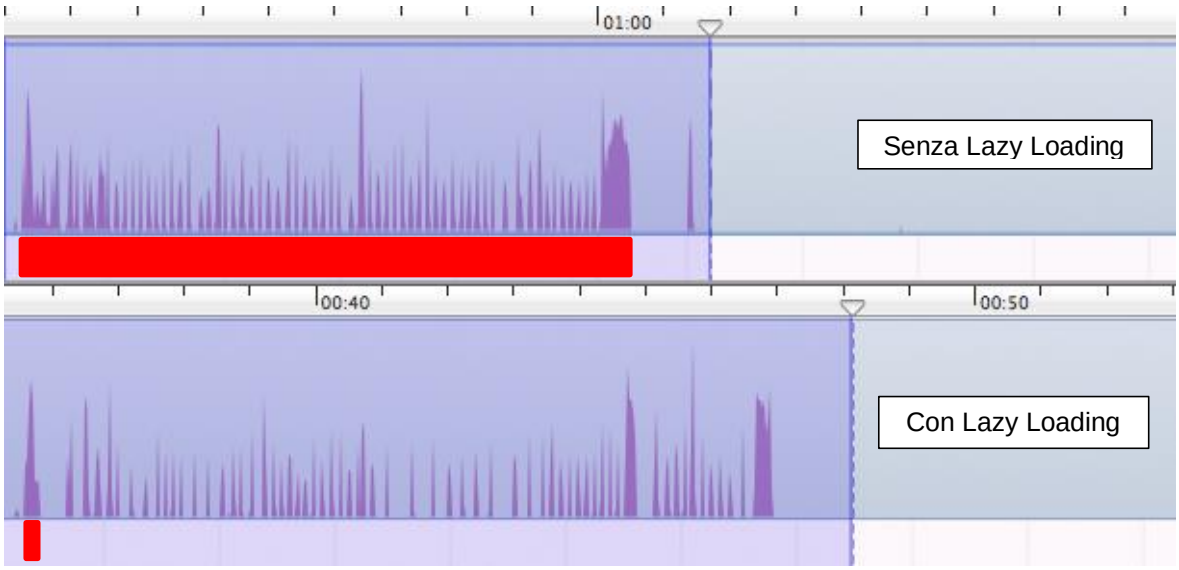

Figura 33 – Confronto con e senza lazy loading (in rosso l'intervallo in cui l'utente rimane bloccato)

L'utilizzo del Lazy Loading, come si vede dal grafico di figura 33, aumenta leggermente i ritardi dovuti alla ricorsività dell'aggiunta di ogni prodotto. Nel caso senza, i prodotti vengono inseriti tutti alla fine quindi c'è solo un'operazione di aggiunta. Se consideriamo, invece, il caso con Lazy Loading avremo tante operazioni di aggiunta per quanti sono i prodotti contenuti in una categoria. Sebbene la scelta di non utilizzare il Lazy Loading possa sembrare conveniente in termini di ritardi, non lo è; il non utilizzo di tale funzionalità, infatti, bloccherebbe l'applicazione fino allo scaricamento totale dei prodotti, mentre in caso di utilizzo, l'applicazione si sbloccherebbe "immediatamente", dato che il tipo di operazione è di tipo asincrona non bloccante e l'unico tempo di attesa sarebbe quello di caricamento della vista, mediamente 170 ms. Nei due casi proposti dal grafico, se non si utilizzasse il Lazy Loading, l'utente rimarrebbe bloccato per circa 11 secondi, tempo non accettabile anche considerando che lo stesso varia in base al numero di prodotti scaricati (nell'esempio della figura vengono scaricati solo tre prodotti), mentre nel caso opposto, nonostante i secondi per completare l'operazione siano circa 13, l'utente potrebbe compiere altro, come decidere di ritornare indietro o selezionare uno dei prodotti già visualizzati, quasi istantaneamente.

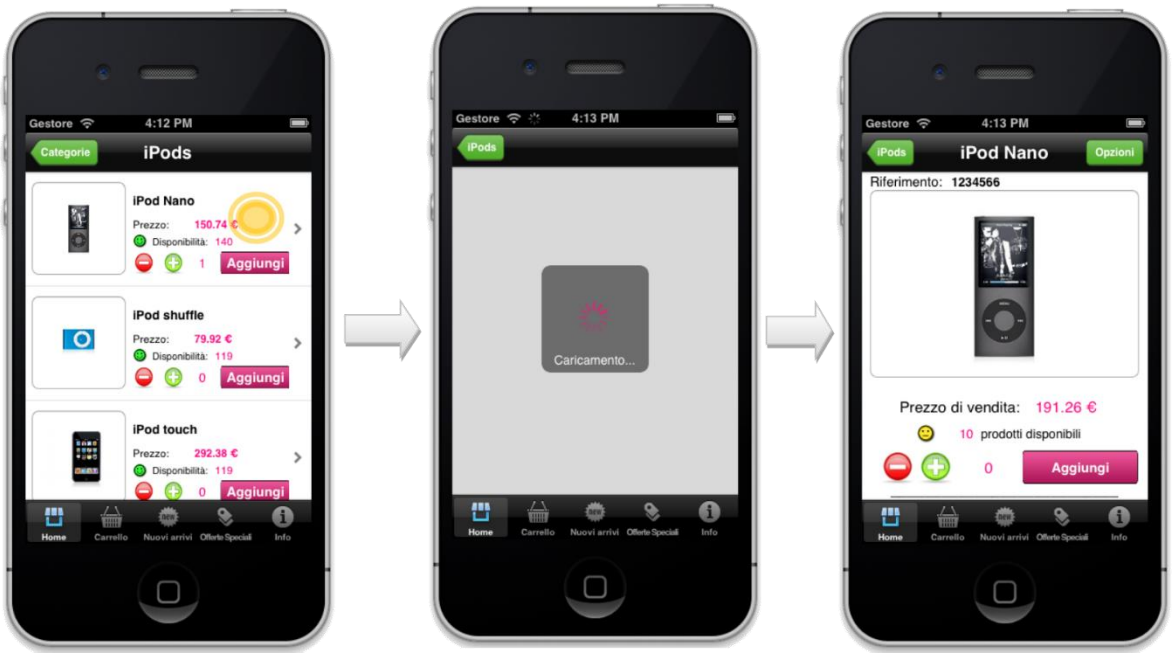

Figura 34 – Scenario dettaglio prodotto

Lo scenario di figura 34 mostra la selezione di un prodotto, e precisamente di un *iPod Nano*.

La schermata di caricamento è necessaria perché questa operazione può richiedere anche molto tempo, ciò è dovuto al download di tutte le informazioni appartenenti ad un prodotto. Nel caso in esame, l'*iPod Nano*, venduto dallo store, contiene varie configurazioni quali colore e capacità di memoria. Tali configurazioni comporteranno il download di ulteriori immagini e di prezzi differenziati per ogni configurazione.

Nelle figure 35 e 36 esempi di scelta di opzioni singole e multiple. Nel primo scenario abbiamo la scelta di una singola opzione: *Colore*; nel secondo tre opzioni: *Processore*, *Colore* e *Disco Rigido*.

Figura 35 – Scenario cambio opzione prodotto

Figura 36 – Scenario opzioni multiple

SCENARI CARRELLO:

Al fine di mantenere la sincronicità tra l'applicazione e il web server, anche l'operazione di aggiunta o rimozione di un prodotto dal carrello richiede un impegno di risorse. Tutto sommato, però, queste operazioni non richiedono tempi elevati. L'aggiunta di un prodotto al carrello è un'operazione che impiega circa 170 ms per essere effettuata, ciò perché è un'operazione di scrittura che richiede l'invio di un file XML; d'altro avviso, invece, è la rimozione di un prodotto il cui tempo è veramente piccolo (circa 20 ms).

Figura 37 – Scenario aggiunta prodotto al carrello

Figura 38 – Scenario svuotamento carrello

Il tempo impiegato per aprire il carrello, cliccando sull'apposito bottone della toolbar (barra inferiore) varia in base al contenuto, dato che vengono gestite le singole righe contenute nello stesso. Riassumendo, più righe ci sono in un carrello, e per riga s'intende un prodotto aggiunto indipendentemente dalla quantità, maggiore sarà il tempo per visualizzazione il contenuto dello stesso. Nell'esempio di figura 38, il caricamento del primo riquadro, formato da tre righe, ha richiesto circa 1220 ms.

SCENARI ORDINE:

L'ultima serie di scenari che andremo a vedere riguarda la parte conclusiva del processo di acquisto, l'ordine, che è poi il fine principale dell'applicazione.

Per accedere alla schermata di un ordine bisogna innanzitutto aver aggiunto dei prodotti al carrello, dopodiché verrà abilitato il pulsante che rimanda alla schermata relativa agli ordini. Il caricamento di questa schermata richiede tempi superiori a quelli delle altre schermata, stimati nell'ordine dei secondi. Ciò è dovuto a tutte le operazioni di calcolo dei prezzi, dello scaricamento dei metodi di spedizione e pagamento, ed ai componenti presenti nella vista. Nello scenario illustrato in figura 39 vengono rappresentate varie parti della vista Ordine: nella parte iniziale ci sarà un riepilogo dei prodotti acquistati, disposti incolonnati con relative quantità e prezzi; scendendo (secondo riquadro) troveremo l'indirizzo di spedizione scelto dall'utente, qualora ne fosse presente almeno uno (è possibile cambiarlo premendo su di esso). In caso l'utente non abbia un indirizzo può aggiungerlo premendo sul bottone relativo (terzo riquadro).

Nella schermata, inoltre, è presente uno switch che permette all'utente di richiedere la fattura in un indirizzo differente da quello di spedizione, ed abilitare il pulsante relativo alla scelta dell'indirizzo alternativo.

Figura 39 – Scenario ordine

Al di sotto degli indirizzi ci saranno due selettori che permettono di scegliere il metodo di spedizione e quello di pagamento. Riguardo quest'ultimo, l'app mette a disposizione tre moduli (figura 40): Contrassegno, Bonifico e PayPal. La scelta della forma di pagamento può comportare l'aggiunta di ulteriori costi al totale delle spese (se presenti) definiti nelle API del negozio online e calcolati dal framework.

Figura 40 – Scenario scelta metodo di pagamento

Nel caso l'utente scegliesse PayPal come forma di pagamento, varia anche il bottone di conferma (figura 41). PayPal mette a disposizione delle classi, create ad hoc, per gestire i pagamenti che nelle specifiche di funzionamento richiedono proprio l'utilizzo del bottone personalizzato. A differenza di ciò che si può pensare questo bottone non è semplicemente un'immagine, ma un oggetto Objective-C.

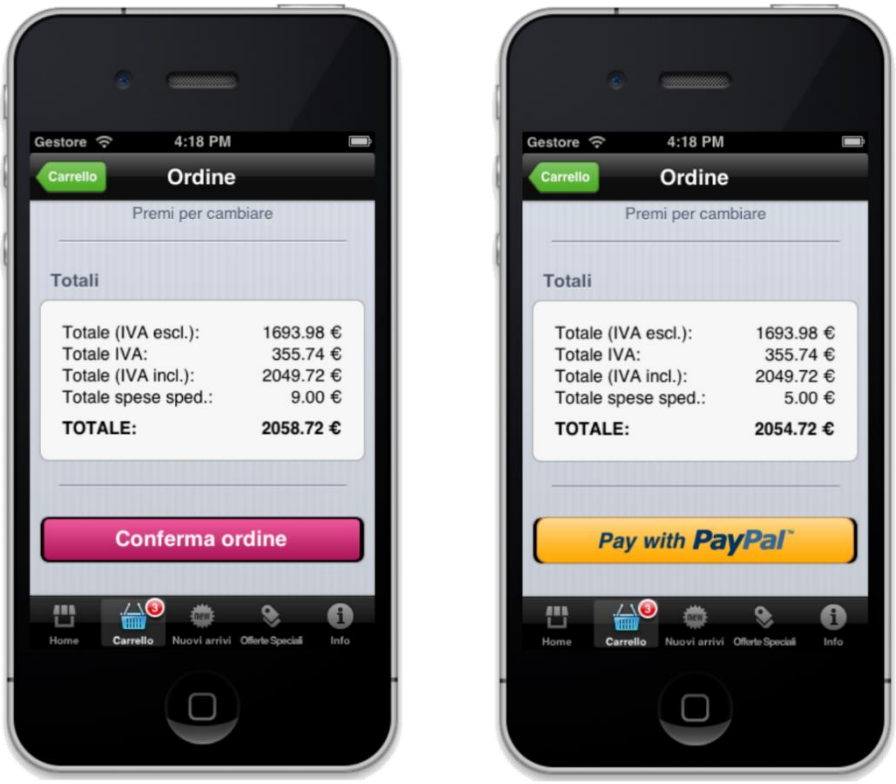

Figura 41 – Conferma ordine: Contrassegno e PayPal

L'ultima parte della schermata, infine, mostra il riepilogo dei totali e il suddetto bottone di conferma. Nel caso di pagamenti via bonifico o contrassegno, l'operazione di conferma impiega circa 390 ms per essere completata; nel caso di PayPal la situazione cambia. Il modulo di pagamento di PayPal impiega molto più tempo per essere istanziato (stimato a più di 5 secondi), ciò non dipende dall'applicazione, ma dalla logica delle classi fornite e pertanto non può essere analizzato o ottimizzato.

CONCLUSIONI

Negli ultimi anni la crescita degli acquisti online anche nel nostro paese ha riscosso parecchio successo e ciò è stato possibile grazie allo sviluppo di nuove tecnologie applicate al commercio elettronico ed all'aumento dei negozi virtuali. Fino a poco tempo fa la realizzazione di un sito di e-Commerce richiedeva risorse che solo una grande azienda poteva permettersi; oggi invece è tutto più semplice ed economico, grazie anche alle piattaforme che permettono di creare negozi online con pochi click.

Le opportunità date da un sito di vendita sono tante, l'importante è che l'azienda si sappia presentare al meglio trasmettendo agli utenti un senso di serietà e professionalità che, ai fini dell'acquisto, sono caratteristiche più determinanti del prezzo. Il consumatore deve potersi orientare facilmente all'interno del catalogo prodotti. Questi ultimi devono essere suddivisi in più categorie e ogni prodotto deve essere dettagliatamente descritto e accompagnato da una o più immagini.

L'aumento delle vendite dei nuovi dispositivi mobili sul quale girano sistemi operativi come Android, Windows Phone e iOS e sui quali sono installabili applicazioni di vario genere, compresi i browser per la navigazione in Internet, ha contribuito alla diffusione dell'e-Commerce. All'utente viene data la possibilità di acquistare i beni ovunque esso si trovi e direttamente dal suo terminale. Tuttavia, in certi casi, tale acquisto può essere complicato date le dimensioni ridotte degli schermi degli smartphone, ragion per cui spesso ogni negozio permette di scaricare dagli store ufficiali di questi dispositivi un'applicazione nativa, ottimizzata per le risoluzioni dei display e che permetta di effettuare queste transazioni in maniere semplice.

Il progetto iniziale di questo elaborato prevedeva proprio lo sviluppo di un framework per iOS che consentisse agli sviluppatori di utilizzare le funzionalità offerte dalla piattaforma Prestashop (nota piattaforma di e-Commerce) su un'applicazione nativa per i dispositivi Apple.

Per soddisfare i requisiti richiesti sono state sviluppate delle librerie di funzioni che hanno permesso di far comunicare il framework con un server remoto sul quale era presente un'istanza di un negozio virtuale. Nello specifico, dopo la lettura della documentazione ufficiale di Prestashop, è stato effettuato il porting della libreria PrestashopRESTLib, appartenente ad un progetto già esistente per Android e contenente i metodi di comunicazione con le API della piattaforma. Per garantire il corretto funzionamento della risultante classe Objective-C (linguaggio di programmazione per iOS), sono state effettuate delle prove usando gli Unit Test di XCode (IDE), utilizzati proprio per la fase di debug, e soltanto dopo aver appurato che questa libreria funzionasse correttamente è stata implementata una classe WSEntryPoint, contenente tutte le funzionalità di alto livello del framework. Infine, per testare ed analizzare il comportamento a runtime, è stata creata un'app ad hoc che utilizzasse tutte le funzionalità offerte dal framework. I risultati ottenuti sono stati soddisfacenti. L'app, ed in particolare il framework, rispecchiano le specifiche richieste, tuttavia sono stati necessari opportuni accorgimenti, come ad esempio la correzione di alcuni bug, dovuti all'implementazione errata di qualche metodo: tipicamente errori di estrazione degli elementi dai file XML ricevuti dal server o altri errori semantici nei

metodi (per citarne uno, getCarriers di WSEntryPoint non ritornava solo i corrieri disponibili per la destinazione selezionata dall'utente, ma l'intera lista di quelli utilizzati dal negozio), ed di altre problematiche legate all'efficienza dell'applicazione, come l'ottimizzazione del codice al fine di utilizzare la memoria del dispositivo il meno possibile. Trattandosi di un framework di e-Commerce, una problematica affrontata è stata quella di risolvere il problema dello scaricamento delle liste di prodotti che bloccavano l'applicazione fino al completamento del download di tutti quelli associati ad una categoria. La risoluzione di tale inefficienza ha permesso di estendere le funzionalità del framework, introducendo il lazy loading per lo scaricamento asincrono dei prodotti (aggiunta dei prodotti alla vista utente man mano che questi vengono ricevuti) che sblocca quasi istantaneamente l'applicazione altrimenti bloccata dal completamento dell'operazione.

Il risultato finale è stato un framework che fa un uso corretto della memoria, senza inutili sprechi, e che soprattutto è rapido a runtime, ovviamente nei limiti dei tempi imposti dalla connessione di rete. Tuttavia, tale lavoro può essere ulteriormente migliorato attraverso varie ottimizzazioni nei sorgenti.

Come per tutte le innovazioni non ci sarà mai un punto d'arrivo e questo progetto si pone come base per progetti futuri che possano estendere le funzionalità offerte da questo framework. Una possibile estensione, ad esempio, può essere l'aggiunta di funzionalità come i feedback e i commenti degli utenti sia su un negozio virtuale che sui prodotti acquistati. Tale funzionalità, peraltro, è spesso utilizzata nei siti come certificazione di qualità. Un utente che visualizza dei commenti positivi potrebbe essere più invogliato all'acquisto e tra l'altro sarebbe più sicuro di non incorrere in "fregature". Altro possibile spunto, forse un po' "ambizioso", può essere l'utilizzo di immagini tridimensionali per i prodotti che permetterebbe agli utenti di avere una visione più completa del prodotto, dato che a differenza delle immagini bidimensionali offrono più dettagli. Seppur questa caratteristica sia un po' difficile da realizzare, nulla vieta che non sia attuabile, anche perché i moderni dispositivi oramai offrono prestazioni elevate paragonabili ai computer di qualche anno fa.

BIBLIOGRAFIA

- Allan Alasdair, *Learning iPhone Programming*, O'Reilly, 2010;
- Amedeo Enrico, *Objective-C*, Apogeo Editore, 2011;
- Grönlund Hans-Eric – Francis Colin – Grimes Shawn, *iOS 6 Recipes – A Problem-Solution Approach*, Apress, 2012;
- Kelley Jeff, *Learn Cocoa Touch for iOS*, Apress, 2012;
- Kochan Stephen G., *Programming in Objective-C,* 4th Ed., Addison-Wesley, 2012;
- Mark David – Nutting Jack – LaMarche Jeff – Olsson Fredrik, *Beginning iOS 6 Development – Exploring the iOS SDK*, Apress, 2012;
- Nahavandipoor Vandad, *iOS 6 Programming Cookbook*, O'Reilly, 2013;
- Picchi Andrea, *Objective-C 2.0 per iOS e OS X*, Edizioni FAG Milano, 2013;
- Sadun Erica, *The Core iOS 6 Developer's Cookbook*, Addison-Wesley, 2012;

- *https://developer.apple.com/library/ios/navigation*
  (Documentazione ufficiale su iOS ed i framework di Apple)
- *http://doc.prestashop.com/display/PS15/Developer+Guide*
  (Documentazione ufficiale Prestashop 1.5)
- *http://www.html.it/guide/guida-objective-c*
- *http://iosdeveloperzone.com/2011/05/05/introduction-to-properties-in-objective-c*
- *http://www.devapp.it/wordpress/corso-di-programmazione-cocoa-touch*
- *http://www.devapp.it/wordpress/xcode-4-scopriamo-le-novita-introdotte-con-il-nuovo-ambiente-di-sviluppo-apple.html*
- *http://cocoadevcentral.com/d/learn_objectivec*